A Roadmap for Privacy Preserving Speech Processing

Cornelius Glackin[1], *Gerard Chollet*[1], *Nazim Dugan*[1], *Nigel Cannings*[1], *Julie Wall*[2], *Shahzaib Tahir*[3], *Indranil Ghosh Ray*[3], *Muttukrishnan Rajarajan*[3], *Ryan Falkner*[4], *Atta Badii*[4]

[1]Intelligent Voice Ltd., [2]University of East London, [3]City University London, [4]University of Reading

***Abstract*— This paper presents an overview of a strategy for enabling speech recognition to be performed in the cloud whilst preserving the privacy of users. The strategy advocates a demarcation of responsibilities between the client and server-side components for performing the speech recognition task. On the client-side resides the acoustic model, which symbolically encodes the audio and encrypts the data before uploading to the server. The server-side then employs searchable encryption-based language modelling to perform the speech recognition task. The paper details the proposed client-side acoustic model components, and the proposed server-side searchable encryption which will be the basis of the language modelling. Some preliminary results are presented, and potential problems and their solutions regarding the encrypted communication between client and server are discussed. Preliminary benchmarking results with acceleration of the client and server operations with GPGPU computing are also presented.**

***Index Terms*—** Speech Recognition, Privacy, Searchable Encryption, GPGPU Computing

1. INTRODUCTION

In many activities involving big data, cloud computing offers a common distributed infrastructure for the storage of large amounts of data in a scalable, efficient, and low cost way. For sensitive data there is the possibility to use encryption for the secure storage of data in the cloud. However, whilst we have become increasingly good at encrypting data at rest, in order to process the data on the cloud we first need to decrypt it, which in turn excludes the possibility for using the cloud's resources to process sensitive data, unless it can be done in a secure way.

* Research supported by the Innovate UK as part of the Privacy Preserving Speech Recognition in the Cloud Project, Project No.: 102506

[1]C. Glackin, G. Chollet, N. Dugan, and N. Cannings are with Intelligent Voice Ltd, London, UK. (email: neil.glackin@intelligentvoice.com)

[2]S. Tahir, I. Ghosh Ray, and M. Rajarajan are with the School of Mathematics, Computer Science & Engineering, Department of Electrical and Electronic Engineering, City University London, London, UK. (email: r.muttukrishnan@city.ac.uk)

[3]Ryan Falkner3, and Atta Badii are with the Intelligent System Research Laboratory, University of Reading, Reading, UK. (email: atta.badii@reading.ac.uk)

Speech contains biometric and other information which should remain private and therefore inaccessible to the cloud provider. Cloud users want to hide sensitive data such as speech, from cloud providers; similarly, companies using cloud services want to protect their intellectual property from cloud providers and users. Hence the need for strategies for processing data securely in the cloud becomes increasingly more important.

In theory fully homomorphic encryption (FHE) offers the possibility of constructing encrypted algorithms that operate on encrypted data in such a way that neither the client's data nor the server's intellectual property is decrypted on the cloud. Since Gentry's plausible construction of an FHE scheme in 2009 (Gentry, 2009), FHE has understandably gathered increasing interest from cryptographers. However, FHE utilizes modular multiplication and addition operations on primitives on the order of millions of bits (Wang, 2012), making them currently unfeasible for big data tasks, even when configured for GPGPU acceleration. Partially homomorphic encryption (PHE) schemes such as Paillier encryption (Paillier, 1999) offer some speed up, but these schemes are still computationally demanding. Additionally, PHE schemes are limited in the range of mathematical operations they can implement, and hence they typically employ some form of secure multiparty computation by necessity to augment them. However, this then necessitates that large amounts of encrypted data are passed to the client periodically for decryption and processing outside of the cloud before re-encryption of the results and transfer back to the cloud; understandably for practical purposes the communication overhead then becomes a limiting factor.

Speech is not reproducible in the sense that no speaker is capable of making the same utterance the same way twice, there are always small acoustic differences between utterances that have the same base transcription and this is a particular challenge of performing encrypted speech recognition. Hence in order to utilize the cloud for processing large amounts of audio data in a secure way, further innovation in cryptographic schemes is required. By reformulating the typical speech recognition task in such a way as to facilitate cloud computation, for example by reducing speech recognition to a search procedure (Chollet, 1999), practical secure large scale speech processing can now be conducted in the cloud.

Section 2 presents the proposed architecture and the client/server – acoustic model/language model configuration. Section 3 outlines the privacy preserving speech encoding on the client-side. Section 4 outlines the GPGPU acceleration of AES encryption which is the basis of the proposed security. Section 5 then outlines the Ranked Searchable Encryption (RSE) scheme being developed as the basis for secure language modelling on the server/cloud side. Finally, Section 6 presents conclusions and discusses the planned future developments and potential problems with the approach.

2. PROPOSED ARCHITECTURE

Our proposed solution involves the compression of speech containing biometric identifiers to a symbolic representation that anonymizes the users' identity (Chollet, 1999), and then on the other hand to use searchable symmetric encryption (Curtmola, 2006) to enable the finding of strings of symbols (e.g. phones) in an encrypted speech transcription. Encrypted string matching will then be performed to realize the language modelling component of the speech recognition system (Pathak, 2013). Figure 1 illustrates the concept and the demarcation of responsibilities between the client and cloud server.

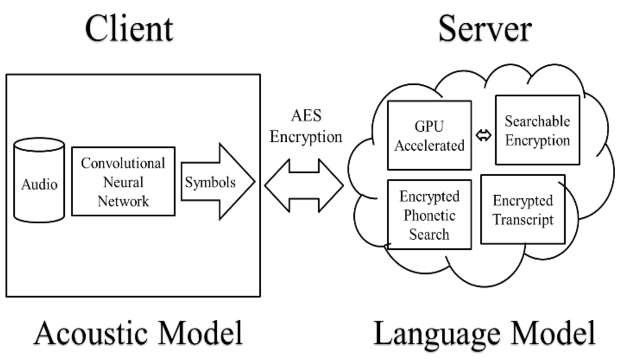


Figure 1 The client and cloud server concept and division of responsibilities for the speech processing task

Speech recognition is typically broken down into acoustic and language modelling tasks. The acoustic model converts raw speech wave forms into acoustic units such as phones. The language model incorporates natural language processing and Bayesian probability theory to infer the text transcription, given what is known of a particular language, and what words the sequences of phones likely correspond to. As can be seen from Figure 1, in the proposed system, the acoustic model resides on the client-side and the language model resides on the server/cloud-side.

From the perspective of preserving the privacy of personal data there are broadly two types of sensitive information contained in speech. These are the biometric identifiers relating to someone's identity i.e. the speech pattern itself, and then secondly there is the speech content i.e. what is being said. The acoustic model operates on the biometric data and we have taken the decision that the raw form of this data, the audio itself, is processed on the client side to convert it to phonetic symbols. The audio and its symbolic representation are then encrypted and then stored on the cloud. The symbolic representation of the audio utterances encoded on the client-side and stored on the cloud server can be searched in the encrypted domain. The setup enables secure speech processing in that the audio remains secured on the cloud until a particular user conducts searches on the database and downloads specific utterances of interest. At which point once the specific audio of interest is back on the client-side it can be decrypted with the client's private key. In this way, the cloud server is designed to be a repository for the encrypted audio, and at no point is the audio database decrypted on the cloud after it leaves the client machine.

The next section will now outline the operation of the client-side speech encoding which converts the client's audio to a symbolic representation.

3. PRIVACY PRESERVING SPEECH ENCODING

Regarding Privacy Preservation, in the cloud modality we need to make sure that personal information is not shared on the cloud. Since speech is a biometric data type, it is possible to identify someone and accurately infer a whole host of information about someone that extends beyond the obvious information such as gender, to data such as height, weight, age, health and so on. Hence we need to ensure that speech itself is not stored unencrypted on the cloud. This constraint

The client-side speech encoding may be viewed also as compressing the audio data into phonetic symbols.

3.1 Data-driven Automatic Speech Recognition

Traditionally, Automatic Speech Recognition (ASR) involves multiple successive layers of feature extraction to compress the amount of information processed from the raw audio so that the training of the acoustic model does not take an unreasonably long time. However, in recent years with increases in computational speed, adoption of parallel computation with General Purpose Graphics Processing Units (GPGPUs), and advances in neural networks (the so-called Deep Learning trend), many researchers are replacing the traditional ASR algorithms with data-driven approaches that simply take the audio data in its frequency form (e.g. Spectrogram) and process it with a Deep Neural Network (DNN), or more appropriately (since speech is temporal) with a Recurrent Neural Network (RNN) that can be trained quickly with GPGPUs. The RNN then converts the Spectrogram directly to phonetic symbols and in some cases directly to text (Hannun, 2014).

The problem with many of these approaches from the encryption point of view is that they typically combine the acoustic model and the language model with one neural network. This involves aligning the acoustic data (containing sensitive biometrics) at various stages of the network training with the text transcription with Expectation Maximization (EM), Viterbi Search or Connectionist Temporal

Classification (CTC) (Graves, 2006). In our approach, as previously stated, we think that a higher level of privacy preservation can be attained by separating the acoustic and language model training between the client and server-sides of the system. Thus we need a way to train the acoustic model in isolation to the language model. In the acoustic model developed in this project we also use spectrograms as input and phonemes as output classes to train the acoustic model and we use a Convolutional Neural Network (CNN) at the core of the acoustic model. The proceeding sub-section explains the intricacies of spectrograms and how their generation can be accelerated with parallel computation with GPGPUs. Being able to train a system to identify time-frequency intervals in acoustic data and relate it to acoustic units such as phonemes requires extremely accurate labelling of acoustic data, and this is afforded by the well-known TIMIT speech corpus.

3.2 TIMIT Speech Corpus

The TIMIT speech corpus of read speech was designed in 1993 as a speech data resource for acoustic phonetic studies and has been used extensively for the development and evaluation of automatic speech recognition studies. TIMIT contains broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The corpus includes time-aligned orthographic, phonetic and word transcriptions as well as a 16-bit, 16 KHz speech waveform file for each utterance. TIMIT was designed to further acoustic-phonetic knowledge and automatic speech recognition systems. It was commissioned by DARPA and worked on by many sites, including Texas Instruments (TI) and Massachusetts Institute of Technology (MIT), hence the corpus' name. TIMIT is the most accurately transcribed speech corpus in existence as it contains not only transcriptions of the text but also contains accurate timing of phones. This is impressive given that the average English speaker utters 14-15 phones a second. The accuracy of the corpus makes it ideal for training the acoustic model on the client-side of the system.

3.3 Deep Acoustic Modelling

Arguably it is only since 2006 (Hinton, 2006) that the training of neural networks with more than two layers has been possible. The analysis of the vanishing and exploding error phenomena, its relationship to sigmoidal activation functions (Hochrieter, 1991), and subsequent alleviation with such techniques as rectified linear units (ReLU) in feed-forward nets and 'gated recurrency' in recurrent networks (Hochreiter, 1997; Graves, 2009) has resulted in a resurgence of neural network research, namely the so called 'deep learning' trend. There are several elements that have brought this resurgence about, previously inefficient network training algorithms (when trained for small datasets) such as stochastic gradient descent (SGD) benefit markedly from economies of scale when applied to big data (Bottou, 2008). Other innovations with batch sizes (e.g. mini batches), improved learning rates (e.g. batch normalization (Loffe, 2015)), and regularization (e.g. dropout (Srivastava, 2014)) have also contributed to this resurgence. This renewed reliance on backpropagation in machine learning is also reflected in the graph-based paradigm employed by many of the deep learning frameworks e.g. Theano (Bergstra, 2011), TensorFlow (Dean, 2015), where nodes on the graph (neurons) have states and their derivatives facilitating efficient gradient calculation. The ability train deep networks would be irrelevant without the exploitation of the GPGPU. In addition to the typically graph-based paradigm, all the mainstream deep learning frameworks are now providing support for GPGPU computation. Many innovations around accelerating gradient-descent algorithms themselves (Recht, 2011), allow simultaneous adaptation of parameters during parallel training without locking the parameters in the GPGPUs shared memory.

Inspired by the work of Barlow (Barlow, 1953), the first convolutional neural network was developed by Fukushima (Fukushima, 1980). Fukushima's 'Neocognitron' performed image processing operations on images, and the design has been refined over the years, most notably with the LeNet-5 architecture (LeCun, 1990) which he used to learn the MNIST handwritten character data set. LeNet 5 was the first network to use convolutions and subsampling or pooling layers.

One of the main strengths of the CNN is that since Ciresan's seminal GPGPU implementation (Ciresan, 2011) they are now typically trained in parallel with a GPGPU, and in fact are now arguably the most common type of deep neural network currently being trained. The state of the art in CNNs is arguably the GoogLeNet (Szegedy, 2015) which was the architecture that won the ImageNet competition. ImageNet (ImageNet, 2014) is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. GoogleNet won the 2011 competition by classifying 1.2 million images into 1000 classes.

CNN's were the first deep learning network to make use of GPGPU's for their computation and we use them as the basis of the acoustic model presented in this work, primarily for the robustness and computational efficiency. They are typically used for classifying static images but we make them work for audio by using sliding windows operating over the spectrograms.

3.4 Spectrograms

A spectrogram is a time-frequency representation of a signal (in our case audio). It is obtained by applying a Fast Fourier Transform (FFT) to the audio signal. This analysis essentially separates the frequencies and amplitudes of its component sinusoidal waves. The result can then be displayed visually, with degrees of amplitude (represented by colour of greyscale), at various frequencies (usually on the vertical axis) by time (horizontal). Depending on the size of the Fourier analysis window, different levels of frequency/time resolution are achieved. A long window resolves frequency at the expense of time—the result is a narrow-band spectrogram, which reveals individual harmonics (component frequencies), but smears together adjacent 'moments'. If a short analysis window is used, adjacent harmonics are smeared together, but with better time resolution. The result is a wide-band spectrogram in which individual pitch periods appear as vertical lines (or striations), with formant structure. We typically use wide-band or Short-Term Fourier Transform (STFT) spectrograms, since we want to align acoustic data with phonetic symbols with timing that is as accurate as possible.

`SoX` is used to convert audio files in `.wav` format to a `.dat` file containing the timing of each sample frame and the integer value of the frames energy. The `.dat` file also contains the sampling frequency in the header of the file. The spectrogram code loads the `.dat` file, extracts the sampling rate automatically and uses this sampling rate as a parameter in the FFT calculation. In the CPU version of the code the well-known Cooley-Tukey algorithm is implemented. However, in our implementation of spectrograms for acoustic model input during training we can either generate them with the CPU on the client-side or with GPGPU if it is available. For the GPGPU version of the code we use NVIDIA's `CuFFT` library if the client-side has a GPGPU. Once the STFT is calculated the magnitude of the complex Fourier transform is determined and then log scaled to obtain the spectrogram. In both the CPU and GPGPU case the spectrograms are then saved to disk in Portable Network Graphics (PNG) format, which uses two libraries `libpng` (libpng) distributed as standard in Ubuntu Linux) and `pngwriter` (pngwriter) which needs to be installed and compiled from source. The spectrogram for the entire utterance is then saved to disk, the pixel size of the resulting spectrogram is 256 × number of frames/sample rate.

3.5 *Preliminary Experiments*

We use an implementation of the GoogLeNet architecture (CAFFE, DIGITS). The GoogLeNet (Szegedy, 2015) implementation was trained with SGD. Before the deep learning boom, gradient descent was usually performed by using the full set of training samples (full batch) to determine the next update of the parameters. The problem with this approach is that it is not parallelizable, and hence cannot by implemented efficiently on GPGPU. SGD does away with this approach by computing the gradient of the parameters on a single or few (mini batch) or training samples. For large sizes of datasets SGD performs qualitatively as well as batch methods but outperforms them in computational time.

Once the convolutional acoustic model is trained it is then uploaded to the client-side. It can then be used to perform inferencing, converting the audio to phonetic symbols. The audio is first converted to spectrograms, then passed to the trained CNN and the CNN classifies the sliding windows operating over the spectrogram into phonetic symbols. Figure 2 illustrates the operation of this convolutional speech encoder.

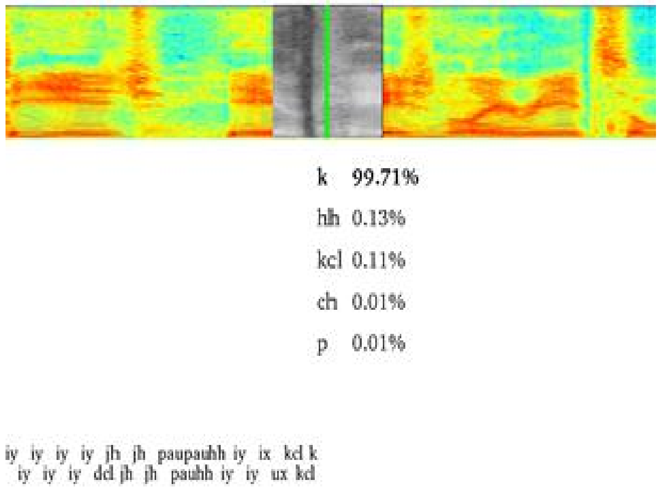


Figure 2 Client-side convolutional speech encoder

As can be seen from the figure the sliding windows operating over the spectrogram (each one is 256×256 greyscale pixels) is classified by the CNN into phoneme classes. The phoneme classes are then encrypted with AES and uploaded to the cloud. Hence the cloud only contains a symbolic representation of the encrypted speech data, the biometric identifiers associated with the raw audio remain on the client-side.

4. GPU ACCELERATION OF THE ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

GPGPU acceleration is advocated where possible throughout the system both on the client-side (if is available and on the server side (the server-side is assumed to have GPGPU functionality). In addition to accelerating the time it takes to train the convolutional acoustic model and convert the audio to spectrograms, GPGPU computation is also used for the encryption itself.

The client needs to encrypt the encoded speech and transmit it to the cloud server over the communication channel. The encoding of the audio compresses the amount of data that needs to be encrypted, but further efficiencies can be gained by accelerating the encryption itself. We use the Advanced Encryption Standard (AES) algorithm as the basis of the security in the proposed system, and we use GPGPU-based AES encryption if it available on the client-side for its speed.

Two versions of the Advanced Encryption Standard (AES) algorithm have been implemented on both the CPU in plain C/C++ and on the GPU using CUDA C. The particular version of AES that has been implemented is the Electronic Cook Book (ECB) mode which splits plain text into a number of blocks of 128 bits (16 bytes), each of which is independently encoded by the algorithm using a series of keys derived from a provided encryption key. The encoding takes place over 10 rounds (using a 128-bit key) which involves various substitutions and permutations using a different value derived from the encryption key in each round.

v1 of the algorithms uses inline C functions for computations in the `MixColumns()` step of the algorithm whereas v2 uses a pre-calculated lookup table which is copied over and stored on the device. For brevity the following preliminary results only show the v2 of the algorithm.

The CUDA implementations currently only exploit coarse grain parallelism optimizations using the GPUs vast number of multiprocessors to process cipher blocks in parallel. This is achieved by assigning individual cipher blocks (128 bits) to different blocks which are processed by one of many of the devices' multiprocessors. However please note that the total number of threads assigned needs to be equal to or greater than the number of cipher blocks to be processed otherwise all blocks will not be processed.

The AES implementation were setup and run with varying input size to generate the comparison performance graph between the CPU Host and the GPU device. Figure 3 shows the differences in execution time between host and device as the number of cipher blocks increases.

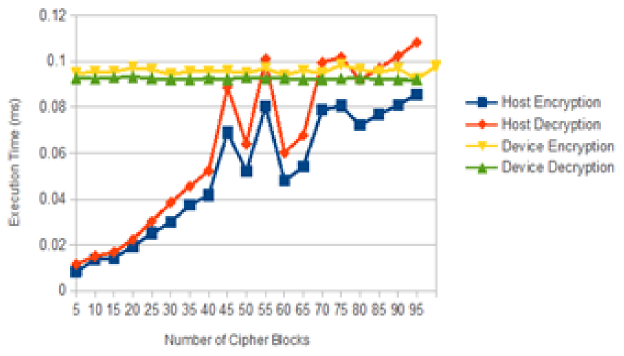


Figure 3 Comparison between CPU Host and GPU Device illustrating differences in execution time of AES encryption for less than 100 cypher blocks

These show clear linear relationships displayed by the two approaches with an overlap occurring between roughly 80-100 blocks of 128 bits where the performance of the GPU implementation becomes greater than that of the CPU implementation. The initial limitation of the GPU implementation is the overhead of copying the input over to the device from host memory however the benefit of parallel processing makes this obsolete beyond this point. This relationship has been observed to continue as such with an increase in the number of input blocks up to 3000, as is illustrated by Figure 4.

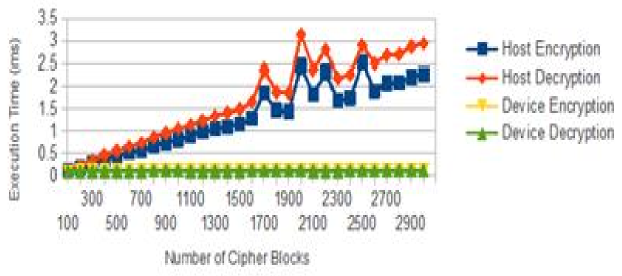


Figure 4 Comparison between CPU Host and GPU Device illustrating differences in execution time of AES encryption for up to 3000 cypher blocks

The next Section outlines the proposed encryption scheme which will be the basis for the server-side language modelling.

5. SEARCHABLE ENCRYPTION FOR LANGUAGE MODELLING

Figure 5 illustrates the potential places of attacks that we need to address for the development of the security of the secure speech processing system.

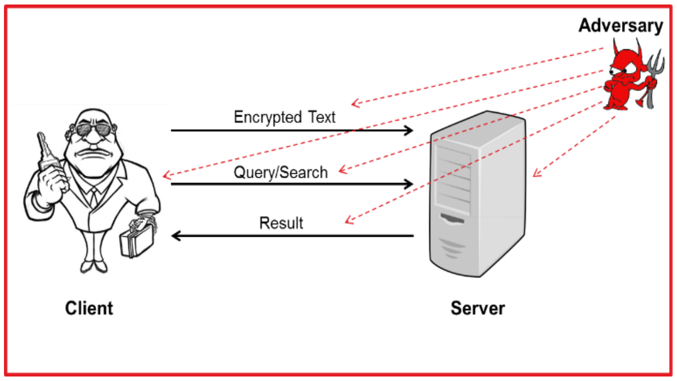


Figure 5 Adversarial Assumptions for the client-cloud server model

The server is assumed to be 'friendly but curious', meaning that it will not actively try and break the encryption but that it will monitor the traffic and infer the content of the audio if it can, which means that the data should be encrypted. We also assume that the client is taking care of their own network and its security. The communication channels will be assumed to be under threat from an adversary as is illustrated from the figure. The communication channels will be used to transmit AES encrypted symbolic speech data, and in return AES encrypted search results and transcriptions in the final system functionality. This is the basis by which we will design suitable encryption.

When designing suitable encryption algorithms, we need to be aware that there are trade-offs between security, efficiency of the algorithm and query effectiveness which is closely related to the usability of the system.

In this paper, we present one of the searchable encryption schemes we have been developing with the speech recognition task in mind, we have termed it Ranked Searchable Encryption, but first we present some background on deterministic searchable encryption.

5.1 Deterministic Encryption

Deterministic encryption always encrypts the same message to the same ciphertext. The property preserved by deterministic encryption is equality i.e., for any given two encryptions one can test if the underlying messages are equal by just checking the given ciphertexts. Due to this equality property, the encrypted database leaks a large amount of data even before the client searches. This means that if the server sees two or more equal ciphertexts in the encrypted database, it knows that the corresponding encrypted documents contain a keyword in common. In addition, the server learns the frequency with which keywords appear which makes the encrypted database vulnerable to frequency analysis. Another issue is that since tokens are deterministic encryptions of the search terms, the server will always know whether the client is repeating a search a not.

A third issue occurs when the deterministic encryption scheme is based on a public-key scheme. In this case, all the deterministic encryptions (both in encrypted database and in the tokens) are encrypted using the client's public key which is, obviously, public and available to the server. The server can then mount a dictionary attack on the encrypted database by encrypting a list of possible keywords and comparing them to the ones found in the encrypted database and in the tokens. If it finds a match, then it knows the keyword. Hence the solution based on deterministic encryption supports fast search on encrypted data. However, the public-key based scheme unfortunately leaks a considerable amount of information to the server.

Searchable encryption built using identity based encryption (IBC) mitigates most of the drawbacks that exist in the above deterministic encryption based scheme. In general, the IBC scheme consists of four algorithms:

Setup (λ): On input the security parameter λ, this algorithm returns a master secret key msk and public parameters pp.
Extract (pp, msk, id): On input public parameters pp, a master secret key msk and an identity id ∈ I, this algorithm outputs a secret key sk.
Enc (pp, m, id): On input public parameters pp, a message m ∈ M and an identity id ∈ I, this algorithm outputs a ciphertext c.
Dec (pp, c,sk): On input public parameters pp, a ciphertext c and a secret key sk, this algorithm outputs either a message m or a failure symbol ⊥.

The above identity based encryption can be reformulated to support searchable encryption as follows:

KeyGen (λ): On input the security parameter λ, this algorithm returns a private/public key pair (sk, pk).
PEKS (pk, w): On input a public key pk and a keyword w ∈ W, this algorithm produces a searchable ciphertext c.
Trapdoor (pk, sk, w): On input a public key pk, a secret key sk and a keyword w ∈ W, this algorithm outputs a trapdoor tp for w.
Test (pk, c, tp): On input a public key pk, a searchable ciphertext c and a trapdoor tp, this algorithm outputs either True or False.

The above transformation is secure in the traditional ciphertext indistinguishability sense. Identities and their secret keys in the original scheme map to keywords and trapdoors in the transformed scheme, respectively. The anonymity requirement informally states that ciphertexts leak no information regarding the identity of the recipient, leading to the commonly desired keyword-privacy guarantees over ciphertexts in searchable encryption (the transformed one). The standard notion of ciphertext indistinguishability in identity based encryption leads to Computational Consistency in the transformed scheme, which informally means that it is hard for computationally bounded adversaries to find two distinct keywords such that the trapdoors for the first keyword positively match the ciphertexts of the second keyword.

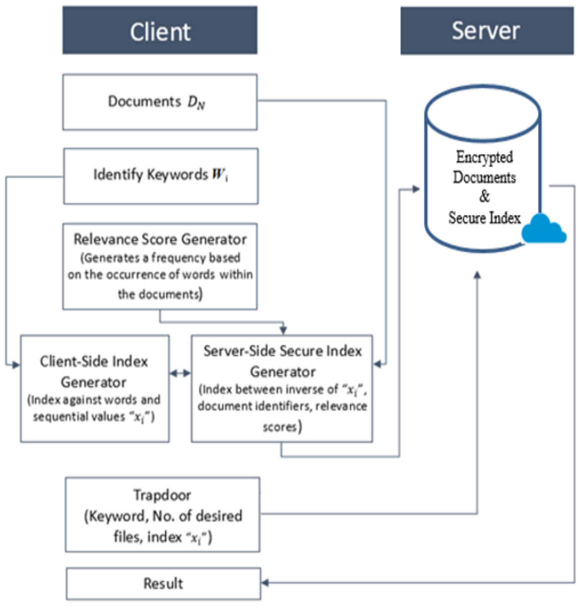
5.2 Ranked Searchable Encryption

We will consider the client-server infrastructure by visualizing a scenario in which there are two parties, Alice (Client) and a Cloud Server (CS). Alice intends to upload all her documents (encrypted speech files) $D = \{D_1, D_2, \dots, D_i\}$ to the CS to enable remote access. The CS performs the searching of the relevant documents on the behalf of Alice. In the scheme it is assumed that the CS is a trusted but curious server. Being trusted means that the CS acts in a known and designated manner but the CS is equally also willing and curious to get hold of any information about the documents held with it. To prevent theft of any of the information Alice decides to encrypt all the documents. Once the documents are encrypted and outsourced she is challenged with the problem of searching on the encrypted documents. Whenever Alice decides to view a particular file she has to download all the documents from the CS and after decrypting all of them she can get hold of her required set of files. This creates unnecessary network traffic and post processing overhead. Alice decides to outsource the documents in such a way that she would only have to download the relevant and desired documents while keeping the security and privacy of the outsourced files intact. This requires a scheme to be developed that would facilitate performing textual searches over encrypted data.

Searching over encrypted documents is performed in three phases (Setup, Searching and Outcome). These phases are further divided into sub-steps/sub-phases. The first phase i.e. the Setup Phase comprises of three steps i.e. Keyword Identification, Client Index Generation and Server Index Generation. In the first step Alice generates an exhaustive set of unique Keywords $W = \{W_1, W_2, \dots, W_N\}$ from the set of documents D to be outsourced. In the second step (Client Index Generation), Alice builds a client-side index table Ic. The Ic is stored with Alice and is never revealed to the CS. In step 3 (Server Index Generation), Alice generates a secure ranked server-side index Is and outsources it to the CS along with the encrypted set of documents D. Step 3 involves relevance frequencies of the keywords to be calculated and inserted in the index table.

In the Searching Phase, Alice generates a Trapdoor *Ti* for the particular keyword W_i she is willing to search. The generated Trapdoor is then transmitted to the CS to facilitate the search. In the Outcome Phase the CS returns the encrypted set of desired files to Alice in the ranked order.

Figure 6 shows the flow of events of the proposed RSE scheme where a client is interacting with a cloud server (CS). It can be seen that all the tasks are performed by the client, whereas, the searching is done at the CS side.



Client
Server
Documents D_N
Identify Keywords W_i
Relevance Score Generator (Generates a frequency based on the occurrence of words within the documents)
Client-Side Index Generator (Index against words and sequential values "x_i")
Server-Side Secure Index Generator (Index between inverse of "x_i", document identifiers, relevance scores)
Encrypted Documents & Secure Index
Trapdoor (Keyword, No. of desired files, index "x_i")
Result

Figure 6 RSE Flow of Events

The RSE scheme comprises of four polynomial time algorithms Π = (KeyGen, BuildIndex, BuildTrap, SearchOutcome) such that:

***K*←KeyGen (1*λ*):** is a probabilistic key generation algorithm run by the client. The algorithm takes a security parameter λ as the input and returns a master key K and a symmetric key k shared with the CS.

(Ic, Is)←BuildIndex (K,D): is run by the client to generate indices. The algorithm takes a master key K and a collection of documents D to be outsourced to the CS as input. The algorithm returns a client side index Ic and a server side index Is.

***Tw*← BuildTrap (k, K, w, Ic, num):** is run by the client. The algorithm takes the symmetric key k, master key K, keyword w, client side index Ic and the number (num) of documents D required as the input. The algorithm returns a trapdoor Tw.

***id*(*Di*)← SearchOutcome (k, Is, Tw):** is run by the CS. The algorithm takes the symmetric key k, server side index Is and the trapdoor (Tw) as the input and returns the desired document identifiers $id(Di)$ containing the keyword w in ranked order.

Pre-processing is done on the client side in three major steps, namely

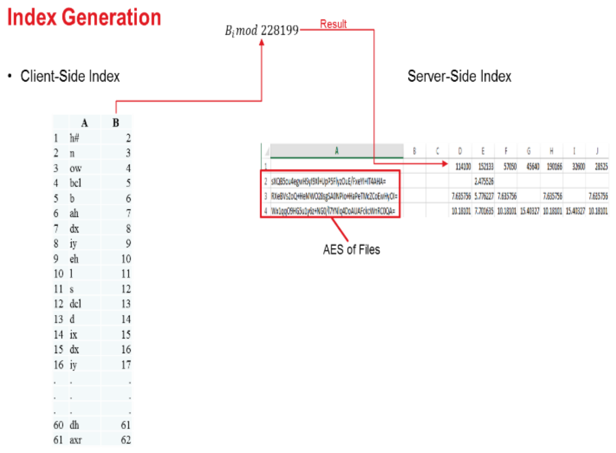
1. Frequency computation
2. Client-side index generation
3. Server-side index generation

5.2.1 *Frequency computation*

First the scheme computes the frequency of the words appearing in each of the selected files as shown in the above figure. For example, the column corresponding to the word 'this' tells about its frequency of occurrence in the files word.doc, Latest.doc, new.docx and F_t which are 0, 1, 1 and 2 respectively. This data is essential for the server side index generation.

5.2.2 *Client-side index generation*

The next task is to generate the client-side index. The client side index table is a collection of all key words each assigned with a unique integer other than 0 and 1. If the total number of such set of keywords is, say, '*n*' in number, then a prime number '*p*' is chosen such that *p* > *N*. All integers that are assigned for the keywords are from the set {2, 3, …, *p*-1}. In the implementation *p* is 228199.



	A	B
1	h#	2
2	n	3
3	ow	4
4	bcl	5
5	b	6
6	ah	7
7	dx	8
8	iy	9
9	eh	10
10	l	11
11	s	12
12	dcl	13
13	d	14
14	ix	15
15	dx	16
16	iy	17
.	.	.
.	.	.
.	.	.
60	dh	61
61	axr	62

	A	B	C	D	E	F	G	H	I	J
1				114100	152133	57050	45640	190166	32600	28525
2	sXQB5cu4egwHSyI9XI+Up?5FIyzOuE/FxeYI+IT4AHA=				2.475526					
3	RXeBVsZoQ+HeMWO28sgSA0NPio+HaPeTMc2CoEwHyOI=			7.635756	5.776227	7.635756		7.635756		7.635756
4	Wx1qqO9HG5u1y6z+NG0/I7YNIq4DoAUAFckWnRC0QA=			10.18101	7.701635	10.18101	15.40327	10.18101	15.40327	10.18101

Figure 7 Modulus Prime Encryption Index Table Generation

5.2.3 *Server-side index generation*

Server-side index table is quite similar to the frequency table with three modifications:

Firstly, the words are replaced by the integers which are the multiplicative inverse of their corresponding client-side index computed modulo prime p. For example, the client-side index of this is 2. The multiplicative inverse of 2 in modulo

228199 is 114100. So in the Server-side index table, the word ‘this’ is replaced by 114100.

Secondly, the file names in the frequency table are replaced by the encrypted file names. For example, `word.doc` is replaced by AES(`word.doc`) as shown in the figure.

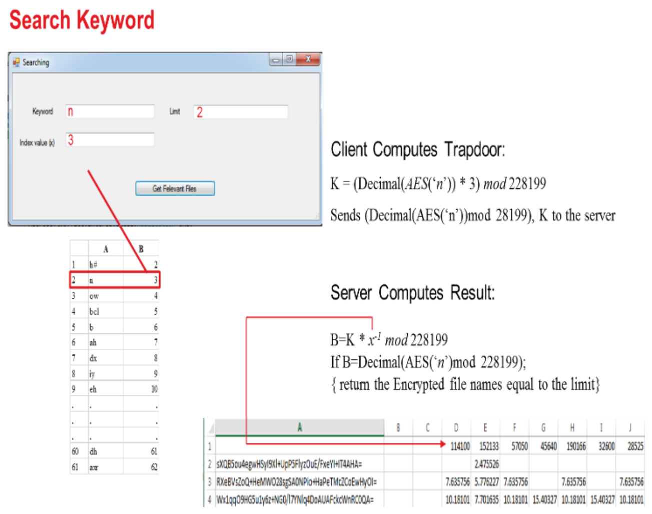
Lastly, the frequencies are replaced by relevant scores which are computed using the following formula:

$$Score(Q, F_d) = \sum_{t \in Q} \frac{1}{|F_d|} \left(1 + \ln f_{d,t}\right) \ln\left(1 + \frac{N}{f_t}\right)$$

5.2.4 *Searching for a keyword*

To search for a keyword, say the phoneme ‘*n*’, the client will compute E = (Decimal (AES (‘*n*’))) mod 228199 and the trapdoor K = (Decimal (AES (‘*n*’)) * 3) mod 228199.

The client’s query being sent to the server is of the form (K, E). Note that 3 is the client-side index of the word ‘is’. In the server side, after receiving (*K*, *E*), *K* will be multiplied modulo 228199 with the integers occurring in the first row of server-side index table one by one unless product matches *K*. For example since the multiplicative inverse of 3 in modulo 228199 is 152133, ($K \times 152133$) *mod* 228199 = Decimal (AES (‘*n*’)) *mod* 228199 = *E*. Now the entries of the column corresponding to the integer 152133 are to be checked. The higher the score, the more relevant the corresponding file is with respect to the search. If the number of files in which the search is be performed is, say, 2, then the top two files according to the top two relevant scores for the keyword ‘*n*’ are 7.7 and 5.7 and the corresponding encrypted files are `new.docx` and `Latest.doc`. So the server will return the encrypted new.docx first and then `Latest.doc`. In this way, a phonetic search of the encrypted database can be carried out.



Keyword: n Limit: 2

Index value (x): 3

K = (Decimal(*AES*(‘*n*’)) * 3) *mod* 228199

Sends (Decimal(AES(‘n’))mod 28199), K to the server

	A	B
1	h#	2
2	n	3
3	ow	4
4	bcl	5
5	b	6
6	ah	7
7	dx	8
8	iy	9
9	eh	10
.	.	.
.	.	.
.	.	.
60	dh	61
61	axr	62

B=K * x^{-1} *mod* 228199

If B=Decimal(AES(‘*n*’)mod 228199);

{ return the Encrypted file names equal to the limit}

	A	B	C	D	E	F	G	H	I	J
1				114100	152133	57050	45640	190166	32600	28525
2	sXQB5ou4egwHSyI9Xl+UpP5FlyzOuE/FxeYI+IT4AHA=				2.475526					
3	RXeBVsZoQ+HeMWO28sgSA0NPio+HaPeTMcZCoEwHyOI=			7.635756	5.776227	7.635756		7.635756		7.635756
4	Wx1qqO9HG5u1y6z+NG0/I7YNlq4DoAUAFckcWnRC0QA=			10.18101	7.701635	10.18101	15.40327	10.18101	15.40327	10.18101

Figure 8 Searching for a keyword

6. CONCLUSIONS

We have presented an outline of an ASR system that preserves privacy within its operation. Whilst the complete system is still in development the rationale for how the system works has been presented along with specification and preliminary operation of many of the core modules. The rationale advocates that audio data is uploaded to the cloud and remains encrypted once it leaves the client-side. On the client-side the audio data is additionally encoded into symbols by a novel convolutional neural network based acoustic model. The encoded speech (phonetic symbolic strings) are then encrypted with AES and uploaded to the cloud. RSE then provides the capability to perform phonetic searching of the encoded audio securely in the cloud realizing the speech recognition task. The inherent trapdoor security that RSE employs preserves the privacy of searches.

Future work is aimed at a number of developments specifically aimed at satisfying the requirements of a robust commercial encrypted speech engine. One of the understandable criticisms of the scheme is that if new data is added to the encrypted repository that server-side index table needs to be recalculated. This is an obvious necessity of the reliance on relevance ranking that the scheme employs. A future iteration of encrypted speech will remove the ranking aspect which will enable the adding of rows to the server-side index without a global recalculation operation over the entire database.

We have demonstrated single keyword (phone) searches in the encrypted domain. Future work will also describe how sequences of phonetic symbols can be searched for without ‘leaking’ information about the search history. This is fundamental to the usability of the system since it will then be possible for the user to search for keywords. The keyword will be converted into phonetic strings on the client-side with a simple lexicon, and then the phonetic string representation of the keyword can be used to search the encrypted database on the cloud. It will also be possible to use n-best representations of the phonetic search strings. This will involve parallel searches with the n-best phonetic strings, but should promote robustness by accounting for variability in pronunciation due to different accents and dialects for example.

We have used a CNN-based encoder on the client-side encoder as it is lightweight, it is a compressed python object which can be used to encode the symbolic representation of each audio file quickly whether the client has GPU capability or not.

ACKNOWLEDGEMENTS

This research was supported by Innovate UK as part of the ‘Privacy Preserving Speech Processing in the Cloud’ project (Project No.: 102506).

REFERENCES

G. Chollet, J. Cernocky, A. Constantinescu, S. Deligne and F. Bimbot "Toward ALISP: A proposal for Automatic Language Independent Speech Processing" in Computational Models of Speech Pattern Processing, K. Ponting (ed.), NATO ASI Series Vol 169, pp 375-388, 1999.

R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions, Proc. 13th ACM Conf. Comput. Netw. Secur. Pp. 79, 2006.

SoX, the Swiss Army knife of sound processing programs, http://sox.sourceforge.net

The NVIDIA CUDA Fast Fourier Transform library (cuFFT): https://developer.nvidia.com/cufft

The official PNG reference library: http://www.libpng.org/pub/png/libpng.html

PNGwriter is a C++ library for creating PNG images: http://pngwriter.sourceforge.net/

DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM (1993) by J. S. Garofolo, L. F. Lamel, W. M. Fisher, et al. https://catalog.ldc.upenn.edu/LDC93S1

G. E. Hinton, S. Osindero, and Y. The, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527-1554, 2006.

S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991.

S. Hochreiter, J. Schmidhuber, "Long Short-term Memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

A. Graves, J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," Advances in Neural Information Processing Systems, vol. 22, pp. 545-552, 2009.

L. Bottou, O. Bousquet, "The tradeoffs of large scale learning," Advances in Neural Information Processing Systems, vol. 20, pp. 161-168, 2008.

S. Loffe, C. Szegedy, "Batch normalisation: accelerating deep network training by reducing internal covariate shift," arXiv preprint, arXiv: 1502.03167, 2015.

J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, "Theano: Deep learning on GPGPUs with python," Neural Information Processing Systems (NIPS), 2011.

J. Dean, et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems, White Paper, Google Research, 2015.

B. Recht, C. Re, S. Wright, F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," Advances in Neural Information Processing Systems, pp. 693-701, 2011.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).

Graves, A., Fernández, S., Gomez, F. and Schmidhuber, J., 2006, June. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23rd international conference on Machine learning (pp. 369-376). ACM.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A. and Ng, A.Y., 2014. Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.

C. Gentry, "Fully homomorphic encryption using ideal lattices," Proc. 41st ACM Symposium on Theory of Computing, 2009.

W. Wang, Y. Hu, L. Chen, X., Huang, B. Sunar, "Accelerating fully homomorphic encryption using GPGPUs," IEEE Conference on High Performance Extreme Computing, pp. 1-5, 2012.

P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," Advances in Cryptology, EUROCRYPT'99, pp. 223-238, 1999.

M. A. Pathak, B. Raj, S. Rane, and P. Smaragdis, "Privacy-preserving speech processing: cryptographic and string-match frameworks show promise," IEEE Signal Processing Magazine, vol. 30, no. 2, pp. 62-74, 2013.

Barlow, H.B., 1953. Summation and inhibition in the frog's retina. The Journal of physiology, 119(1), p.69.

CAFFE Deep Learning Framework: http://caffe.berkeleyvision.org/

NVIDIA DIGITS Interactive Deep Learning GPGPU Training System: https://developer.nvidia.com/digits

D. C. Ciresan, U. Meier, J. Masci, L. Gambardella, J. and Schmidhuber, 'Flexible, high performance convolutional neural networks for image classification, IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, no. 1, pp. 1237, 2011.

ImageNet: http://www.image-net.org/

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, Advances in Neural Information Processing Systems 2 (NIPS*89), Denver, CO, 1990.

Fukushima, Kunihiko, 'Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,' Biological Cybernetics 36 (4): 193-202, 1980