Privacy-preserving Outsourced Speech Recognition for Smart IoT Devices

Zhuo Ma ‡ , Yang Liu *‡, Ximeng Liu ‡ , *Member, IEEE*, Jianfeng Ma, *Member, IEEE*, Feifei Li, *Member, IEEE*

Abstract—Most of the current intelligent Internet of Things (IoT) products take neural network based speech recognition as the standard human-machine interaction interface. However, the traditional speech recognition frameworks for smart IoT devices always collect and transmit voice information in the form of plaintext, which may cause the disclosure of user privacy. Due to the wide utilization of speech features as biometric authentication, the privacy leakage can cause immeasurable losses to personal property and privacy. Therefore, in this paper, we propose an outsourced privacy-preserving speech recognition framework (OPSR) for smart IoT devices in the long short term memory (LSTM) neural network and edge computing. In the framework, a series of additive secret sharing based interactive protocols between two edge servers are designed to achieve lightweight outsourced computation. And based on the protocols, we implement the neural network training process of LSTM for intelligent IoT device voice control. Finally, combined with the universal composability theory and experiment results, we theoretically prove the correctness and security of our framework.

Index Terms—Privacy-preserving, Smart IoT Devices, LSTM, Speech Recognition, Forward Propagation, BPTT

1 Introduction

RECENTLY, the development of SmartHome [1] and E-health [2] proceeds the varieties of applications of intelligent products of Internet of Things (IoT) into people's horizon. As shown in Fig.1, to facilitate the control of the smart devices, intelligent voice control based on speech recognition is deployed as the interface to receive human commands, like Amazon Alexa and Apple Siri [3]. However, speech as the most commonly used communication way for human can also contain plenty of valuable and sensitive information and may cause severe damage to personal property or security. And as one of the unique biometric features toward each person, speech features are often used to implement speaker identification and speaker verification, such as authenticating bank accounts [4]. The popularity of the speech recognition service in smart IoT industry leads to a mass of speech data faced with a risk of leakage to the service provider in the form of plaintext. For a corrupted service provider or a malicious adversary, the speech data can be used as the materials to imitate the user's voice and obtain the access permission to bank accounts or achieving other illegitimate purposes. What's more, besides the speech features, the parameters of speech recognition neural networks are also the adversary's attack target. Under the common situation, the speech recognition systems of most application are run on cloud servers. If the neural network parameters are given in plaintext format, it is completely possible that the administrator of the cloud server may steal the parameters and sell them out of commercial purpose. Because, nowadays, it is still a costly work to train a mature neural speech recognition network which can be used in commercial activities.

- *Z. Ma, Y. Liu and J. Ma are with School of Cyber Engineering, Xidian University, Xi'an 710071, China.*
  E-mail: mazhuo@mail.xidian.edu.cn,bcds2018@foxmail.com, jfma@mail.xidian.edu.cn
- *X. Liu is with College of Mathematics and Computer Science, Fuzhou University, China and Fujian Provincial Key Laboratory of Information Security of Network Systems, Fuzhou, China.*
  E-mail: snbnix@gmail.com
- *F. Li is with with School of Computing, University of Utah, Salt Lake City, UT, 84112.*
  E-mail: lifeifei@cs.utah.edu
- *∗ Corresponding author*
  ‡ These authors contributed equally to this work.

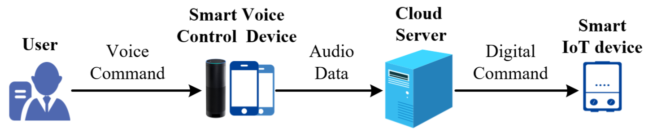

Fig. 1. Speech Recognition based Voice Control for Smart Devices

Also, speech recognition has always been a complicated and challenging work for the smart IoT devices providers. Along with the advances of deep learning neural network, the long short term memory (LSTM) network-based speech recognition methods have exceeded and substituted the traditional ones. Nevertheless, although the accuracy is improved, the demand for superior computing power and huge amount of memory space is still high. For most of the deep learning based applications, even at experimental stage, a computational budget of 1.5 billion multiply-adds at inference time have to be reserved to guarantee their practicality in real-world use [5]. And the limitation of IoT processing capacity also increases the difficulty of real-time response for speech recognition applications. Simply for reduction of noise, the local IoT voice control device needs tens of milliseconds to compute [6]. As a consequence, more and more currently applied frameworks of speech recognition in IoT choose to outsource the user data and trained neural network to the cloud data centers [7]. However, all the audio feature data about our speech is exposed to not only the intelligent IoT voice control service provider but also the cloud server operators. And due to the long communication distance, the demands for robust bandwidth and the short-latency are even higher.

In order to optimize the performance of cloud computing and IoT devices, a new computing paradigm, edge computing, is proposed. In the last year, Nguyen *et al.* [8] specially design a decentralized and revised content-centric networking based mobile edge computing platform for IoT devices. By processing data at the edge of the network which is close to the data source and balancing load, edge computing can dramatically reduce response time and improve the battery life of IoT device [9]. However, the privacy leakage problem has not been resolved in the paradigm, since the audio features are still exposed to the cloud servers and the service provider. Generally speaking, homomorphic encryption and additive secret sharing that allow computation on encrypted data are ideal ways to solve this problem. Accordingly, as early as 2007, Smaragdis *et al.* [10] have proposed a two-party secret sharing based privacy-preserving framework for Hidden Markov Models (HMM). And years later, Manas *et al.* [11] utilize the homomorphic encryption to protect the data privacy of Gaussian Mixture Models (GMM). However, current homomorphic encryption based frameworks are all time-consuming and memory-intensive, which lead to its impracticability in real application. And up to now, most of the similar frameworks are towards the traditional speech recognition techniques which has been completely overperformed by the LSTM based techniques in 2007 [12]. Consequently, what is required now is to address the issue of privacy leakage in LSTM based IoT speech recognition framework while maintaining high efficiency.

To this end, in this paper, we propose OPSR for wider applications of smart IoT devices. In the framework, we use the secret sharing technique to ensure the security and privacy for both the audio data of users and the neural network parameters of intelligent IoT voice control service providers. To achieve this, we specially design several interaction protocols for some mathematical operations and the different gates of LSTM. And all the audio features and the weight matrixes of LSTM are computed in the form of ciphertext. Furthermore, to improve efficiency and handling capacity, edge computing is introduced into OPSR. We arrange two independent edge servers dedicated to computing. Note that we assume that there is a trusted third party to generate uniformly random values for the interaction protocols. And the audio features are extracted on the local IoT device and then transported to the edge servers. The contributions of this paper are summarized as follows.

- A privacy-preserving LSTM based speech recognition framework for smart IoT devices is designed. By operating the secret sharing protocols on two edge servers, the framework can protect the privacy of user in speech recognition at the expense of a little efficiency. Meanwhile, it can also protect the privacy of neural network parameters for the intelligent IoT voice control provider.
- Responding to the different gates of LSTM, we propose several additive secure interactive protocols for secure computation of the intelligent IoT voice control, such as the secure sigmoid function and secure tanh function. Due to the avoidance of computation-intensive cryptographic operations like homomorphic encryption, the protocols can efficiently get the computation result with negligible error and minimum communication overhead.
- We use experiments to confirm that our framework is efficient enough for the application of smart IoT devices voice control under the privacy-preserving condition. Besides, the accuracy of the neural network is almost not influenced compared with the normal situation.

The rest of this paper is organized as follows. Related work is discussed in Section 2. In Section 3, the primitives about LSTM and secure multiparty computation are briefly introduced. Section 4 gives the system model of OPSR. Section 5 lists the additive secret sharing based protocols for basic mathematical operations. Section 6 illustrates the details of forward propagation and back propagation of LSTM while the data privacy is preserved. The security and the correctness is given and proved in Section 7 and Section 8. The last Section is the conclusion of this paper.

2 Related Work

Speech recognitions are more and more widely deployed to serve the smart Internet of Things (IoT) products and bring us with great convenience. This owes not only to the advances of hardware equipment, but also to the development of artificial intelligence technology, especially recurrent neural network (RNN). Different from the sequential structure of CNN [13], RNN forms a complex recurrent chain structure through three basic layers, input layer, hidden layer and output layer. However, Bengio, et al [14] prove that, because of the long-term dependencies problem, RNN becomes unreliable under complex application environments. Thus, an improved RNN algorithm, LSTM, is brought into our horizon. The difference between RNN and LSTM is that the latter uses three gates to substitute the three layers. In [15], LSTM is also used to analyse the security of secure protocols. The development of deep learning network provide dependable and convenient control interface to support the extensive set of applications for IoT, such as SmartHome [1] and E-health [2]. The most popular speech recognition algorithm has always been the Hidden Markov Model (HMM) [16] until 2007. In that year, a deep learning network, the Long Short Term Memory network (LSTM), is successfully trained to exceed the traditional method [12]. In 2017, the CNN-LSTM based conversational speech recognition system proposed by Microsoft reaches no more than 10% error rate which is a much better performance than the traditional methods in practical application scenario [17].

Intuitively, to protect the privacy of the audio features, we can use secure and computable encryption algorithm to encrypt them before outsourcing to the cloud server. The commonly used encryption methods are homomorphic encryption (HE) and Yao's Garbled Circuit (GC) protocol. After CryptoNets [18] is proposed by Microsoft, many researchers make use of additive HE to protect decentralized data privacy of deep learning network, such as the CryptoDL [19], MiniONN [20] and RNNQ [21]. By virtue of the homomorphism, this encryption method makes it possible to do the linear operations involved in the neural network in ciphertext form. And the nonlinear operations can also be skillfully calculated by utilizing the iterative formulas to approximate the real results [22], [23]. However, it is a consensus that current homomorphic encryption algorithms always introduce expensive computational costs into the system. Recently, Deepsecure [24] based on GC is proposed for capacitating the deep learning network to avoid the exposure of data provided by distributed clients. Although Bita *et al.* declares that Deepsecure decreases at least two orders-of-magnitude runtime, Ahsan *et al.* [25] point out that GC was still not practical due to severe implementation issues in its efficiency and reusability. Another approach for privacy preservation of such deep learning based application is differential privacy technology [26] [27]. By adding perturbations into the user data, differential privacy can hide the features of single sample, but preserve the statistic features for the usage of deep learning. However, the most serious problem of differential privacy is that it cause obvious reduction of data availability, which can lead the extreme decrease of accuracy for most neural network [28].

3 Preliminaries

In this section, we briefly introduce existing primitives, including Recurrent Neural Networks (RNN) based Long Short Term Memory network (LSTM) and additive secret sharing based protocols.

3.1 Features of LSTM

LSTM is one of the most popular recurrent neural network [29]. Besides the linear feature transformation function, two activation functions are used in the neurons of LSTM to make it capable of long-term memory, which are sigmoid function $\sigma(x)$ and tanh function $tanh(x)$. Let (b_f, b_i, b_C, b_o) and (W_f, W_i, W_C, W_o) are responding to the bias and weight matrix in the different gates, respectively. And C_t, x_t, O_t and h_t denote the neural cell state, the input, the output of output gate and the output of LSTM at time t, respectively. f_t, i_t and O_t are the outputs of forget gate, input gate and output gate. Then, the three gates can be expressed as following.

1) **Forget Gate**. The 1st step in LSTM is to determine what information received from the previous cell we would like to discard. And this is realized by the forget gate.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{1}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

2) **Input Gate**. The input gate utilizes a sigmoid function and a tanh function to determine what fresh information should be added into the current cell state, and then updates the state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{3}$$

$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{4}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{5}$$

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C_t}. \tag{6}$$

3) **Output Gate**. In this gate, LSTM outputs a filtered value as output based on the current neural cell state. And the filter is made up of a tanh function.

$$O_t = \sigma(W_O \cdot [h_{t-1}, x_t] + b_O), \tag{7}$$

$$h_t = O_t \times \tanh(C_t). \tag{8}$$

3.2 Additive Secret Sharing Protocols

Additive secret sharing protocols mainly works for secure multiparty computation (SMC) and privacy preservation. Due to its excellent performance, additive secret sharing based encryption protocols can be deployed to design effective privacy-preserving computation models of IoT [30]. According to the universal composability framework [31], secret sharing protocols can be regarded as plenty of "component", with which we can build a bigger but equally secure system.The formal illustration is as follows.

Lemma 1. A protocol is perfectly simulatable if all its sub-protocols are perfectly simulatable. [32]

Here, we just briefly introduce several existing "components" which are indispensable for later description. And in the next sections, we propose more protocols to expand the "components" set. What should be noted is that all the protocols in this paper, including the newly proposed ones, are two party setting (2PC) [33] and have a trusted third party to generate uniformly random values.

1) **Secure Addition Protocol**. Given an input two-tuple (μ, ν), the $SecAdd(\cdot)$ protocol outputs (ζ_1, ζ_2), where $\zeta_1 + \zeta_2 = \mu + \nu$. In the process, there is no need for the two participants and the third party to interact with each other.
2) **Secure Multiplication Protocol**. The $SecMul(\cdot)$ protocol is based on the *Beaver's triplet* [34]. Given a two-tuple (μ, ν) as input, the protocol outputs another two-tuple (ζ_1, ζ_2) for the two participants, satisfying $\zeta = \zeta_1 + \zeta_2 = \mu \cdot \nu$ During the protocol, the third party has to generate a random triplet (a, b, c) and $c = a \cdot b$ to guarantee that the two inputs cannot be identified by the adversaries.
3) **Secure Comparison Protocol**. The protocol, denoted by $SecCmp(\cdot)$, uses the most significant bit to compare which one of the two inputs μ and ν is more bigger. During comparison, the two inputs are respectively occupied by two participants and not revealed to each other. If $\mu < \nu$, $SecCmp(\mu, \nu)$ outputs 1, otherwise outputs 0 [35].

4 System Architecture

4.1 System Model

As shown in Fig.2, OPSR is composed of seven participants, a user U with his smart audio IoT device AD, two edge servers S_1 and S_2, the trusted third party T, the smart IoT device I and the intelligent IoT voice control service provider SP. Among them, only S_1 and S_2 are responsible for computation. While being used, AD collects the speech

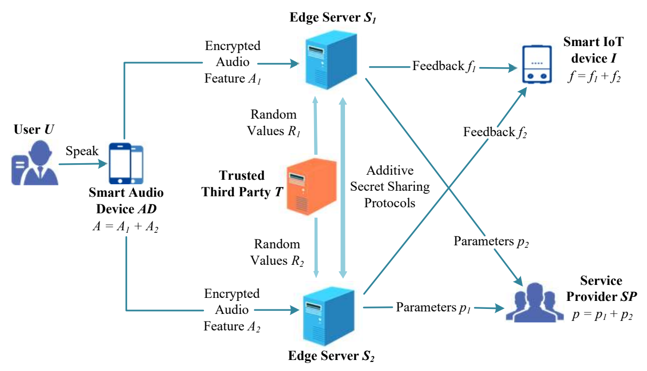

Fig. 2. System Model

features of U, and then sends them to two edge servers. Seconds later, I can receive the feedback of the edge servers and recover encrypted the command like "open" or "close". Denote A as the matrix containing the preprocessed audio features with timing relationship. Before sent to S_1 and S_2, A is split into two random shares A_1 and A_2 for encryption and privacy, where $A = A_1 + A_2$. In the data processing procedure, T does nothing but generates uniformly random numbers. This means that T can be a light server or even a personal computer. Except random numbers, other values required for the additive secret sharing protocols are directly exchanged between S_1 and S_2. Let f denote the output of LSTM. After a series of computation, two edge servers output f_1 and f_2 to I. Thus, during the process of computation, all of the audio features or neural network parameters are computed in the form of ciphertext. And I is able to recover the output from encrypted status by simply computing $f = f_1 + f_2$. Similarly, if SP would like to terminate a training network in the current server provider, he can also retrieve the latest neural network parameters by calculating $p = p_1 + p_2$ and deploy them on the other edge servers.

4.2 Security Model

As for the security model, we adopt the standard semi-honest security model [36], which is also called passive or honest-but-curious model. According to the model, the two edge servers in our system execute the protocols as they are supposed to do [37]. However, they can still passively learn the information about the protocols, and driven by their curiosity, will try to get more benefits for themselves through known information.

Furthermore, we assume that at most one edge server can be corrupted in our model and no participant tries to collude with each other. Otherwise, the adversary can directly obtain the original audio features by simply adding the two stolen secret sharings. And even though S_1 and S_2 have enough cryptography knowledge, they can learn nothing but the preassigned parameters and the messages received from other participants in accordance with our proposed protocols. Naturally, it is rational to hypothesize that there are secure communication channels among the completely honest participants. In addition, what T can learn in the protocols are just some meaningless random numbers. Thus, as we suppose, T can be arbitrary honest lightweight end device. Last but not least, except the edge servers, other participants cannot be corrupted or dishonest.

5 SECRET SHARING BASED FUNCTIONS

Before constructing OPSR, we propose several new secure interaction sub-protocols in this section. In these protocols, the initialization phase is executed offline. The preparation phase and iteration phase are online operations. φ_i is used to output of the functions in iteation i. ξ_i and ς_i denote the intermediate values of iteration process.

5.1 Secure Vector Concatenation Function

In LSTM, there is an simple but important operation that concatenates two short vectors together to form a longer one. That is,

$$\begin{aligned}[u, v] &= [(u_0, u_1, u_2, ...), (v_0, v_1, v_2, ...)] \\ &= (u_0, u_1, ..., v_0, v_1, ..).\end{aligned} \tag{9}$$

It is clear to discover that S_1 and S_2 can just perform the secure vector concatenation function $SecCon(u, v)$ by computing $[u', v']$ and $[u'', v'']$ with no need for them to communicate with each other, because

$$\begin{aligned}SecCon(u, v) &= [u' + u'', v' + v''] \\ &= (u'_0 + u''_0 ..., v'_0 + v''_0 ...) \\ &= [u', v'] + [u'', v''].\end{aligned} \tag{10}$$

5.2 Secure Exponential Function with Base e

Let u be the encrypted input which is close to zero, and the output be the exponentiation with natural base e, $\varphi(u) = e^u$. Due to the fact that the base of the natural logarithm e is an approximate value, it is impossible to directly calculate the precise value of the function. Therefore, the only way to compute the function is to use the approximation functions to simulate it. In this paper, for secure computation, we adopt Maclaurin Series to approximate the function, because it is absolutely convergent for all complex numbers. The definition of the Maclaurin Series for exponentiation with natural base e is as follows.

$$\varphi(u) = e^u = \sum_{k=0}^{\infty} \frac{1}{k!} \cdot u^k. \tag{11}$$

From the equation above, the secure natural exponential function $SecExp(u)$ with can be formed by an iterative algorithm. Given the exponent u, what we have to calculate is $\xi_i = \xi_{i-1} \cdot u$, where $\xi_0 = u \cdot u$. If the required precision is ϵ, the iteration terminates as long as $\xi_i < \epsilon$.

Note that, given the precision ϵ, the smaller u is, the faster Maclaurin Series convergence speed is. And the ideal result is that the scale of the input u is kept over a relatively small field. Given the input is between 0 and 1 and the iteration number is 10 to 20, we can get an approximate value with an accuracy not less than 10^{-10}.

Initialization. S_1 and S_2 respectively get its uniformly random input u_1 and u_2, satisfying $u = u_1 + u_2$. Then, let i denote the round of iteration. S_1 computes $\varphi'_0 = 1 + 1 \cdot u_1$ and S_2 computes $\varphi''_0 = 1 \cdot u_2$. Given the precision ϵ, the iteration process is as follows.

Iteration. The iteration process is mainly implemented by alternatively invoking $SecMul(\cdot)$ and $SecAdd(\cdot)$. Firstly, S_1 and S_2 cooperatively compute $(\xi_0', \xi_0'') = SecMul(u, \frac{u}{2})$ and $\varphi_1 = SecAdd(\varphi_0, \xi_0)$. Then, repeatedly compute $\xi_i = SecMul(\xi_{i-1}, \frac{u}{i+2})$ and invoke the secure comparison protocol $SecCmp(\xi_i, \epsilon)$ to determine whether the precision meets the requirement. If so, terminate the iteration and output φ_i' and φ_i''. Otherwise, invoke the secure addition protocol to compute $\varphi_i = SecAdd(\varphi_{i-1}, \xi_{i-1})$ and continue. Using sigma function, the final result is written as

$$\varphi(u) = e^u \approx \sum_{i=0}^{\infty} \frac{1}{i!} \cdot u^i = 1 + \sum_{i=1}^{\infty} \xi_i. \tag{12}$$

5.3 Secure Reciprocal Function

Besides $SecExp(\cdot)$, the secure reciprocal function $SecInv(u)$ is also one of the fundamental "components" to implement the secure activation functions in LSTM. To preserve the global additivity of OPSR, we utilize Newton-Raphson iterative method to approximate $\varphi(u) = \frac{1}{u}$, where u is the divisor. Newton-Raphson method uses basic operations, multiplication and addition, to successively find better approximation to the root of reciprocal function, which is also included in the mathematic library of Matlab R2018b.

Similar to $SecExp(u)$, our secure reciprocal function is also made of an iterative process. Given the divisor u, the $SecInv(u)$ keeps trying to approximate the root of the equation $\phi(x) = \frac{1}{x} - u = 0$ by calculating $x_i = x_{i-1} \cdot (2 - u \cdot x_{i-1})$. What should be noted is that, to confirm convergence of the iteration , x_0 is an initial estimate in the range of $0 < x_0 < \frac{2}{u}$. In addition, for Newton-Raphson method, the more the number of iteration rounds is, the more precise the approximation is. And due to the fact that Newton-Raphson is quadratic convergent, it can be guaranteed that each iteration doubles the effective numbers of the approximation as long as the initial estimate satisfys the condition mentioned above.

Initialization. S_1 and S_2 respectively get its uniformly random input u_1 and u_2, satisfying $u = u_1 + u_2$. The trusted third party T generates a uniformly small random number r. Then, r is split into random shares $r = r_1 + r_2$, where $r_1 > 0$ and $r_2 > 0$. The shares of r_i are distributed to the corresponding edge server P_i.

Preparation. Prior to the iteration, S_i use the random value r_i to mask their inputs by computing $\alpha_1 = r_1 + u_1$ and $\alpha_2 = r_2 + u_2$. α_1 and α_2 are the masked results. And then send the masked values α_i each other. Next, S_1 and S_2 privately determine the initial estimate φ_0 by computing $\varphi_0' = \frac{1}{2(u_1+\alpha_2)}$ and $\varphi_0'' = \frac{1}{2(u_2+\alpha_1)}$, where $\varphi_0 = \varphi_0' + \varphi_0''$.

Iteration. S_1 and S_2 invoke the secure multiplication function to collaboratively calculate $(\xi_i', \xi_i'') = SecMul(\varphi_{i-1}, u)$. Then they set $\xi_i' = 2 - \xi_i'$ and $\xi_i'' = -\xi_i''$. Again, by invoking the secure multiplication function, the servers can get $\varphi_i = SecMul(\varphi_{i-1}, \xi_i)$. Repeat the calculation process above, until the function converges to the point where we want. Generally speaking, ten iterations can guarantee the precision reaches 10^{-9}. The whole iteration process can be written as the following mathematic equation.

$$\varphi(u) = \frac{1}{u} \approx \varphi_i = \varphi_{i-1} \cdot (2 - u \cdot \varphi_{i-1}). \tag{13}$$

5.4 Secure Square Root

Secure square root is an operations used in the initialization function to generate Gaussian distributed random matrix for LSTM. To perform the secure square root function $SecSqrt(u)$, we use Newton-Raphson iterative method to approximate $\varphi(u) = \sqrt{u}$. Compared with the secure reciprocal function, the difference is that the equation to solve becomes $\phi(x) = x^2 - u = 0$ and the iterative process changes into $x_i = \frac{1}{2} \cdot (x_{i-1} + \frac{u}{x_{i-1}})$ in $SecSqrt(u)$. In addition, there is an alternative iteration Binary Search Method that can find the square root, whose time complexity is $\mathcal{O}(\log \mathcal{N})$, same as Newton-Raphson method. However, to reach the same precision, this method always has to iterate more times. Due to the fast convergence speed, Newton-Raphson method is also the basis for the processor design and the computer graphics.

Initialization. The initialization process is identical to the secure reciprocal function, except that there is no need to generate and distribute random numbers.

Iteration. S_1 and S_2 alternatively invoke the secure multiplication function $SecMul(\cdot)$ and the secure reciprocal function $SecInv(\cdot)$ to complete the iteration. They first compute $(\varsigma_i', \varsigma_i'') = SecInv(\varphi_{i-1})$. Next, the secure multiplication function is used to calculate $(\xi_i', \xi_i'') = SecMul(\varsigma_i, u)$. Then, S_1 and S_2 complete one iteration by computing $\varphi_i = SecAdd(\frac{1}{2}\varphi_{i-1}, \xi_i)$. It is noted that φ_0 is initialized by $\varphi_0 = \frac{1}{2}u = \frac{1}{2}(u_1 + u_2)$. The convergence speed of the iteration is the same as the secure reciprocal function. And the whole iteration process can also be written as the form of the mathematic equation.

$$\varphi(u) = \sqrt{u} \approx \varphi_i = \frac{1}{2} \cdot (\varphi_{i-1} + \frac{u}{\varphi_{i-1}}). \tag{14}$$

5.5 Secure Natural Logarithm Function

For the logarithm function, we focus on the natural logarithm function $\varphi(x) = ln(x)$. It is because this kind logarithm is the most commonly used one in the modern natural science. And with the help of the change of base formula $\log_a b = \frac{\ln b}{\ln a}$, the common logarithm can also be transformed into the form of natural logarithm. To perform secure natural logarithm function $SecLog(\cdot)$, we adopt Maclaurin Series to approximate the final result. Let the input $v = \frac{u-1}{u+1}$, the formula of Maclaurin Series for the natural logarithm can be expressed as:

$$\varphi(u) = \ln u = \ln \frac{1+v}{1-v} = 2 \cdot \sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot u^{2k+1}. \tag{15}$$

Given the iteration equation, we repeatedly compute $\xi_i = \xi_{i-1} \cdot u^2$ and call the comparison function $SecCmp(\cdot)$ to judge when to terminate the iteration.

Initialization. The initialization process is identical to the secure square root function.

Preparation. S_1 and S_2 first compute the reciprocal of $u+1$ by invoking $d_i = SecInv(u+1)$. Then, transform the input u into $v = v_1 + v_2$, where $v_i = SecMul(u-1, d)$. Let $s = s_1 + s_2$ denote the square of u, where $s_i = SecMul(v, v)$.

Iteration. Just like the secure exponentiation function, the iterative computation is also composed of $SecMul(\cdot)$ and $SecAdd(\cdot)$. Let $\xi_0 = v$ and $\varphi_0 = 0$. S_1 and S_2

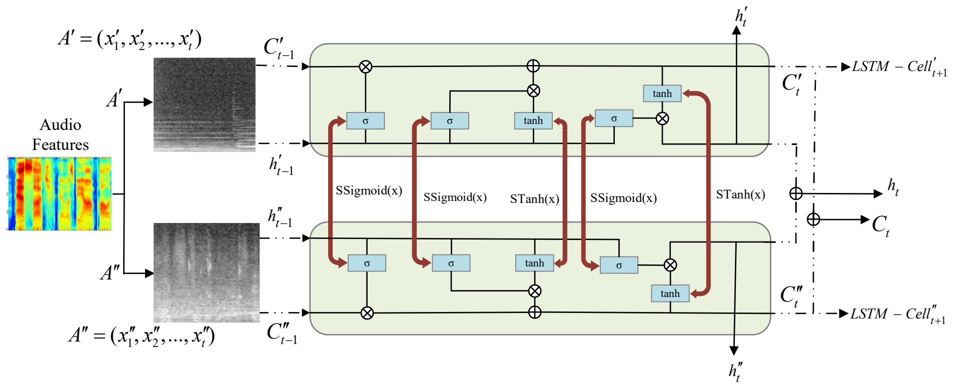

Fig. 3. Forward Propagation of LSTM

continually calculate $\xi_i = SecMul(\xi_{i-1}, s)$ and $\varphi_i = SecAdd(\varphi_{i-1}, \frac{1}{2i-1} \cdot \xi_{i-1})$ until the secure comparison protocol $SecCmp(\xi_i, \epsilon)$ return 0. Let R denote the final number of the iteration rounds. Note that the final output is $2 \cdot \varphi_R$ instead of φ_R. Using sigma function, the iteration process can be expressed as.

$$\varphi(u) = \ln u \approx 2 \cdot \sum_{i=0}^{\infty} \frac{1}{2i+1} \cdot u^{2i+1} = 2 \cdot \sum_{i=0}^{\infty} \xi_i. \tag{16}$$

5.6 Effectiveness of Maclaurin Series

Since Maclaurin Series converges very slow as the input is far away from 0, we have to further prove the inputs of $SecExp$ and $SecLog$ are close to 0. In section 6, it can be discovered that the inputs of these two protocols in OPSR are determined by three factors, including the audio features, the parameters of neural network and the outputs of sigmoid and tanh. And if all the three types of values are close to 0, it can be ensured that the inputs $SecExp$ and $SecLog$ here can never be far away from 0. Then, we have:

- According to the system model of OPSR, the audio features A are all preprocessed. As a consequence, it is reasonable to assume that data normalization, as the most fundamental preprocessing method [38], [39], has been applied to the audio features. And the normalization keeps the values of audio features between 0 and 1.
- To make the neural network to be able to learn significant information, it is a common realization that the parameters of neural network should be initialized with small random numbers, which are usually as small as 10^{-2}.
- Sigmoid function and tanh function can only output values between 0 and 1. Except for the calculation of the four intermediate states, the whole procedure of LSTM is about the multiplication of the two functions outputs.

In conclusion, it can be proved that $SecExp$ and $SecLog$ can reach its ideal convergence speed in OPSR.

5.7 Example of The Secure Function

To make the non-specialist easier to understand the core idea of our secure functions, in Table 1, a simple example of $SecExp$ proposed in Section 5.2 is given to show the detailed computation process. In the example, S_1 and S_2 get the secret shares $(1, 2)$ of input 3, and want to compute e^3. In each iteration, they invoke $SecAdd$ and $SecMul$ to update the iterative parameters. Finally, two random secret shares of e^3 are obtained. Moreover, the computation processes of other secure functions in the section are similar. For brevity, we only give one example of $SecExp$ here.

TABLE 1
An Example of Secure Exponentiation Function $SecExp$

Stages	Steps	Operations
Initialization	1	$u_1 \leftarrow 1, u_2 \leftarrow 2$
	2	$\varphi'_0 \leftarrow u_1 + u_2 = 1+1 = 2, \varphi''_0 \leftarrow u_2 = 1$
	3	S_1 inputs $u_1 = 1$ and $\frac{u_1}{2} = 0.5$ into $SecMul$; S_2 inputs $u_2 = 2$ and $\frac{u_2}{2} = 1$ into $SecMul$. And then, obtain $(\xi'_0, \xi''_0) \leftarrow (2, 2.5) = SecMul(3, 1.5)$. Since random values are generated to mask the secret shares in $SecMul(\cdot)$, we just randomly select two values to represent the secure multiplication result here.
	4	S_1 and S_2 invoke $SecAdd(\varphi_0, \xi_0)$ and compute $\varphi'_1 \leftarrow 2+2 = 4, \varphi''_1 \leftarrow 1+2.5 = 3.5$.
	5	S_1 and S_2 set the iteration index $i = 1$
Iteration	1	S_1 inputs $\xi'_{i-1} = 2$ and $\frac{u_1}{i+2} = \frac{1}{3}$ into $SecMul$; S_2 inputs $\xi''_{i-1} = 2.5$ and $\frac{u_2}{i+2} = \frac{2}{3}$ into $SecMul$. And then, obtain $(\xi'_i, \xi''_i) \leftarrow (3.5, 1) = SecMul(\xi_{i-1}, \frac{u}{i+2})$.
	2	S_1 and S_2 invoke $SecAdd(\varphi_i, \xi_i)$ and compute $\varphi'_{i+1} \leftarrow \varphi'_i + \xi'_i = 4 + 3.5 = 7.5$, $\varphi''_{i+1} \leftarrow \varphi''_i + \xi''_i = 3.5 + 1 = 4.5$.
	3	$i \leftarrow i+1$ and go for next iteration.
Output	1	S_1 outputs 12.0513, and S_2 outputs 8.0342, where $12.0513 + 8.0342 = 20.0855 \approx e^3$.

6 Privacy-Preserving LSTM for Encrypted Audio Features

In this section, we propose the privacy-preserving framework for forward propagation (FP) and back propagation

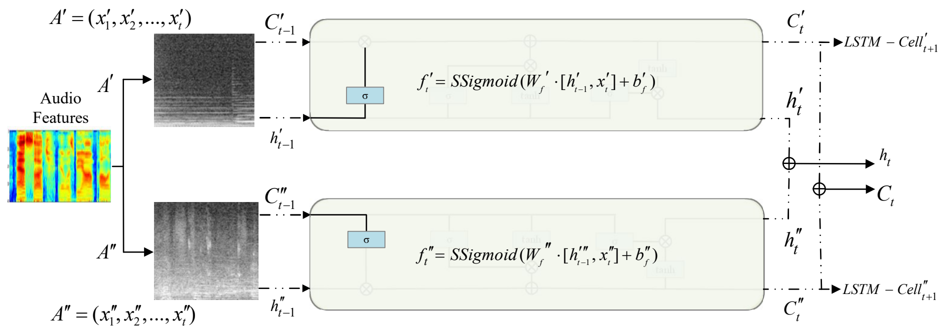
$A' = (x'_1, x'_2, ..., x'_t)$
Audio Features
A'
A''
C'_{t-1}
h'_{t-1}
σ
$f'_t = SSigmoid(W'_f \cdot [h'_{t-1}, x'_t] + b'_f)$
C'_t
h'_t
$LSTM-Cell'_{t+1}$
h_t
C_t
C''_{t-1}
h''_{t-1}
σ
$f''_t = SSigmoid(W''_f \cdot [h''_{t-1}, x''_t] + b''_f)$
h''_t
C''_t
$LSTM-Cell''_{t+1}$
$A'' = (x''_1, x''_2, ..., x''_t)$

Fig. 4. Forget Gate

through time (BPTT) of LSTM. FP is applied in both the pre-trained and the training neural network. BPTT is only used for training process of LSTM. For simplicity, the notion with x' and x'' are used to mask the shares distributed to the two edge servers. The matrix addition and multiplication are respectively calculated by $SecAdd(\cdot)$ and $SecMul(\cdot)$. Specially, due to the fact that the intelligent IoT voice control service provider would not like to reveal the trained network, the weight matrixes and biaes are not publicly available to all the participants. They are split into secret sharing $W_k = W'_k + W''_k$ and $b_k = b'_k + b''_k$ for the two edge servers, where b_k is bias vector and W_k is weight matrix and $k \in \{f, i, C, O\}$. In addition, W_{kh} and W_{kx} denote the split matrixes of W_k, where $W_k = [W_{kh}, W_{kx}]$.

6.1 Secure Forward Propagation of LSTM

A hidden neuron of the standard Long Short-Term Memory recurrent network is usually composed of three gates, forget gate, input gate and output gate. Each gate deploys at least one nonlinear function and plenty of linear functions to process the information from the previous recurrence. Since all of the necessary secure "components" have been proposed, the remaining work for the secure forward propagation of LSTM is just to appropriately combine these functions to design the secure interactive sub-protocols. As shown in Fig.3 and the corresponding pseudo codes in Protocol 1, given the audio feature data with time-series $A = (x_0, x_1, ..., x_t)$, S_1 and S_2 complete the computation of the three gates of LSTM by invoking the secure functions proposed in Section 3, Section 5 and Section 6.1.1. The outputs of cell at time t, which are C_t and h_t, are subsequently used to update the cell state and output at time $t+1$. The symbols about the three gates in the section have the same meanings as Section 3. The details of the three gates privacy-preserving computation are given as follows.

6.1.1 *Secure Sigmoid and Tanh*

To maintain nonlinearity, LSTM deploys at least one activation function in each of the gates. Therefore, we have to discuss the activation functions firstly. There are three kinds of commonly used activation functions in neural network, sigmoid, tanh and ReLu. Unlike the disappointing performance in the convolutional neural network (CNN), the sigmoid function and the tanh function successfully avoid the problem of gradient explosion in the recurrent neural network. As for ReLu, it is rarely applied in RNN for its special mathematical characteristic. Thus, for LSTM, what we have to concern is only how to securely compute $\sigma(x)$ and $tanh(x)$. For simplicity, we use $SSigmoid(x)$ and $STanh(x)$ to represent these two secure functions in the remaining article. In addition, we implement the protocols of $SSigmoid(x)$ and $STanh(x)$ based on the Eq.2 and Eq.5, respectively.

Protocol 1 Secure Forward Propagation of LSTM

```
Input: $S_1$ has encrypted audio feature data $(x'_0, x'_1, ..., x'_t)$,
    $S_2$ has encrypted audio feature data $(x''_0, x''_1, ..., x''_t)$,
    $x_t = x'_t + x''_t$ is the input data at time $t$;
Output: $S_1$ outputs the current cell state $C'_t$ and the final
    output $h'_t$; $S_2$ outputs the current cell state $C''_t$ and the
    final output $h''_t$.
1: Set index $\gamma = 1$ and initialize the shares of $f_0$, $i_0$, $h_0$, $\widetilde{C_t}$
   $C_0$, $O_0$.
2: while $\gamma < t$ do
3:     $S_1$ and $S_2$ compute the forget gate by invoking
       $(f'_\gamma, f''_\gamma) = SSigmoid(W_f \cdot [h'_{\gamma-1}, x_\gamma] + b_f)$.
4:     $S_1$ and $S_2$ compute the inpute gate by invoking
       $(i'_\gamma, i''_\gamma) = SSigmoid(W_i \cdot [h_{\gamma-1}, x_\gamma] + b_i)$.
5:     $S_1$ and $S_2$ obtain the candidate cell state by computing $(\widetilde{C'_\gamma}, \widetilde{C''_\gamma}) = STanh(W_C \cdot [h_{\gamma-1}, x_\gamma] + b_C)$.
6:     The cell state is updated by letting $S_1$ and $S_2$ compute $(C'_\gamma, C''_\gamma) = f_\gamma \times C_{\gamma-1} + i_\gamma \times \widetilde{C}_\gamma$.
7:     $S_1$ and $S_2$ compute the inpute gate by invoking
       $(O'_\gamma, O''_\gamma) = SSigmoid(W_O \cdot [h_{\gamma-1}, x_\gamma] + b_O)$.
8:     The output of LSTM at time $\gamma$ is determined by
       letting $S_1$ and $S_2$ compute $(h'_\gamma, h''_\gamma) = O_\gamma \cdot STanh(C_\gamma)$
9:     Go for next iteration.
10: end while
11: $S_1$ and $S_2$ return $(C'_t, h'_t)$ and $(C''_t, h''_t)$, respectively.
```

Secure Sigmoid Function. Sigmoid function is also known as Logistic function. Since its distribution is consistent to the excitatory and inhibitory states of biological neurons, it is once thought to be the kernel of the artificial neural network. Let u denote the input. To implement $SSigmoid(u)$ with additive secret sharing, the two edge servers S_1 and S_2 first have to compute $(q_1, q_2) = SecExp(-u)$. Then, S_1 sets $q_1 = 1 + q_1$. By invoking the secure reciprocal function $SecInv(\cdot)$, the final output is determined by

$$SSigmoid(u) = SecInv(q_1 + 1, q_2) = \frac{1}{1 + e^{-u}}. \quad (17)$$

Secure Tanh Function. Similar to the sigmoid function, Tanh function also maintain the nonlinear monotonically ascending and descending relationship between input and output. The difference is that it is not such sensitive to value change in the range $[-1, 1]$, which is more in line with the law of human brain nerve saturation. As far as the secure sigmoid function has been completed, the Tanh function can be simply computed by the formula $tanh(x) = 2\sigma(2x) - 1$. Given the input u, that is

$$\begin{aligned} STanh(u) &= 2SSigmoid(u) - 1 \\ &= \frac{e^u - e^{-u}}{e^u + e^{-u}} = \frac{2}{1 + e^{-2u}} - 1. \end{aligned} \tag{18}$$

6.1.2 Forget Gate

The forget gate determines how much cell state information C_{i-1} of the previous cell is preserved in the current cell. As illustrated in Fig.4, to achieve this, the input x_i is put into the sigmoid function after a series of linear operations. And the output which is in the range of 0 to 1 acts on C_{i-1}. Due to the fact that weight matrixed W_f and bias b_f for the forget gate are not publicly known, at time t, S_1 computes

$$\begin{aligned} f'_t &= SSigmoid(W'_f \cdot [h'_{t-1}, x'_t] + b'_f) \\ &= SSigmoid(W'_{fh} \cdot h'_{t-1} + W'_{fx} \cdot x'_t + b'_f) \\ &= (1 + e^{-W'_{fh} \cdot h'_{t-1} - W'_{fx} \cdot x'_t - b'_f})^{-1}. \end{aligned} \tag{19}$$

And S_2 computes:

$$\begin{aligned} f''_t &= SSigmoid(W''_f \cdot [h''_{t-1}, x''_t] + b''_f) \\ &= SSigmoid(W''_{fh} \cdot h''_{t-1} + W''_{fx} \cdot x''_t + b''_f) \\ &= (1 + e^{-W''_{fh} \cdot h''_{t-1} - W''_{fx} \cdot x''_t - b''_f})^{-1}. \end{aligned} \tag{20}$$

Note that LSTM is a kind of time-ordered neural network. Thus, we use x_t to denote the input at time t.

6.1.3 Input Gate

There are three tasks done in the input gate. The first one is to compute a filtered input vector. Again, we use our secure sigmoid function to complete this work. Fig.5 shows that the only difference between the forget gate and the input gate is that the parameters are changed. Given the input weight matrix W_i, input bias b_i and time t, S_1 computes

$$\begin{aligned} i'_t &= SSigmoid(W'_i \cdot [h'_{t-1}, x'_t] + b'_i) \\ &= SSigmoid(W'_{ih} \cdot h'_{t-1} + W'_{ix} \cdot x'_t + b'_i) \\ &= (1 + e^{-W'_{ih} \cdot h'_{t-1} - W'_{ix} \cdot x'_t - b'_i})^{-1}. \end{aligned} \tag{21}$$

And S_2 computes

$$\begin{aligned} i''_t &= SSigmoid(W''_i \cdot [h''_{t-1}, x''_t] + b''_i) \\ &= SSigmoid(W''_{ih} \cdot h''_{t-1} + W''_{ix} \cdot x''_t + b''_i) \\ &= (1 + e^{-W''_{ih} \cdot h''_{t-1} - W''_{ix} \cdot x''_t - b''_i})^{-1}. \end{aligned} \tag{22}$$

Then, by invoking the tanh function, a candidate cell state vector is determined to control how much input information is adopted by the current cell. The details about the computation process are as follows. S_1 calculates

$$\begin{aligned} \widetilde{C'_t} &= STanh(W'_C \cdot [h'_{t-1}, x'_t] + b'_C) \\ &= STanh(W'_{Ch} \cdot h'_{t-1} + W'_{Cx} \cdot x'_t + b'_C) \\ &= 2(1 + e^{-W'_{Ch} \cdot h'_{t-1} - W'_{Cx} \cdot x'_t - b'_C})^{-1} - 1. \end{aligned} \tag{23}$$

And S_2 calculates

$$\begin{aligned} \widetilde{C''_t} &= STanh(W''_C \cdot [h''_{t-1}, x''_t] + b''_C) \\ &= STanh(W''_{Ch} \cdot h''_{t-1} + W''_{Cx} \cdot x''_t + b''_C) \\ &= 2(1 + e^{-W''_{Ch} \cdot h''_{t-1} - W''_{Cx} \cdot x''_t - b''_C})^{-1} - 1. \end{aligned} \tag{24}$$

Finally, according to the previous cell state C_{i-1}, the new coming input vector i_t and the candidate cell state vector $\widetilde{C_t}$, the cell state is updated by calling the secure multiplication function and the secure addition function. S_1 has one share of the current cell state

$$C'_t = f'_t \times C'_{t-1} + i'_t \times \widetilde{C'_t}. \tag{25}$$

And S_2 has the other share of the current cell state

$$C''_t = f''_t \times C''_{t-1} + i''_t \times \widetilde{C''_t}. \tag{26}$$

6.1.4 Output Gate

From the description above, it can be derived that the output of LSTM is simultaneously influenced by the long-term memory and the current input. Thanks to the control of forget gate, the network can preserve information a long time ago. Due to the existence of the input gate, insignificant contents from the new coming data are immediately discarded. And all of the facts lead to the following computation of output gate. As shown in Fig.6, an output weight matrix W_o and an output bias b_o are utilized to compute the candidate output vector. We indicate S_1 to calculate:

$$\begin{aligned} O'_t &= SSigmoid(W'_O \cdot [h'_{t-1}, x'_t] + b'_O) \\ &= SSigmoid(W'_{Oh} \cdot h'_{t-1} + W'_{Ox} \cdot x'_t + b'_O) \\ &= (1 + e^{-(W'_{Oh} \cdot h'_{t-1} - W'_{Ox} \cdot x'_t - b'_O})^{-1}, \end{aligned} \tag{27}$$

and S_2 to calculate:

$$\begin{aligned} O''_t &= SSigmoid(W''_O \cdot [h''_{t-1}, x''_t] + b''_O) \\ &= SSigmoid(W''_{Oh} \cdot [h''_{t-1} + W''_{Ox} \cdot x''_t + b''_O) \\ &= (1 + e^{-W''_{Oh} \cdot [h''_{t-1} - W''_{Ox} \cdot x''_t - b''_O})^{-1}. \end{aligned} \tag{28}$$

Then, the final output is determined by both the candidate output O_t and the current cell state C_t. To compute it, we have to depend on the secure multiplication function and the secure tanh function once more. That is to say, we let S_1 and S_2 respectively compute:

$$\begin{aligned} h'_t &= O'_t \cdot STanh(C'_t) \\ &= 2(1 + e^{-C'_t})^{-1} - 1, \end{aligned} \tag{29}$$

and

$$\begin{aligned} h''_t &= O''_t \cdot STanh(C''_t) \\ &= 2(1 + e^{-C''_t})^{-1} - 1. \end{aligned} \tag{30}$$

Both h'_t and h''_t are then sent to the smart IoT device I as feedback. And I can decrypt the ciphertext by simply adding them together, $h_t = h'_t + h''_t$.

6.2 Back Propagation Based Training of LSTM

6.2.1 Gaussian Initialization

The starting point of the training for LSTM is always the initialization of the weight matrixes. An appropriate initialization method cannot only increase the accuracy of the results, but also improve the convergence speed and avoid

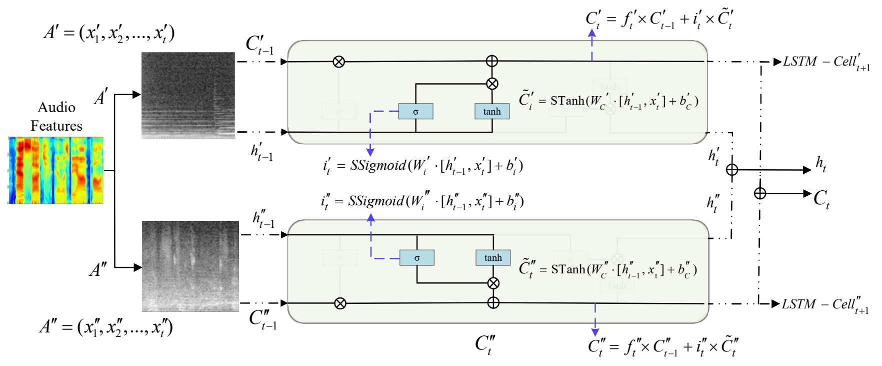
$A' = (x'_1, x'_2, ..., x'_t)$
Audio Features
A'
A''
$A'' = (x''_1, x''_2, ..., x''_t)$
C'_{t-1}
h'_{t-1}
h''_{t-1}
C''_{t-1}
$C'_t = f'_t \times C'_{t-1} + i'_t \times \tilde{C}'_t$
$\tilde{C}'_t = \mathrm{STanh}(W'_C \cdot [h'_{t-1}, x'_t] + b'_C)$
$i'_t = SSigmoid(W'_i \cdot [h'_{t-1}, x'_t] + b'_i)$
$i''_t = SSigmoid(W''_i \cdot [h''_{t-1}, x''_t] + b''_i)$
$\tilde{C}''_t = \mathrm{STanh}(W''_C \cdot [h''_{t-1}, x''_t] + b''_C)$
$C''_t = f''_t \times C''_{t-1} + i''_t \times \tilde{C}''_t$
C''_t
σ
tanh
h'_t
h''_t
h_t
C_t
$LSTM - Cell'_{t+1}$
$LSTM - Cell''_{t+1}$

Fig. 5. Input Gate

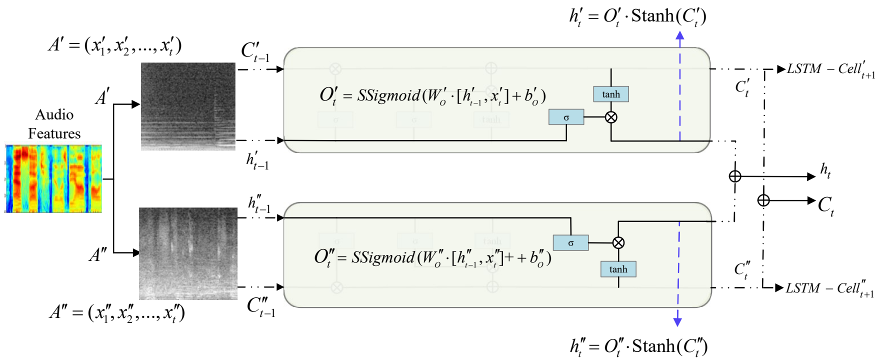
$h'_t = O'_t \cdot \mathrm{Stanh}(C'_t)$
$A' = (x'_1, x'_2, ..., x'_t)$
Audio Features
A'
A''
$A'' = (x''_1, x''_2, ..., x''_t)$
C'_{t-1}
h'_{t-1}
h''_{t-1}
C''_{t-1}
$O'_t = SSigmoid(W'_o \cdot [h'_{t-1}, x'_t] + b'_o)$
$O''_t = SSigmoid(W''_o \cdot [h''_{t-1}, x''_t] + + b''_o)$
σ
tanh
C'_t
C''_t
h_t
C_t
$LSTM - Cell'_{t+1}$
$LSTM - Cell''_{t+1}$
$h''_t = O''_t \cdot \mathrm{Stanh}(C''_t)$

Fig. 6. Output Gate

the problems of vanishing gradient and exploding gradient. In LSTM, the most commonly used initialization method is the random Gaussian initialization. Consequently, we have to propose a secure random generator whose output conforms to the Gaussian-distribution. Considering the random generation algorithms, Box Muller Method is one of the most effective ones that satisfies our need. By utilizing it, the uniformly distributed random numbers received from the trusted third party T can be directly transformed to be Gaussian distributed. Therefore, we use the secret sharing based computation functions to remodel the Box Muller Method, and then, build a secure Gaussian distributed random generator $GInit(\cdot)$.

Initialization. The trusted third party T generates two uniformly random number θ and r, where r satisfys $0 < r < 1$. Then, both of them are splits them into random shares $\theta = \theta_1 + \theta_2$ and $r = r_1 + r_2$. The shares of θ_i and r_i are sent to the corresponding edge server P_i.

Random Generation. Let $r_1 = 1 - r_1$. S_1 and S_2 collaboratively calculate $p_i = SecLog(r)$ and $q_i = SecSqrt(-2 \cdot p)$. Then, θ is process by $u_i = SecSin(2\pi\theta)$ and $v_i = SecCos(2\pi\theta)$. And the secure multiplication is invoked twice to compute $o_1 = SecMul(q, u)$ and $o_2 = SecMul(q, v)$. o_1 and o_2 are two random numbers that belong to two independent Gaussian distributions. To initialize the weight matrixes, we have to repeatedly execute this function until all values are assigned.

Note that, besides the secure functions proposed in the previous sections, we still have to use secure sine function $SecSin(\cdot)$ and cosine function $SecCos(\cdot)$ in $Ginit(\cdot)$. And both of them can be computed by two similar Maclaurin Series

$$sin(x) = \sum_{i=0}^{\infty} \frac{(-1)^n}{2i+1} x^{(2i+1)!}, \tag{31}$$

and

$$cos(x) = \sum_{i=0}^{\infty} \frac{(-1)^n}{2i} x^{(2i)!}. \tag{32}$$

It can be discovered these two sum formulas are almost same as the series we use in the design of secure natural logarithm function $SecLog(\cdot)$. Therefore, for sake of brevity, we do not discuss the details of these two functions here.

6.2.2 BP Based Computation of LSTM

The training process of LSTM is based on the back propagation through time (BPTT) algorithm. Fortunately, both of sigmoid function $\sigma(x)$ and tanh function $\tanh(x)$ have excellent mathematical properties, $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ and $\tanh'(x) = 1 - \tanh^2(x)$. That means their derivatives are functions of their original function. Furthermore, BPTT algorithm is one of the application of the Gradient Descent (GD) algorithm. And GD algorithm is composed of the linear operations with respect to the derivatives. Consequently, we can simply use the original function $\sigma(x)$ and $\tanh(x)$ to denote the BPTT based training process of LSTM. Namely, the securely training process of LSTM can be simply implemented by invoking our proposed additive secret sharing protocols, as what we have done in the forward propagation. Thus, for brevity's sake, we do not give the detailed derivation process, but only list the final

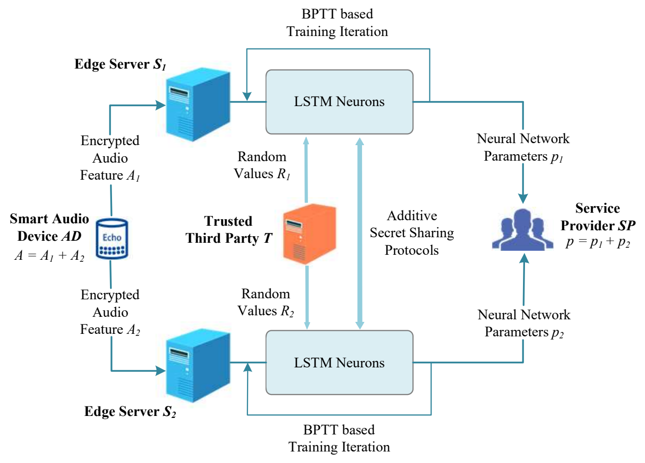
BPTT based Training Iteration
Edge Server S_1
LSTM Neurons
Encrypted Audio Feature A_1
Random Values R_1
Neural Network Parameters p_1
Smart Audio Device AD
$A = A_1 + A_2$
Echo
Trusted Third Party T
Additive Secret Sharing Protocols
Service Provider SP
$p = p_1 + p_2$
Encrypted Audio Feature A_2
Random Values R_2
Neural Network Parameters p_2
LSTM Neurons
Edge Server S_2
BPTT based Training Iteration

Fig. 7. Back Propagation Based Training of LSTM

computation formulas. As for the original weight matrixes for training, we can repeatedly invoke the secure random Gaussian initialization function $GInit(\cdot)$ to initialize them. In addition, the process of BPTT is as shown in Fig.7. Given the encrypted audio features A_1 and A_2, the edge servers first complete secure FP computation of LSTM neurons. Then, based on the computation results of FP, they execute the iteration protocol of secure BPTT with the help of the trusted third party and output the encrypted loss or other information to the speech recognition service provider. Specially, the computation direction of BPTT is opposite to FP.

Assume that the iterative forward propagation of privacy-preserving LSTM has been finished. Let δ_{t-1} denote the error vector at time $t-1$. It can be calculated by a partial derivative function with respect to the output h_t at time t.

$$\begin{aligned}\delta_{t-1} &= \delta_{O,t}W_{Oh} + \delta_{f,t}W_{fh} + \delta_{i,t}W_{ih} + \delta_{\widetilde{C},t}W_{Ch} \\ &= f(h_{t-1}).\end{aligned} \tag{33}$$

Respectively, $\delta_{O,t}$, $\delta_{f,t}$, $\delta_{i,t}$ and $\delta_{\widetilde{C},t}$ denote the partial derivative with respect to h_{t-1}. And they can be computed by the following formulas with our additive secret sharing protocols. Among them, $\widetilde{C_t}$, f_t, i_t and O_t are obtained during the forward propagation.

$$\delta'_{O,t}, \delta''_{O,t} = \delta_t \cdot STanh(C_t) \cdot O_t \cdot (1 - O_t), \tag{34}$$

$$\delta'_{f,t}, \delta''_{f,t} = \delta_t \cdot O_t \cdot STanh'(C_t) \cdot C_{t-1} \cdot f_t \cdot (1 - f_t), \tag{35}$$

$$\delta'_{i,t}, \delta''_{i,t} = \delta_t \cdot O_t \cdot STanh'(C_t) \cdot \widetilde{C} \cdot i_t \cdot (1 - i_t), \tag{36}$$

$$\delta'_{\widetilde{C},t}, \delta''_{\widetilde{C},t} = \delta_t \cdot O_t \cdot STanh'(C_t) \cdot i_t \cdot (1 - \widetilde{C}^2), \tag{37}$$

$$STanh'(C_t) = 1 - STanh(C_t)^2, \tag{38}$$

where δ_t is the error vector at time t.

Let the batch size be k and the original error vector $\delta_k = 1$. Then, we can get the error at time no more than k by iteratively computing Eq.(33).

Given the errors and $\kappa \in \{O, f, i, C\}$, the gradients of weight matrixes W_κ and biases b_κ are able to be simply derived as follows. Let ∇ denote the symbol of gradient and $[\nabla W_{\kappa h}, \nabla W_{\kappa x}] = \nabla W_\kappa = [\nabla W'_{\kappa h} + \nabla W''_{\kappa h}, \nabla W'_{\kappa x} + \nabla W''_{\kappa x}]$. We have

$$\nabla W'_{\kappa h}, \nabla W''_{\kappa h} = \sum_{j=1}^{k} SecMul(\delta_{\kappa,j}, h_{t-1}), \tag{39}$$

$$\nabla W'_{\kappa x}, \nabla W''_{\kappa x} = SecMul(\delta_{\kappa,j}, x_t), \tag{40}$$

$$\nabla b'_\kappa, \nabla b''_\kappa = \sum_{j=1}^{k} \delta_{\kappa,j}. \tag{41}$$

Assume the learning rate of LSTM is LR and publicly available. Then, we can use the gradient values to update weight matrixes and biases.

$$W'_{new,\kappa}, W''_{new,\kappa} = W_{old,\kappa} - \nabla W_\kappa \cdot LR, \tag{42}$$

$$b'_{new,\kappa}, b''_{new,\kappa} = b_{old,\kappa} - \nabla b_\kappa \cdot LR. \tag{43}$$

Different form the forward propagation, all of the encrypted parameters are sent to the service provider SP, not the smart IoT device I. Because, as mentioned before, these information has highly commercial value and the operator would not like to share them with others. What is similar is that they are also additive and able to be decrypted by adding the corresponding values.

6.3 Feasibility of Multiparty Computation

Our OPSR is two-party setting. If more edge servers are introduced, it is not workable under the current configuration. But for completeness, we further discuss the feasibility to extend to multi-party computation. OPSR is totally based on the four basic secure functions proposed in 5. To maintain the additivity of their sharings, we utilize Maclaurin Series and Newton-Raphson method to approximate these functions. And the approximation is only composed of additions and multiplications. For the addition, it is trivial to prove that it can be securely calculated by multiple participants. Thus, the feasibility of multi-party computation for OPSR depends heavily on the $SecMul(\cdot)$. Then, due to the fact that $SecMul(\cdot)$ is originally designed for multiparty in [34], it can be deduced that, if more edge servers are introduced, our framework would still be workable with some improvements.

7 Correctness And Security Analysis

7.1 Correctness of OPSR

At the begining of OPSR, the audio features A is split into $A = A_1 + A_2$. Then, plenty of linear and nonlinear operations work on A under our self-designed protocols. Austerely, the final output p and f may not be identical to the values out of the original algorithm. Here, we give theoretical derivation to prove the outputs of our framework are the exact values.

Firstly, the protocol outlined in Section 3 have been proved that, no matter they are invoked how many times, the output is still correct. Secondly, the four function proposed in Section 5, except $SecCon(\cdot)$, utilize the algorithms that are commonly used in CPU and mainstream mathematical softwares to approximate the real output. Meanwhile, the operations used in all the functions except $SecInv(\cdot)$

are nothing but multiplication and addition. For $SecInv(\cdot)$, it actually calls an extra function $SecRec(\cdot)$ which is also one of the four functions. Thus, theoretically, our functions is able to reach arbitrary degree of accuracy if power allows. And as long as the accuracy reaches at least the same degree as the commonly required for neural network, it can be said that our proposed functions are both additive and as correct as the original functions. Furthermore, the activation functions $SSigmoid(\cdot)$ and $STanh(\cdot)$ are composed of the linear combination of the above functions. That means the outputs of them ϕ satisfies $\phi = \phi_1 + \phi_2$. Finally, we can conclude, given Ψ as an arbitrary function, we have $\Psi = \Psi_1 + \Psi_2$, if and only if $\Psi = \psi(\chi_1, \chi_2, ...)$ where ψ is an arbitrary linear mapping and χ_i $(i = 1, 2, ..)$ can be any secure function in this paper. Then, according to the inference, we can guarantee $p = p_1 + p_2$ and $f = f_1 + f_2$, because both the forward propagation and back propagation can be regarded as the Ψ.

7.2 Security of OPSR

To prove the security of the protocols in this paper, we first have to formally define what is semi-honest security [32].

Definition 1. *We say that a protocol π is secure if there exists a probabilistic polynomial-time simulator S that can generate a view for the adversary A in the real world and the view is computationally indistinguishable from its real view.*

Moreover, besides the *Lemma 1* mentioned in Section 3, we also need the following lemmas.

Lemma 2 [32]. *If a random element r is uniformly distributed on Z_n and independent from any variable $x \in Z_n$, then $r \pm x$ is also uniformly random and independent from x.*

Lemma 3 *The protocols $SecAdd$, $SecMul$ and $SecCmp$ are secure in the semi-honest model.*

Due to the fact that $SecCon$ protocol is only performed locally, it is trivial to prove its security according to *Definition 1*. We just have to prove the security of the other protocols.

Theorem 1. *The protocols $SecExp$, $SecInv$, $SecSqrt$ and $SecLog$ are secure in the semi-honest model.*

proof. In $SecExp$, given the number of iteration round τ, what S_1 holds is $View_1 = (u_1, \mathcal{G}'_1, \mathcal{F}'_1, \epsilon_1)$, where $\mathcal{G}'_1 = \{\xi'_0, \xi'_1, ..., \xi'_\tau\}$ and $\mathcal{F}'_1 = \{\varphi'_0, \varphi'_1, ..., \varphi'_{\tau-1}\}$. And ξ'_i and φ'_i are respectively the outputs of $SecMul$ and $SecAdd$. Meanwhile, with u_1, they also compose the inputs of the next iteration. According to *Lemma 3*, it is guaranteed that $\mathcal{G}'_1$ and $\mathcal{F}'_1$ are sets of uniformly random values. Thus, all of them can be perfectly simulated by the simulator S_1, and are unable to distinguished by the adversary A in polynomial time. Similarly, S_2 can also hold $View_2$ which is simulatable and indistinguishable. As mentioned in the last sub-section, the other three protocols are implemented through similar polynomials composed of protocols that are proved to be secure above. Consequently, we give the details of their proof in appendix. □

Theorem 2. *The protocols $SSigmoid$ and $STanh$ are secure in the semi-honest model.*

proof. For $SSigmoid$, the views of S_1 and S_2 are $View_1 = (u_1, q_1)$ and $View_2 = (u_2, q_2)$. The final oupt is $\sigma_i = SecInv(f) = SecInv(1 + SecExp(-u))$. Because $SecInv$ and $SecExp$ have been proved secure in *Theorem 1*, σ_i as well as q_i and u_i are uniformly random and simulatable. As a consequence, $View_1$ is simulatable and computationally indistinguishable from the view simulated by simulator for the adversary. In the same way, it is deduced that the simulator can also generate a polynomial-time indistinguishable view for $View_2$. Consequently, we can obtain the conclusion that $SSigmoid$ is secure in the semi-honest model. Furthermore, due to $STanh = 2 \cdot SSigmoid(2u) - 1$, $STanh$ can be locally computed based on the output of $SSigmoid$ without need of any protocols to exchange data. Therefore, $STanh$ is as secure as $SSigmoid$. □

Theorem 3. *The interactive protocols for forward propagation of LSTM are secure in the semi-honest model.*

proof. In the process of forward propagation, all the outputs of the three gates are finally calculated by the simulatable building block $SSigmoid$ or $STanh$. The inputs of the two secure functions are obtained from the computation results of $SecAdd$ and $SecMul$. According to *Lemma 2* and *Lemma 3*, all the inputs can be regarded as random elements, which means they can also be simulated by the simulators. Then, due to the security of $SSigmoid$ and $STanh$ proved in *Theorem 2*, we can deduce that the real views of the interactive protocols and simulated views by simulators are disguishable for the adversary. □

Theorem 4. *The interactive protocols for back propagation through time of LSTM are secure in the semi-honest model.*

proof. Compared with forward propagation, the only computational difference for back propagation is involved $GInit$. Therefore, to prove the security of the interactive protocols for back propagation, we just have to prove $GInit$ is secure in the semi-honest model and the other proof is completely the same as the proof of *Theorem 3*. For $GInit$, the real view of S_1 is $View_1 = (\theta_1, r_1, p_1, o_1, u_1, v_1)$. Among the elements of $View_1$, θ_1 and r_1 are uniformly random values generated by trusted third party. The others are outputs of secure protocols proved to be secure in *Theorem 1* and *Lemma 3*. Therefore, the real view of S_1 is computational distinguishable from the view simulated by the simulator. Similarly, it can be proved that $View_2$ is also simulatable and computational distinguishable for the adversary. □

8 Experiments

In this section, we present the experimental results of the secret sharing based secure sigmoid/tanh function and the privacy-preserving LSTM interactive protocols. To implement our framework, we utilize NumPy for parallel computation of matrixes in Python 3. The audio data is part of TIMIT corpus with 123 coefficients which are from a Fourier-transform-based filter-bank [40]. All the data is encrypted on an embedded chip called Raspberry Pi 3 Model B (RP3B) which is commonly used in the Dueros open platform for smart IoT devices [41]. Then, the ciphertexts are respectively sent to two edge servers for privacy-preserving LSTM training and pre-trained IoT speech recognition. Each server is equipped with an Intel(R) Core(TM) i5-7400 CPU @3.00GHz and 8.00GB of RAM.

8.1 Performance of OPSR

The proposed privacy-preserving LSTM for intelligent IoT device voice control consists of two parts, secure forward

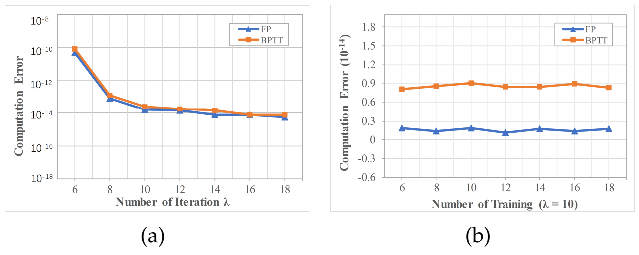

Fig. 8. Computation errors of forward propagation and backward propagation: (a) errors for different iterative numbers λ of secure functions; (b) errors for different training steps of LSTM with $\lambda = 10$.

propagation and secure backward propagation through time. For a pre-trained LSTM neural network, we only have to do the forward computation. But for training, both of two need to be calculated. Therefore, we try to compare their performance under different conditions to evaluate their accuracy and efficiency.

Both FP and BP call the secure sigmoid function and the secure tanh function for more than one times during computation. After a series of linear and nonlinear operations, the errors of them may spread and lead to the inaccuracy of the final output. However, from Fig. 8 (a) and Fig.9, it can be seen that, with different iteration numbers λ for the functions in Section 5, the errors of FP and BPTT do not decrease as fast as the secure sigmoid function. However, the values of FP and BPTT computation errors are much smaller. This is caused by the method we adopt to initialize the weight matrixes. In our experiments, the weight matrixes are set to Gaussian distributed random numbers between -10^{-2} and 10^{-2}. And this is a common method during training the neural network and can help improve the convergence speed. In addition, Fig. 8 (b) depicts that the number of training steps is almost unable to affect the errors of our framework. In this way, we guarantee that the errors do not increase along with the convergence of the neural network. It is further proved that our framework is valid for neural network training and the error is completely negligible with only a few iteration.

TABLE 2
Runtime and message size of privacy-preserving LSTM for each training sample

Stage	Runtime Per Sample(ms)		Size Per Sample(MB)	
	Initialization	Iteration	Initialization	Iteration
Encryption on RP3B	0.9	-	-	-
Forward Propagation	92.95	189.39	1.08	0.94
Backward Propagation	37.16	74.78	0.39	0.25

Furthermore, we evaluate the efficiency and overhead of our privacy-preserving framework. Given the default length of data $\ell = 32$ and the number of iteration 10, we apply 2000 audio data with 123 features on our neural network which has 32 neural units. And the time step is set to 8. Thus, the size of weight matrix W_κ, where $\kappa \in \{O, f, i, C\}$. is 155×32. And the size of output and bias are respectively 8×32 and 1×32. As shown in Table 2, each training sample only needs no more than one second runtime and about two millibytes communication load. The extra overhead for local intelligent voice control device RP3B is about 1 millisecond runtime for encryption and mainly caused by generating random values. Specially, the overhead of BPTT is much less than FP in our framework. This is because the most time-consuming operation $Res_{inter} = \delta_t \cdot O_t \cdot STanh'(C_t)$, which is involved in all the three gates of BPPT in LSTM, has to be calculate only once. This also explains why we do not respectively list the runtime and message size for each gate of BPTT in Table 3.

We then respectively evaluate the performance of the three gates for our privacy-preserving LSTM network that is performed by the two edge servers. Table 3 shows the runtime and message size of different gates for processing one audio sample. It can be discovered that the three gates cost very little runtime and communication price since the efficient parallel algorithms are applied. And theoretically, given the iteration round ℓ, the numbers of exchanging messages are $3\ell + 2$, $6\ell + 6$ and $6\ell + 3$ respectively for forget gate, input gate and output gate. Assume that the average length of the corresponding matrixes is n, we have the same complexity of communication overhead for the three gates as $\mathcal{O}(\ell \cdot n^2)$. What is also noteworthy is that most operations of the three gates in FP are irrelevant. Therefore, we can even further improve the efficiency of our framework by concurrently computing them.

TABLE 3
Runtime and message size of each LSTM gate for each training sample

Gates	Forward Propagation			
	RunTime Per Sample(ms)		Size Per Sample(MB)	
	Initialization	Iteration	Initialization	Iteration
Forget Gate	12.48	25.44	0.246	0.218
Input Gate	45.97	93.67	0.503	0.445
Output Gate	34.49	70.28	0.334	0.279

TABLE 4
Comparison of Runtime and message size with other schemes

	RunTime Per Sample(ms)	Message Size Per Sample(MB)
Our Scheme	395.18	2.435
Common Scheme [42]	17.23	0
GMM with HE [11]	1073.47	47.6
HMM with 2PC [10]	874.49	6.61

In Table 5, we further analyze the advantages of OPSR through comparison. It can be observed that the most differences between the common IoT speech recognition frameworks and the later three frameworks is the lack of security

TABLE 5
Comparison of Different Schemes

	Common Scheme [42]	GMM with HE [11]	HMM with 2PC [10]	Our Scheme
Security & Privacy	×	✓	✓	✓
Additivity	×	✓	✓	✓
Efficiency	✓	×	✓	✓
Support Deep Learning	✓	×	×	✓
Outsourced Server	×	✓	✓	✓

and privacy for the audio feature data. Meanwhile, the traditional framework cannot support the service provider to securely use untrusted third-party servers. Moreover, as mentioned before, due to the cost on computation price, the homomorphic encryption (HE) based frameworks is less efficient than the additive secret sharing based ones. As shown in Table 4, to process one frame of a speaker, the HE based privacy-preserving GMM scheme [11] needs 1073.47 milliseconds. And privacy-preserving HMM scheme proposed by Smaragdis *et al.* [11] needs almost two times runtime and computation overhead compared with OPSR. As for the common scheme, due to lack of privacy-preserving mechanism, it is much more efficient than the privacy-preserving schemes. However, as mentioned in Section 1, the risk it have to face with in applications is more. Moreover, compared with the traditional Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM), LSTM based models have got higher accuracy and wider application in recent years [43]. Therefore, while ensuring privacy, our framework outperforms previous work in accuracy and efficiency.

8.2 Performance of Privacy-Preserving LSTM Interactive Protocols

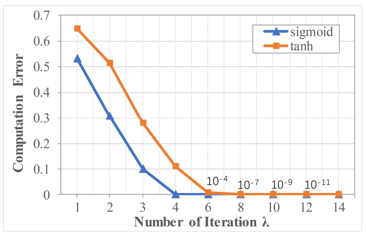

Fig. 9. The computation error of sigmoid and tanh function in difference iteration number λ

Since sigmoid function or tanh function are implemented by iteration method, the most time-consuming operation of each gate in secure LSTM is computing them. In addition, as mentioned before, tanh can be derived from sigmoid by local computation. Therefore, we mainly evaluate the performance of the $SSigmoid(\cdot)$ protocol. And the performance of $STanh(\cdot)$ is almost identical to it.

To compute $f(x) = e^x$ and $f(x) = \frac{1}{x}$ in the secure sigmoid protocol, we introduce iterative method $SecExp(\cdot)$ and $SecInv(\cdot)$ into our framework. And this inevitably causes the errors of the outputs. Fig. 9 gives the changes of errors when the iterative times differs. It can be seen that, when the number of iteration is 10, the error reaches 10^{-9} which is completely negligible in practical computation. Consequently, we set the default iterative number to 10 in the following experiments. Besides, the secure square root protocol $SecSqrt(\cdot)$ and the secure logarithm protocol $SecLog(\cdot)$ respectively utilize the same approximation methods as $SecInv(\cdot)$ and $SecExp(\cdot)$. Thus, their computation errors and convergence properties are also identical and not deliberately evaluated here.

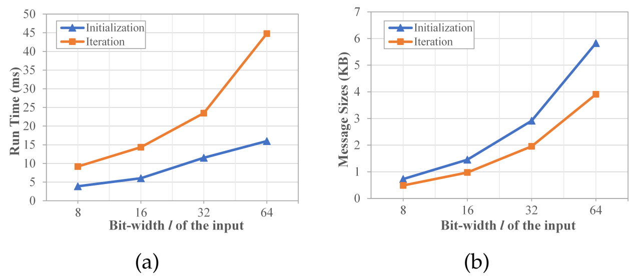

Fig. 10. Secure sigmoid function: (a) Runtime for different length ℓ of the ciphertext; (b) Message sizes between the two edge servers for different length ℓ of the ciphertext.

Due to the fact that the precision of data representation and the degree of security are strongly influenced by the ciphertext length ℓ, we experiment the changes of our $SSigmoid$ protocol runtime under different ℓ. Note that the runtime of initialization and iteration relate to not only the different phases of $SSigmoid$, but also the involved $SecMul$. So does the message size computation. As shown in Fig. 10 (a) and (b), both the runtime and the message sizes rise with the length of input. However, they are still in the degree of milliseconds and kilobytes. And this is totally tolerable in practical application, because usually, speech recognition for smart IoT devices allows a delay of several seconds. Specially, if the required degree of security is not very high, $\ell = 32$ is enough to protect the privacy. Thus, the following experiments are all performed under $\ell = 32$.

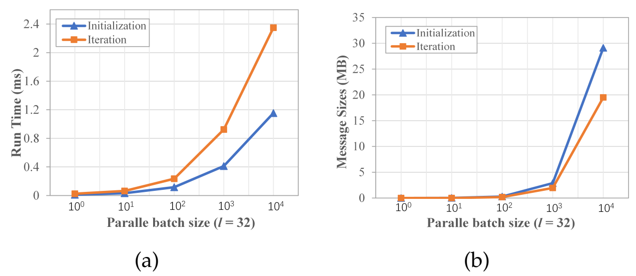

Fig. 11. Secure ssigmoid function: (a) Runtime for different parallel batch size; (b) Message sizes between the two edge servers for different parallel batch size.

In the neural networks, data is always represented as vectors or matrixes. It is too inefficient to process one element at a time. Therefore, with the help of NumPy, we further evaluate the performance of $SSigmoid$ in parallel. As illustrated in Fig.11 (a), although the runtime of $SSigmoid$ protocol goes up with the parallel batch size, it is far more slower. When the parallel batch size is expanded 10^4 times, the runtime only increases about 10^2 times. However, what can not be avoided is that the message size grows in proportion to the parallel batch size as shown in Fig. 11 (b).

9 Conclusion

In this paper, we study the secret sharing based method to protect the privacy of the long short term memory neural network. Based on the secret sharing based sub-protocols for linear and nonlinear operations, we first respectively design several interactive protocols for the each gate of LSTM. By randomly splitting the audio feature data into secret shares, all the data are efficiently processed under the status of ciphertext in these protocols. Thus, the privacy is preserved without the consciousness of all the participants. And compared with homomorphic encryption and garbled circuit based frameworks, OPSR dramatically reduces the computation and communication overhead for protecting privacy.

Appendix A
Security Proof of Theorem 1

A.1 Security Proof of SecInv

proof. In $SecInv$, given the number of iteration round τ, what S_1 holds is $View_1 = (u_1, r_1, \alpha_1, \mathcal{G}'_1, \mathcal{F}'_1)$, where $\mathcal{G}'_1 = \{\xi'_0, \xi'_1, ..., \xi'_\tau\}$, $\mathcal{F}'_1 = \{\varphi'_0, \varphi'_1, ..., \varphi'_{\tau-1}\}$, and $alpha_1 = u_1 + r_1$. ξ'_i and φ'_i are both the outputs of $SecMul$. According to *Lemma* 2 and *Lemma* 3, it is guaranteed that all the values of $View_1$ are uniformly random. Thus, all of them can be perfectly simulated by the simulator S_1, and are unable to distinguished by the adversary A in polynomial time. Similarly, S_2 can also hold $View_2$ which is simulatable and indistinguishable □

A.2 Security Proof of SecSqrt

proof. In $SecSqrt$, given the number of iteration round τ, what S_1 holds is $View_1 = (u_1, \mathcal{W}'_1, \mathcal{G}'_1, \mathcal{F}'_1)$, where $\mathcal{W}'_1 = \{\varsigma'_0, \varsigma'_1, ..., \varsigma'_\tau\}$, $\mathcal{G}'_1 = \{\xi'_0, \xi'_1, ..., \xi'_\tau\}$ and $F'_1 = \{\varphi'_0, \varphi'_1, ..., \varphi'_{\tau-1}\}$. ξ'_i and φ'_i are respectively the outputs of $SecMul$ and $SecAdd$. ς'_i is the output of $SecInv$. As proved before, $SecInv$ is secure. According to *Lemma* 2 and *Lemma* 3, it is guaranteed that all the values of $View_1$ are uniformly random. Thus, all of them can be perfectly simulated by the simulator S_1, and are unable to distinguished by the adversary A in polynomial time. Similarly, S_2 can also hold $View_2$ which is simulatable and indistinguishable. □

A.3 Security Proof of SecLog

proof. In $SecLog$, given the number of iteration round τ, what S_1 holds is $View_1 = (u_1, v_1, d_1, s_1, \mathcal{G}'_1, \mathcal{F}'_1, \epsilon_1)$, where $\mathcal{G}'_1 = \{\xi'_0, \xi'_1, ..., \xi'_\tau\}$, $\mathcal{F}'_1 = \{\varphi'_0, \varphi'_1, ..., \varphi'_{\tau-1}\}$. v_1, s_1 and ξ'_i are the outputs of $SecMul$. φ'_i is the outputs of $SecAdd$. d_1 is the output of $SecInv$. As proved before, $SecInv$ is secure. According to *Lemma* 2 and *Lemma* 3, it is guaranteed that all the values are uniformly random. Thus, all of them can be perfectly simulated by the simulator S_1, and are unable to distinguished by the adversary A in polynomial time. Similarly, S_2 can also hold $View_2$ which is simulatable and indistinguishable. □

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. U1804263, U1764263, 61872283, U1708262), the China 111 Project (No. B16037).

References

[1] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[2] J. Lin, J. Niu, and H. Li, "Pcd: A privacy-preserving predictive clinical decision scheme with e-health big data based on rnn," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2017, pp. 808–813.

[3] X. Yuan, Y. Chen, Z. Yue, Y. Long, X. Liu, C. Kai, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," 2018.

[4] N. Singh and R. Khan, "Underlying text independent speaker recognition," in *IEEE Conference (ID: 37465)(10th INDIACom 2016 International Conference on Computing for Sustainable Global Development), held on 16th-18th March*, 2016, pp. 11–15.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[6] R. Takashima, Y. Kawaguchi, Q. Sun, T. Sumiyoshi, and M. Togami, "An application of noise-robust speech translation using asynchronous smart devices," in *Asia-pacific Signal & Information Processing Association Summit & Conference*, 2018.

[7] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley, "A brief survey of machine learning methods and their sensor and iot applications," in *Information, Intelligence, Systems & Applications (IISA), 2017 8th International Conference on*. IEEE, 2017, pp. 1–8.

[8] T.-D. Nguyen, E.-N. Huh, and M. Jo, "Decentralized and revised content-centric networking-based service deployment and discovery platform in mobile edge computing for iot devices," *IEEE Internet of Things Journal*, 2018.

[9] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer Systems*, vol. 70, pp. 138–147, 2017.

[10] P. Smaragdis and M. V. Shashanka, "A framework for secure speech recognition," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4. IEEE, 2007, pp. IV–969.

[11] M. A. Pathak and B. Raj, "Privacy-preserving speaker verification and identification using gaussian mixture models," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 2, pp. 397–406, 2013.

[12] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *International Conference on Artificial Neural Networks*. Springer, 2007, pp. 220–229.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1137–1149, 2017.

[14] J. F. Kolen and S. C. Kremer, *A field guide to dynamical recurrent networks*. John Wiley & Sons, 2001.

[15] Z. Ma, Y. Liu, Z. Wang, H. Ge, and M. Zhao, "A machine learning-based scheme for the security analysis of authentication and key agreement protocols," *Neural Computing and Applications*, pp. 1–13, 2018.

[16] H. A. Bourlard and N. Morgan, "Connectionist speech recognition," *Springer International*, vol. 247, 1994.

[17] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5934–5938.

[18] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," *arXiv preprint arXiv:1412.6181*, 2014.

[19] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[20] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 619–631.

[21] C. Eryonucu, E. Ayday, and E. Zeydan, "A demonstration of privacy-preserving aggregate queries for optimal location selection," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2018, pp. 1–3.

[22] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning: Revisited and enhanced," in *International Conference on Applications and Techniques in Information Security*. Springer, 2017, pp. 100–110.

[23] P. Li, J. Li, Z. Huang, T. Li, C.-Z. Gao, S.-M. Yiu, and K. Chen, "Multi-key privacy-preserving deep learning in cloud computing," *Future Generation Computer Systems*, vol. 74, pp. 76–85, 2017.

[24] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[25] A. Saleem, A. Khan, F. Shahid, M. Alam, and M. K. Khan, "Recent advancements in garbled computing: How far have we come towards achieving secure, efficient and reusable garbled circuits," *Journal of Network and Computer Applications*, 2018.

[26] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[27] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[28] K. Mivule, C. Turner, and S.-Y. Ji, "Towards a differential privacy and utility preserving machine learning classifier," *Procedia Computer Science*, vol. 12, pp. 176–181, 2012.

[29] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[30] Z. Ma, Y. Liu, X. Liu, J. Ma, and K. Ren, "Lightweight privacy-preserving ensemble classification for face recognition," *IEEE Internet of Things Journal*, 2019.

[31] D. Hofheinz and V. Shoup, "Gnuc: A new universal composability framework," *Journal of Cryptology*, vol. 28, no. 3, pp. 423–508, 2015.

[32] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[33] Y. Lindell and B. Riva, "Blazing fast 2pc in the offline/online setting with security for malicious adversaries," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 579–590.

[34] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[35] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography Conference*. Springer, 2006, pp. 285–304.

[36] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[37] I. Damgård, C. Orlandi, and M. Simkin, "Yet another compiler for active security or: efficient mpc over arbitrary rings," in *Annual International Cryptology Conference*. Springer, 2018, pp. 799–829.

[38] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on Nuclear Science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[39] E. Ogasawara, L. C. Martinez, D. De Oliveira, G. Zimbrão, G. L. Pappa, and M. Mattoso, "Adaptive normalization: A novel data normalization approach for non-stationary time series," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[40] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[41] K. Jia, M. Kenney, J. Mattila, and T. Seppala, "The application of artificial intelligence at chinese digital

platform giants: Baidu, alibaba and tencent," 2018.

[42] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4960–4964.

[43] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schlüter, and H. Ney, "A comprehensive study of deep bidirectional lstm rnns for acoustic modeling in speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2462–2466.

Zhuo Ma received the his Ph.D. degree in Computer Architecture from Xidian University, Xian, China, in 2010. Now, he is an associate professor at school of Cyber Engineering, Xidian University. His research interests include cryptography, machine learning in cyber security, and Internet of things security.

Yang Liu received his B.S. degree in computer science and technology from Xidian University in 2017. He is currently a master student in the School of Cyber Engineering, Xidian University. His research interests cover formal analysis of protocols and deep learning neural network

Ximeng Liu (S'13-M'16)received the B.Sc. degree in electronic engineering from Xidian University, Xi'an, China, in 2010 and Ph.D. degrees in Cryptography from Xidian University, China, in 2015. Now, he is a full professor at College of Mathematics and Computer Science, Fuzhou University, China. Also, he is a research fellow at School of Information System, Singapore Management University, Singapore. He has published over 100 research articles include IEEE TIFS, IEEE TDSC, IEEE TC, IEEE TII IEEE TSC and IEEE TCC. His research interests include cloud security, applied cryptographyand big data security.

Jianfeng Ma received the Ph.D degree from Xidian University, Xian, China, in 1995. He has been a professor in Department of Computer Science and Technology, Xidian University since 1998. He was the special engaged professor of the Yangtze River scholar in China. His research interests include cryptology, network security, data security and so on.

Feifei Li received the BS degree in computer engineering from the Nanyang Technological University in 2002 and the PhD degree in computer science from the Boston University in 2007.He is currently an associate professor in the School of Computing, University of Utah. His research interests include database and data management systems and big data analytics.