SecureNLP: A System for Multi-Party Privacy-Preserving Natural Language Processing

Qi Feng, Debiao He, *Member, IEEE*, Zhe Liu, *Senior Member, IEEE*, Huaqun Wang, and Kim-Kwang Raymond Choo, *Senior Member, IEEE*

***Abstract*— Natural language processing (NLP) allows a computer program to understand human language as it is spoken, and has been increasingly deployed in a growing number of applications, such as machine translation, sentiment analysis, and electronic voice assistant. While information obtained from different sources can enhance the accuracy of NLP models, there are also privacy implications in the collection of such massive data. Thus, in this paper, we design a privacy-preserving system SecureNLP, focusing on the instance of recurrent neural network (RNN)-based sequence-to-sequence with attention model for neural machine translation. Specifically, for non-linear functions such as sigmoid and tanh, we design two efficient distributed protocols using secure multi-party computation (MPC), which are used to carry out the respective tasks in the SecureNLP. We also prove the security of these two protocols (i.e., privacy-preserving long short-term memory network PrivLSTM, and privacy-preserving sequence to sequence transformation PrivSEQ2SEQ) in the semi-honest adversary model, in the sense that any honest-but-curious adversary cannot learn anything else from the messages they receive from other parties. The proposed system is implemented in C++ and Python, and the findings from the evaluation demonstrate the utility of the protocols in cross-domain NLP.**

***Index Terms*— Secure multi-party computation, natural language processing, seq2seq with attention, long short-term memory.**

Manuscript received March 15, 2020; revised May 4, 2020; accepted May 19, 2020. Date of publication May 25, 2020; date of current version July 9, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC1604000, and in part by the National Natural Science Foundation of China under Grant 61932016, Grant 61972294, Grant 61802180, Grant 61941116, and Grant 61872192, in part by the Natural Science Foundation of Jiangsu Province under Grant BK20180421, in part by the National Cryptography Development Fund under Grant MMJJ20180105, in part by the Fundamental Research Funds for the Central Universities under Grant NE2018106, and in part by the Major Science Research Project of Jiangsu Provincial Education Department under Grant 19KJA310010. The work of Kim-Kwang Raymond Choo was supported in part by the Cloud Technology Endowed Professorship and National Science Foundation CREST under Grant HRD-1736209. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Frederik Armknecht. *(Corresponding author: Debiao He.)*

Qi Feng and Debiao He are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, and also with the State Key Laboratory of Cryptology, Beijing 100878, China (e-mail: fengqi.whu@whu.edn.cn; hedebiao@163.com).

Zhe Liu is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China (e-mail: sduliuzhe@gmail.com).

Huaqun Wang is with the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China (e-mail: wanghuaqun@aliyun.com).

Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: raymond.choo@fulbrightmail.org).

Digital Object Identifier 10.1109/TIFS.2020.2997134

I. Introduction

THE renewed interest in natural language processing (NLP) is partly due to recent trends in artificial intelligence (AI), and the potential of NLP in applications involving textual data. Generally, as shown in Fig. 1, NLP tools and techniques can facilitate the processing, analysis, and interpretation of significant volume of data, and in turn informs decision making and strategy formulation. For example, Tractica [1] estimated that the NLP market (including the NLP hardware, software, and services) is likely to worth $22.3 billion by the year 2025, and NLP software profits increase from $136 million in 2016 to $5.4 billion by the year 2025.

There are, however security and privacy concerns associated with NLP-enabled data analytics applications [2]–[5]. For example, as more data are collected from both our physical and cyber environments, there is a need to ensure private information about the users and their environments is securely communicated and stored, as well as the analysis of such data does not infringe on user privacy. The importance of data security and privacy is partly evidenced by the introduction of exacting privacy regulations, such as the General Data Protection Regulation (EU) 2016/679 (GDPR), whose clauses formally regulate the usage specification on user data, particularly relating to data security and privacy. Hence, in this paper we seek to preserve user privacy without affecting data utility in NLP services.

To achieve privacy preservation in NLP, a common approach is to utilize homomorphic encryption (HE), which allows (encrypted) data to undergo certain arithmetic operations (e.g, addition and multiplication) without the need to decrypt or reveal the underlying data. Hence in recent years, several HE-based building blocks have been proposed. Examples include SecureLR designed by Jiang *et al.* [6] and the computation toolkits designed in [7], [8].

Another approach to achieve privacy preservation is to use multi-party computation (MPC), where computations are performed on the secret inputs from various parties (such as several service providers, users and so on). The parties are not able to learn any information related with others' inputs, with the exception of what can be learned from the output. The output can be made available to all involved parties. As for the NLP tasks, for example, MPC can be leveraged when keywords from one article do not have sufficient scope points to perform tagging using NLP, but a combination of data from several filesets will make the NLP works well.

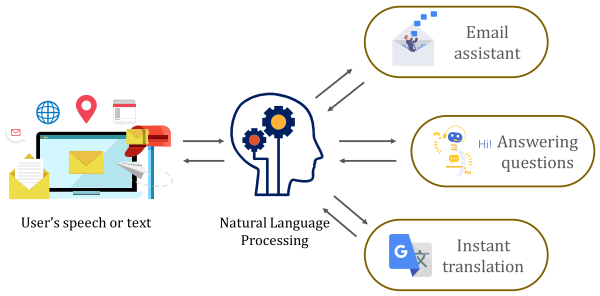

Fig. 1. NLP workflow.

The exchange of plaintext would clearly be not advisable, and using MPC allows us to execute the required computations on privacy-protected data.

There are several types of MPC protocols, such as those based on garbled circuit (GC) [9], [10] and secret sharing [11], [12]. Mohassel and Zhang [13] applied GC to compare two elements and designed SecureML. The latter is a framework that supports privacy-preserving two-party linear regression, logistic regression and natural network training. An inherent limitation of GC-based approaches is the time cost incurred during the generation of a circuit, which limits its suitability for complex NLP tasks. Secret sharing-based MPC protocols can execute the computation on arithmetic circuit symmetrically among multiple parties with high performance, and hence in recent years, a number of secret sharing-based privacy-preserving protocols for machine learning have been proposed in the literature [14]–[22].

We note that most of the previous researches are not designed for natural language processing system. Although linear operations of secret sharing-based toolkits are almost for free, computation of non-linear functions (e.g., sigmoid and tanh) used in NLP tasks frequently are difficult. Our strategy here is totally innovative. Starting from the basis building blocks, i.e., sigmoid $\sigma(u) = \frac{e^u}{e^u-1}$ and tanh $\tau(u) = \frac{e^{2u}-1}{e^{2u}+1}$, our most fundamental functionality is to mask the numerator and denominator respectively with a common multiplicative shared random value and provide conversion between multiplicative and additive secret sharing. The remarkable point is that by using pre-computation technology, this conversion allows parties to jointly calculate the non-linear functions more quicker than GC-based computation.

A. Our Contributions

In this paper, we focus on the recurrent neural network (RNN)-based sequence to sequence (seq2seq) with attention model and design a new distributed framework to protect the privacy of all parties during the NLP tasks. A summary of our contributions is provided below.

We design a series of multi-party interactive protocols for non-linear activation functions, i.e., sigmoid and tanh, by adopting both multiplicative and additive secret sharing. Specifically, we set the secret value to be additive shared such that the activation result can be multiplicative shared (because the secret value is a power of public e). Then, we use a private multi-party multiplication protocol similar to that of [23] to mask the numerator and denominator with the same multiplicative-shared random, respectively, and convert these multiplicative shares to additive shares.

On the basis of above building blocks, we propose two privacy-preserving multi-party protocols for long short-term memory network and RNN-based seq2seq with attention model. Specifically, we consider a set of parties $P_1, \ldots, P_n$, each element of the input sequence $\{x_1, \ldots, x_T\}$ is additively shared among parties and the prediction results $\{y_1, \ldots, y_{T'}\}$ are publicly learned but nothing else.

As a result, we obtain a multi-party privacy-preserving natural language processing SecureNLP that is both simple and efficient. We will also show later in the paper that our protocol is secure and practical. For example, for a two-party seq2seq with attention model with 64 hidden units and in an experimental environment, one seq2seq prediction has a total runtime of 7.08s. Furthermore, it can easily be parallelized to further speed-up the executing time.

B. Organization of the Paper

The remaining of the paper is organized as follows. In Section II, we briefly review the relevant background materials. The system model, building blocks and designed SecureNLP system are respectively described in Sections III, IV, and V. In Sections VI and VII, we present the formal proof and performance evaluation of the protocol, respectively. Finally, we conclude this paper in Section VIII.

II. Preliminaries

In this section, we will first explain the notations used in this paper, a background about NLP system (e.g., LSTM network, RNN-based seq2seq model with attention), and then some basis materials of secret sharing-based MPC (e.g., representation of secret values and private multiplication).

We denote n as the number of parties, and $\mathbb{N}$ as the set of $\{1, \ldots, n\}$. Each secret value x is a number in rational field $\mathscr{F}$, $\mathbf{x} \in \mathscr{F}^m$ in bold symbols refer to a vector of m length, and $\mathbf{M} \in \mathscr{F}^{m \times k}$ in boldface upper case is a matrix with size $m \times k$. For NLP tasks, we denote m as the batch size, T as the length of input sequence, and T' as the length of output sequence. Two running indexes denoting the iteration over T and T' (resp.) are $t = 1, \ldots, T$ and $t' = 1, \ldots, T'$.

A. LSTM Network

To avoid the vanishing gradient problem of simple RNN networks, we choose the long short-term memory (LSTM) network as the basis of seq2seq with attention model. A LSTM network makes a transformation from an input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathscr{F}^{m \times T}$ to another sequence $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T) \in \mathscr{F}^{m \times T'}$ by a series of layers. Each layer consists of some hidden cells (basic units of LSTM) and has a hidden state recording parts of history memory. Each layer comprises three gates, and the output of each layer is connected to its previous layer's hidden state and four sources of inputs, namely: $\boldsymbol{net}^c$ is the input for hidden state, $\boldsymbol{net}^{in}$, $\boldsymbol{net}^f$, $\boldsymbol{net}^{out}$ are inputs to the input gate, forget gate, and output gate, respectively.

We denote the iterations by $t = 1, 2, \ldots, T$ (only one layer works in a single iteration). Throughout this paper, σ and τ denote two non-linear activation functions: sigmoid function $\sigma(u) = \frac{e^u}{e^u - 1}$ and tanh function $\tau(u) = \frac{e^{2u}-1}{e^{2u}+1}$.

For each iteration, we can mathematically express each layer's three gates as follows.

1) *Forget gate*: this first step decides what information that is going to be forgotten from the hidden state. It examines the inputs of the previous layer's hidden state $\boldsymbol{h}_{t-1}$, and the data $\boldsymbol{x}_t$ offered in this iteration. The output $\boldsymbol{f}_t$ is defined as:

$$\begin{aligned} \boldsymbol{net}_t^f &= \mathbf{W}_f \cdot \boldsymbol{x}_t + \mathbf{U}_f \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_f \\ \boldsymbol{f}_t &= \sigma(\boldsymbol{net}_t^f) \end{aligned}$$

In the above equation, $\mathbf{W}_f, \mathbf{U}_f$ denote the weight matrices of the gate that decide how many inputs and hidden states to be kept, and $\boldsymbol{b}_f$ is the bias vector. The sigmoid function makes the final decision, a ratio in the range [0, 1] to represent the proportion between "get rid of these information" and "keep these information".

2) *Input gate*: this step decides how much new information is to be stored in the hidden state. It comprises two parts. The first part controls the input data $\boldsymbol{x}_t$, and the other part creates a new candidate $\tilde{\boldsymbol{c}}_t$ that would be added as part of the memory for this layer's hidden state. The output $\boldsymbol{g}_t$ is defined as:

$$\begin{aligned} \boldsymbol{net}_t^{in} &= \mathbf{W}_{in} \cdot \boldsymbol{x}_t + \mathbf{U}_{in} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_{in} \\ \boldsymbol{g}_t &= \sigma(\boldsymbol{net}_t^{in}) \\ \boldsymbol{net}_t^{c} &= \mathbf{W}_{c} \cdot \boldsymbol{x}_t + \mathbf{U}_{c} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_{c} \\ \tilde{\boldsymbol{c}}_t &= \tau(\boldsymbol{net}_t^{c}) \end{aligned}$$

In the above equation, $\mathbf{W}_{in}, \mathbf{U}_{in}, \mathbf{W}_c, \mathbf{U}_c$ are the weight matrices of this gate, and $\boldsymbol{b}_{in}, \boldsymbol{b}_c$ denote the bias vectors. Similarly, the final decisions are made via activation functions σ and τ.

3) *State update*: this step updates parts of the old hidden state, i.e., $\boldsymbol{c}_{t-1}$, into a new one, i.e., $\boldsymbol{c}_t$. Specifically, the memory cells forget old information and remember some new messages based on the decisions of the forget and input gates as follows:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{g}_t \circ \tilde{\boldsymbol{c}}_t.$$

In the above equation, the operation $\circ$ points to the element-wise product of two vectors.

4) *Output gate*: this step decides which parts of the layer's state are going to be presented as the output. The decision is based on the current input data and the states of both this layer and previous layer. Mathematically, the output of this layer is defined as:

$$\begin{aligned} \boldsymbol{net}_t^{out} &= \mathbf{W}_o \cdot \boldsymbol{x}_t + \mathbf{U}_o \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_o \\ \boldsymbol{h}_t &= \sigma(\boldsymbol{net}_t^{out}) \circ \tau(\boldsymbol{c}_t) \end{aligned}$$

Similarly, $\mathbf{W}_o, \mathbf{U}_o$ denote the weight matrices of this gate to define how much the current input data and history information could affect the output, and $\boldsymbol{b}_o$ is the bias vector. The final output $\boldsymbol{y}_t$ is decided by $\boldsymbol{h}_t$, for example, we can directly assign the output $\boldsymbol{y}_t := \boldsymbol{h}_t$ for each $t \in \{1, \ldots, T\}$.

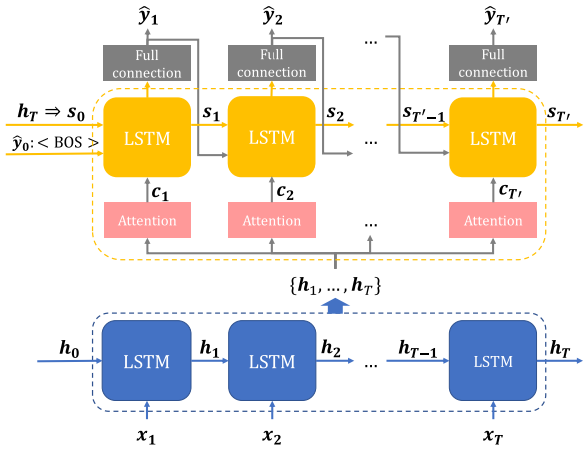

Fig. 2. The paradigm of RNN-based seq2seq model with attention.

B. RNN-Based Seq2seq Model With Attention

Seq2seq model [24] and encoder-decoder [25] are introduced to transform one sequence to another one with arbitrary length. For example, in a machine translation task, the length of English sentences generally cannot be predefined and the corresponding translation in French is also unfixed (and differs in length from the English words). Based on the description in [24], we will briefly describe the RNN-based seq2seq with attention model here. As shown in Fig. 2, a seq2seq with attention model includes two RNN networks (instantiated by the LSTM), named encoder and decoder respectively. The encoder is responsible for the analysis on input sequence while the decoder outputs the transforming result.

The workflow of RNN-based seq2seq model with attention model is mathematically described as follows.

1) Encoder: When given the input sequence of $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\} \in \mathscr{F}^{m \times T}$, the encoder encodes it into the hidden states $\boldsymbol{h} = \{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T\} \in \mathscr{F}^{m \times T}$ via the first multi-layer LSTM network, i.e. for each iteration $t \in \{1, \ldots, T\}$:

$$\boldsymbol{h}_t \leftarrow \mathsf{LSTM}_1(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

Then, the encoder converts the hidden states into the context vectors using the following equation:

$$\boldsymbol{c} = q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T)$$

where $q(\cdot)$ is an customized model function. In this paper, the context vectors are updated during the decoding process by the attention model (the details are described later in this section).

Now, we can see that the hidden states (or context vectors) have loaded the input sequence's information.

2) Decoder With Attention: Based on the encoder's hidden states $\boldsymbol{h} = \{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T\} \in \mathscr{F}^{m \times T}$, the decoder extracts the output sequence $\hat{\boldsymbol{y}} = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{T'}\} \in \mathscr{F}^{m \times T'}$. Similarly for the encoder, in each iteration, the decoder predicts the next symptom $\boldsymbol{y}_{t'}$ with the given context vector $\boldsymbol{c}_{t'}$ and all the

previously predicted symptoms $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t'-1}\}$. The attention mechanism tells the decoder which symptom or encoder's hidden state should have a higher emphasis in this iteration.

The decoder is applied by *another* LSTM network. First, it is initialized by

- the same number of hidden units as that of encoder;
- the encoder's output from all iterations (used as the attention's memory);
- $\boldsymbol{s}_0 = \boldsymbol{h}_T$ where $\boldsymbol{h}_T$ is the hidden state of the encoder's last iteration;
- $\boldsymbol{y}_0$, assigned by a basic identifier, e.g., $\langle$BOS$\rangle$ in the machine translation task who identifies the beginning of a sequence.

Then, for each iteration $t' \in \{1, \ldots, T'\}$, the decoder puts the previous output $\boldsymbol{y}_{t'-1}$, the previous hidden state $\boldsymbol{s}_{t'-1}$ and the current context vector $\boldsymbol{c}_{t'}$ into the LSTM network, i.e.,

$$\begin{aligned} \boldsymbol{s}_{t'} &\leftarrow \mathsf{LSTM}_2(\boldsymbol{y}_{t'-1}, \boldsymbol{c}_{t'}, \boldsymbol{s}_{t'-1}) \\ \boldsymbol{y}_{t'} &:= \boldsymbol{s}_{t'} \end{aligned}$$

$\boldsymbol{c}_{t'}$ is the input sequence's context vector defined by the following equations:

$$\begin{aligned} e_{t't} &= a(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_t) = \boldsymbol{s}_{t'-1}^T \cdot \boldsymbol{h}_t \\ \alpha_{t't} &= \frac{\exp(e_{t't})}{\sum_{k=1}^{T} \exp(e_{t'k})} \\ \boldsymbol{c}_{t'} &= \sum_{t=1}^{T} \alpha_{t't} \cdot \boldsymbol{h}_t \end{aligned}$$

In the above equations, $\boldsymbol{h}_t$ is encoder's hidden state at iteration $t \in \{1, \ldots, T\}$, and $\boldsymbol{s}_{t'-1}$ is decoder's hidden state at iteration $t' \in \{1, \ldots, T'\}$. At the end of this iteration, the decoder gets the final output $\hat{\boldsymbol{y}}_{t'} = \mathbf{W}_y \cdot \boldsymbol{y}_{t'} + \boldsymbol{b}_y$, where $\mathbf{W}_y, \boldsymbol{b}_y$ are the weight matrix and bias vector for this full connection projection. Finally, the decoder outputs the sequence $\{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{T'}\}$ as the NLP task's result. It may be the joint probability for the translation sentence in a machine translation mission.

C. Representation of Values

For each secret shared value $u \in \mathscr{F}$, $\langle u \rangle_i$ denotes the *additive* share for party P_i, and $[u]_i$ denotes the *multiplicative* share for P_i, $i \in \mathbb{N}$. To distinguish from the operations on the general values, $+$ *denotes the addition operation among shares and* $\times$ *is the multiplication operation*. Therefore, we have

$$u = \langle u \rangle_1 + \cdots + \langle u \rangle_n \quad \text{and } u = [u]_1 \times \cdots \times [u]_n$$

Using the natural linear feature of $\langle u \rangle_i$ and $[u]_i$, for secret values u, v and public constant c we have the following:

$$\begin{aligned} \langle u \rangle_i \pm \langle v \rangle_i &= \langle u \pm v \rangle_i, \quad c \times \langle u \rangle_i = \langle c \times u \rangle_i \\ [u]_i \times [v]_i &= [u \cdot v]_i, \quad c^{\langle u \rangle_i} = [c^u]_i \end{aligned}$$

To reconstruct the secret value u, each party P_i sends $\langle u \rangle_i$ (or $[u]_i$) to all the other parties P_j $(j \neq i)$, and all of them calculate $u = \sum_{\ell=1}^{n} \langle u \rangle_\ell$ (or $u = \prod_{\ell=1}^{n} [u]_\ell$). This operation is denoted as $\mathsf{Rec}(\langle u \rangle_1, \ldots, \langle u \rangle_n)$ (or $\mathsf{Rec}([u]_1, \ldots, [u]_n)$).

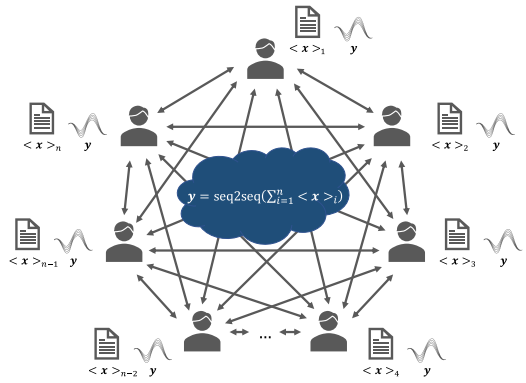

Fig. 3. The system architecture of SecureNLP.

D. Private Multiplication

Our SecureNLP system utilizes our two following subprotocols π_{mul}^{2P} and π_{mul}^{nP} that compute the product of additive and multiplicative shares, respectively, with privacy. Specifically, the multiplication protocols are defined as follows:

- π_{mul}^{2P}: suppose that there are two parties P_A and P_B with two couple of additive shares $\{\langle u \rangle_A, \langle v \rangle_A\}, \{\langle u \rangle_B, \langle v \rangle_B\}$ related to two secret values u and v, the two parties run the π_{mul}^{2P} protocol to calculate $\langle z \rangle_A, \langle z \rangle_B$ such that $\langle z \rangle_A + \langle z \rangle_B = (\langle u \rangle_A + \langle u \rangle_B) \times (\langle v \rangle_A + \langle v \rangle_B)$, each of the parties just learns a single share.
- π_{mul}^{nP}: suppose that there are n parties $P_1, \ldots, P_n$, each of them keeps one of the multiplicative shares $\{[u]_1, \ldots, [u]_n\}$ of the secret value u, the parties run the π_{mul}^{nP} protocol to calculate the additive shares $\langle z \rangle_1, \ldots, \langle z \rangle_n$ such that $\sum_{\ell=1}^{n} \langle z \rangle_\ell = \prod_{\ell=1}^{n} [u]_\ell$. Similarly, each of the parties just learns one single share.

We assume that both protocols are private in the presence of semi-honest adversary. Lindell and Nof [26] suggested two instantiations for π_{mul}^{2P}: one is based on oblivious transfer [27] and has low computation cost but higher bandwidth, while the second solution is based on Paillier encryption [28] and has higher computation cost but much lower bandwidth. Both of them are proven secure under malicious adversary model. Huang *et al.* [29] applied Beaver's triple [30] for semi-honest private multiplication (originally presented in [11], [12]) that has low computation cost and bandwidth. Thus, we use a similar idea as that of Huang *et al.* [29]. As for the multi-party multiplication π_{mul}^{nP}, Doerner *et al.* [23] presented a oblivious transfer-based approach that is secure in the presence of malicious adversary. We simplify the protocol of Doerner *et al.* [23] to a semi-honest one, thus allowing us to achieve higher efficiency.

III. System Architecture

As illustrated in Fig. 3, our designed framework SecureNLP involves multiple parties $P_1, \ldots, P_n$. The trained RNN-based seq2seq with attention model is publicly available and can be obtained by all parties. For the secret values, our solution is based on secret sharing, where every party collects one share of the message and the NLP task is processed on

the summation message. Let $\langle \boldsymbol{x} \rangle_i$, $i \in \mathbb{N}$ denotes party P_i's knowledge and $\boldsymbol{y}$ denotes the processed result. To protect the privacy of parties, the computation will be performed jointly by all parties. After running the seq2seq transformation on the additively shared inputs, all $P_1, \ldots, P_n$ will learn the output $\boldsymbol{y}$ for various NLP tasks, such as natural machine translation, document understanding and question answering.

A. Security Model

Following the approach in [31], the SecureNLP system will be proven in the semi-honest (also known as honest-but-curious) security model. That is to say, the semi-honest adversary is assumed to follow specification of the protocols exactly but attempt to learn something more than expected during the multi-party interactions. The formal definition of the semi-honest security model is that for a protocol π (corresponding to the functionality f), we assume that there is a simulation $\mathcal{S}$ who can build a simulated world where the views (including all the intermediate values and outputs) of adversary $\mathcal{A}$ (controlling the corrupted parties $I \subseteq \mathbb{N}$) are computationally indistinguishable with the views in the real world:

$$\{view^{\pi}_{\mathcal{A}}(\kappa, \langle \boldsymbol{x} \rangle_1, \ldots, \langle \boldsymbol{x} \rangle_n), output^{\pi}(\boldsymbol{y})\} \overset{c}{\approx} Sim(\kappa, \{\langle \boldsymbol{x} \rangle_i\}_{i \in I}, f(\langle \boldsymbol{x} \rangle_1, \ldots, \langle \boldsymbol{x} \rangle_n))$$

B. Design Goals

The design goals of SecureNLP are as follows:

- **Correctness**. NLP tasks are correctly performed over multi-source data, and honest parties only learn the valid predicted output sequence.
- **Privacy protection**. To preserve the privacy of multi-source data, any parties or attackers will not be able to learn a party's secret knowledge during the process of NLP prediction.
- **Highly-efficiency**. The more efficient the SecureNLP is, the wider range of applications it could be utilized for. The designed protocols must be computationally efficient so that it can be deployed in resource-constrained environment.

IV. Building Blocks

In this section, two sub-protocols, i.e., PrivSigm and PrivTanh are designed to support the non-linear sigmoid and tanh functions, which are vital for RNN-based seq2seq with the attention model. Both sub-protocols are assumed to be executed in the multi-party setting, where each party's inputs are his/her additive shares and each party only learns additively shares of the functions' results.

A. Private Sigmoid

The sigmoid function is mathematically defined as $\sigma(u) = \frac{e^u}{e^u - 1}$ for some secret value $u \in \mathscr{F}$. We enhance its privacy with the setting of multi-party execution, denoted as PrivSigm. Before engaging in the protocol, the input u has been additively shared among $P_1, \ldots, P_n$, i.e., $\langle u \rangle_1, \ldots, \langle u \rangle_n \in \mathscr{F}$. To keep the secret shares secure, each party P_i will mask the exponent of $\langle u \rangle_i$ (i.e, $e^{\langle u \rangle_i}$) by multiplying it with a random number $[\rho]_i \in \mathscr{F}$. Based on the linear feature of multiplicative shares, we have $e^{\langle u \rangle_i} \times [\rho]_i = [e^u \cdot \rho]_i$. Then, the parties convert their multiplicative shares into additive shares by invoking π^{nP}_{mul} sub-protocol, i.e., $\{\langle \phi \rangle_1, \ldots, \langle \phi \rangle_n\} \leftarrow \pi^{nP}_{\text{mul}}([e^u \cdot \rho]_1, \ldots, [e^u \cdot \rho]_n)$ such that $\sum_{\ell=1}^{n} \langle \phi \rangle_\ell = \phi = \prod_{\ell=1}^{n} [e^u \cdot \rho]_\ell = e^u \cdot \rho$. To handle the part of minus-one operation, the parties need another private multi-party multiplication protocol to calculate $\{\langle \xi \rangle_1, \ldots, \langle \xi \rangle_n\} \leftarrow \pi^{nP}_{\text{mul}}([\rho]_1, \ldots, [\rho]_n)$, such that $\sum_{\ell=1}^{n} \langle \xi \rangle_\ell = \xi = \prod_{\ell=1}^{n} [\rho]_\ell = \rho$. Interactively, they run $\mathsf{Rec}(\langle \phi \rangle_1 - \langle \xi \rangle_1, \ldots, \langle \phi \rangle_n - \langle \xi \rangle_n)$ to learn $\chi = \phi - \xi$, and locally compute $\langle v \rangle_i = \frac{\langle \phi \rangle_i}{\chi}$. It can be derived that, after finishing the computations, each of the parties has one additive share of the result $v = \sigma(u)$ because

$$\sum_{\ell=1}^{n} \langle v \rangle_i = \frac{\sum_{\ell=1}^{n} \langle \phi \rangle_i}{\chi} = \frac{\phi}{\phi - \xi} = \frac{e^u \cdot \rho}{e^u \cdot \rho - \rho} = \frac{e^u}{e^u - 1}$$

The protocol is described in Algorithm 1.

Algorithm 1 The Online Phase of Privacy Preserving sigmoid Function PrivSigm

Input: Each P_i inputs parameter $\langle u \rangle_i$
Output: Each P_i learns its share $\langle v \rangle_i$

1: Each P_i, $i \in \mathbb{N}$, generates a random number $[\rho]_i$. Then parties run π^{nP}_{mul} with the inputs of $[\rho]_1, \ldots, [\rho]_n$ to obtain $\langle \xi \rangle_1, \ldots, \langle \xi \rangle_n$.
2: P_i locally calculates $[k]_i = e^{\langle u \rangle_i} \times [\rho]_i$ and jointly invoke π^{nP}_{mul} again with the inputs of $[k]_1, \ldots, [k]_n$ to obtain $\langle \phi \rangle_1, \ldots, \langle \phi \rangle_n$.
3: P_i computes $\langle \chi \rangle_i = \langle \phi \rangle_i - \langle \xi \rangle_i$. Then parties run $\mathsf{Rec}(\langle \chi \rangle_1, \ldots, \langle \chi \rangle_n)$ to know χ (that is equal to $(\prod_{\ell=1}^{n} [k]_\ell) - (\prod_{\ell=1}^{n} [\rho]_\ell) = (e^{\sum_{\ell=1}^{n} \langle u \rangle_\ell}) \times (\prod_{\ell=1}^{n} [\rho]_\ell) - (\prod_{\ell=1}^{n} [\rho]_\ell) = (e^{\sum_{\ell=1}^{n} \langle u \rangle_\ell} - 1) \times (\prod_{\ell=1}^{n} [\rho]_\ell)$).
4: P_i outputs $\langle v \rangle_i = \frac{\langle \phi \rangle_i}{\chi}$ locally with the same precision.

B. Private Tanh

The tanh function is mathematically defined as $\tau(u) = \frac{e^{2u} - 1}{e^{2u} + 1}$ for some secret value $u \in \mathscr{F}$. We enhance its security via the secret sharing mechanism. Similarly to the idea of PrivSigm, u is additively shared among $P_1, \ldots, P_n$, and throughout the protocol, every P_i will protect its share $\langle u \rangle_i$ by masking the secret intermediate $e^{2 \times \langle u \rangle_i}$ with an independently random number $[\rho]_i \in \mathscr{F}$ such that $e^{2 \times \langle u \rangle_i} \times [\rho_i] = [e^{2u} \cdot \rho]_i$. Then, the parties execute two π^{nP}_{mul} protocols to learn

$$\{\langle \phi \rangle_1, \ldots, \langle \phi \rangle_n\} \leftarrow \pi^{nP}_{\text{mul}}(e^{2 \times \langle u \rangle_1} \times [\rho]_1, \ldots, e^{2 \times \langle u \rangle_n} \times [\rho]_n)$$
$$\{\langle \xi \rangle_1, \ldots, \langle \xi \rangle_n\} \leftarrow \pi^{nP}_{\text{mul}}([\rho]_1, \ldots, [\rho]_n),$$

where $\sum_{\ell=1}^{n} \langle \phi \rangle_\ell = \phi = \prod_{\ell=1}^{n} e^{2 \times \langle u \rangle_\ell} \times [\rho]_\ell = e^{2u} \cdot \rho$ and $\sum_{\ell=1}^{n} \langle \xi \rangle_\ell = \xi = \prod_{\ell=1}^{n} [\rho]_\ell = \rho$. Now, the parties can calculate the denominator by running

$$\chi \leftarrow \mathsf{Rec}(\langle \phi \rangle_1 + \langle \xi \rangle_1, \ldots, \langle \phi \rangle_n + \langle \xi \rangle_n)$$

The additive share of result $v = \tau(u)$ for each party will be $\langle v\rangle_i = \frac{\langle\phi\rangle_i - \langle\xi\rangle_i}{\chi}$, which is based on the follow equations:

$$\sum_{\ell=1}^{n}\langle v\rangle_i = \frac{\sum_{\ell=1}^{n}\langle\phi\rangle_i - \langle\xi\rangle_i}{\chi} = \frac{\phi-\xi}{\phi+\xi} = \frac{e^{2u}\cdot\rho-\rho}{e^{2u}\cdot\rho+\rho} = \frac{e^{2u}-1}{e^{2u}+1}$$

The designed PrivTanh is shown in Algorithm 2.

Algorithm 2 The Online Phase of Privacy Preserving tanh Function PrivTanh

Input: Each P_i inputs parameter $\langle u\rangle_i$

Output: Each P_i learns its share $\langle v\rangle_i$

1: P_i generates a random number $[\rho]_i$ for $i \in \mathbb{N}$. Then parties run π_{mul}^{nP} with the inputs of $[\rho]_1, \ldots, [\rho]_n$ to obtain $\langle\xi\rangle_1, \ldots, \langle\xi\rangle_n$.

2: P_i calculates $[k]_i = e^{2\times\langle u\rangle_i} \times [\rho]_i$ for $i \in \mathbb{N}$ and then together run π_{mul}^{nP} with the inputs of $[k]_1, \ldots, [k]_n$ to obtain $\langle\phi\rangle_1, \ldots, \langle\phi\rangle_n$.

3: P_i calculates $\langle\chi\rangle_i = \langle\phi\rangle_i + \langle\xi\rangle_i$ for $i \in \mathbb{N}$. Then parties run $\mathsf{Rec}(\langle\chi\rangle_1, \ldots, \langle\chi\rangle_n)$ to obtain χ (that is equal to $(\prod_{\ell=1}^{n}[k]_\ell) + (\prod_{\ell=1}^{n}[\rho]_\ell) = (e^{2\times\sum_{\ell=1}^{n}\langle u\rangle_\ell}) \times (\prod_{\ell=1}^{n}[\rho]_\ell) + (\prod_{\ell=1}^{n}[\rho]_\ell) = (e^{2\times\sum_{\ell=1}^{n}\langle u\rangle_\ell} + 1) \times (\prod_{\ell=1}^{n}[\rho]_\ell)$).

4: P_i calculates $\langle v\rangle_i = \frac{\langle\phi\rangle_i - \langle\xi\rangle_i}{\chi}$ locally with the same precision.

V. Privacy Preserving RNN-Based Seq2seq Transformation

To enable privacy-preserving RNN-based seq2seq prediction, we design two secure interactive protocols among multiple parties. These protocols will be the primary blocks of SecureNLP. The RNN-based seq2seq with attention model is assumed to be publicly available (i.e., the weights and biases in the model are known for all participants). In this way, our SecureNLP just focuses on the transformation phase.

A. Privacy Preserving LSTM Network

As described above, a RNN-based seq2seq with attention model consists of two multi-layer LSTM networks. Therefore, our exploring work starts with the privacy-preserving LSTM network. A single layer of LSTM network mainly performs fully projection, sigmoid, tanh and element-wise product between vectors. Take the advantage of linear feature of additive shares, parties could perform the projection operations with the weights and biases locally. By applying the private sub-protocols PrivSigm and PrivTanh (see Section IV), the parties $P_1, \ldots, P_n$ will work together to perform the non-linear activation functions sigmoid and tanh, respectively. As for the element-wise product between vectors, it is not hard to observe that the element-wise product can be solved by applying the $\frac{n(n-1)}{2}$ two party multiplication sub-protocols π_{mul}^{2P}, which can be executed in parallel for better performance. The designed PrivLSTM is shown in Algorithm 3.

B. Privacy Preserving Seq2seq Transformation

In this section, we present a protocol supporting privacy-preserving seq2seq transformation. Recall that a RNN-based seq2seq with attention model mainly consists of two multi-layer LSTM network for encoder and decoder. Our designed privacy-preserving seq2seq model (as shown in Fig. 4) will apply the building blocks in Section IV to build the corresponding phases. The encoder could be implemented by jointly running the PrivLSTM with the additive shares of input sequence $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ and learn the additive shares of hidden states $\boldsymbol{h} = \{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T\}$.

However, the decoder is a bit more complex. It decodes the encoder's hidden states to another sequence. Let the decoder be initialized by the additive shares of $\boldsymbol{s}_0$ (i.e., assigning as the additive shares of encoder's final hidden state $\{\langle\boldsymbol{h}_T\rangle_1, \ldots, \langle\boldsymbol{h}_T\rangle_n\}$), and $\boldsymbol{y}_0$ (the publicly known identifier). Then, the parties need another PrivLSTM network to extract the information carried by the encoder's hidden states. Specifically, the attention mechanism requires a timely updated context vector that is mathematically defined as follows:

$$e_{t't} = \sum_{i=1}^{n}\langle\boldsymbol{s}_{t'-1}^{T}\rangle_i \times \sum_{j=1}^{n}\langle\boldsymbol{h}_t\rangle_j$$

$$\alpha_{t't} = \frac{\exp(e_{t't})}{\sum_{k=1}^{T}\exp(e_{t'k})}$$

$$\langle\boldsymbol{c}_{t'}\rangle_i = \sum_{t=1}^{T}\alpha_{t't}\times\langle\boldsymbol{h}_t\rangle_i, \quad \text{for } i = \{1, \ldots, n\}$$

The designed PrivSEQ2SEQ is shown in Algorithm 4.

C. The Offline Phase

We now describe how to generate Beaver's triple [30] applied in the private two party multiplication protocol π_{mul}^{2P}. Specifically, the idea of Beaver's triple is summarized as follows.

Assume that the two parties already share $\langle x\rangle$, $\langle y\rangle$, $\langle z\rangle$ where x, y are uniformly random values and $z = x \cdot y$. Then to multiply the shared values $\langle u\rangle$ and $\langle v\rangle$, party P_i, $i \in \{A, B\}$ locally calculates $\langle s\rangle_i = \langle u\rangle_i - \langle x\rangle_i$, $\langle t\rangle_i = \langle v\rangle_i - \langle y\rangle_i$. Both parties run $\mathsf{Rec}(\langle s\rangle_A, \langle s\rangle_B)$ and $\mathsf{Rec}(\langle t\rangle_A, \langle t\rangle_B)$ to learn s and t. Finally, P_A locally sets $\langle w\rangle_A = t\cdot\langle x\rangle_A + s\cdot\langle y\rangle_A + \langle z\rangle_A$ while P_B locally sets $\langle w\rangle_B = t\cdot\langle x\rangle_B + s\cdot\langle y\rangle_B + \langle z\rangle_B + s\cdot t$. This protocol also works for the shared matrices which result in the protocol being more significantly efficient due to vectorization.

Therefore, in this section, we focus on the basic step, i.e., how to compute the shared triple $\langle x\rangle$, $\langle y\rangle$, $\langle z\rangle$. The relationship between the shared triples can be mathematically presented as $\langle z\rangle_A + \langle z\rangle_B = (\langle x\rangle_A + \langle x\rangle_B) \times (\langle y\rangle_A + \langle y\rangle_B) = \langle x\rangle_A \times \langle y\rangle_A + \langle x\rangle_A \times \langle y\rangle_B + \langle x\rangle_B \times \langle y\rangle_A + \langle x\rangle_B \times \langle y\rangle_B$. We can see that $\langle x\rangle_A \times \langle y\rangle_A$ and $\langle x\rangle_B \times \langle y\rangle_B$ can be calculated locally, and the core issue now is how to compute the other two terms. The techniques are similar to prior works such as those of [26], [27], [32], [33]).

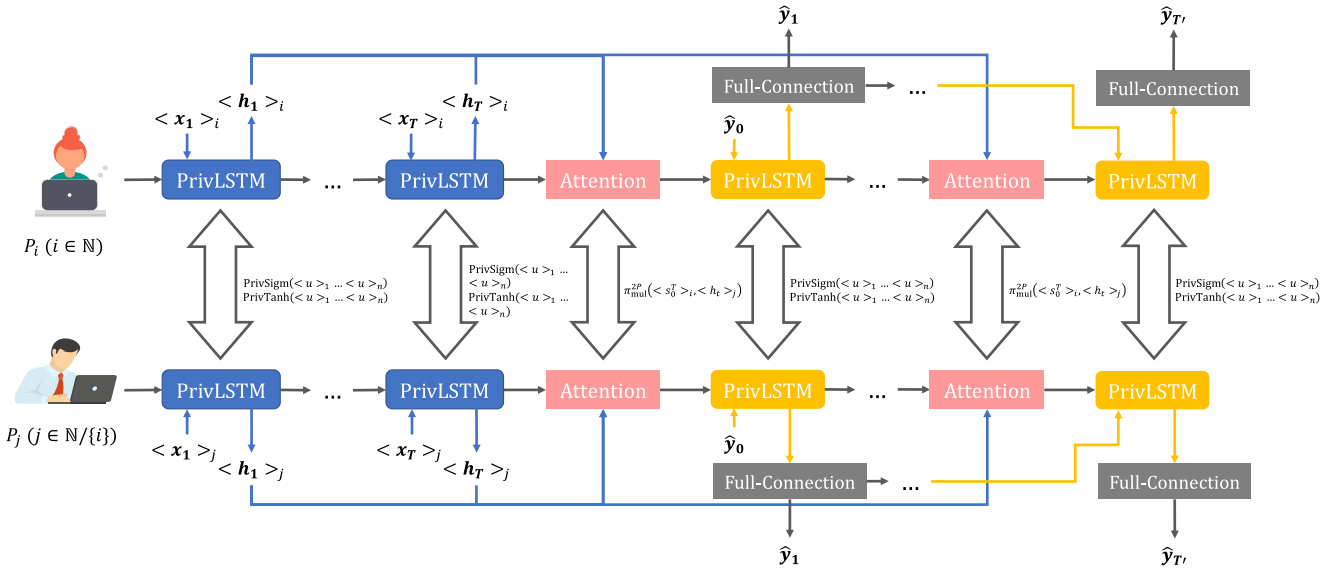

Fig. 4. The paradigm of privacy-preserving seq2seq model with attention.

Algorithm 3 The Online Phase of Privacy Preserving LSTM Network PrivLSTM

Input: Each P_i inputs sequence $\langle \boldsymbol{x} \rangle_i = \{\langle \boldsymbol{x}_1 \rangle_i, \ldots, \langle \boldsymbol{x}_T \rangle_i\}$

Output: Each P_i learns one share of output sequence $\langle \boldsymbol{h} \rangle_i = \{\langle \boldsymbol{h}_1 \rangle_i, \ldots, \langle \boldsymbol{h}_T \rangle_i\}$

1: **for** each iteration $t \in \{1, \ldots, T\}$ **do**

2: *Forget gate*:

2.a: P_1 computes $\langle \boldsymbol{net}_t^f \rangle_1 = \mathbf{W}_f \times \langle \boldsymbol{x}_t \rangle_1 + \mathbf{U}_f \times \langle \boldsymbol{h}_{t-1} \rangle_1 + \boldsymbol{b}_f$, while the other P_i compute $\langle \boldsymbol{net}_t^f \rangle_i = \mathbf{W}_f \times \langle \boldsymbol{x}_t \rangle_i + \mathbf{U}_f \times \langle \boldsymbol{h}_{t-1} \rangle_i$.

2.b: Parties run $\mathsf{PrivSigm}(\langle \boldsymbol{net}_t^f \rangle_1, \ldots, \langle \boldsymbol{net}_t^f \rangle_n)$ to learn $\langle \boldsymbol{f}_t \rangle_1, \ldots, \langle \boldsymbol{f}_t \rangle_n$ respectively.

3: *Input gate*:

3.a: P_1 computes $\langle \boldsymbol{net}_t^{in} \rangle_1 = \mathbf{W}_{in} \times \langle \boldsymbol{x}_t \rangle_1 + \mathbf{U}_{in} \times \langle \boldsymbol{h}_{t-1} \rangle_1 + \boldsymbol{b}_{in}$, $\langle \boldsymbol{net}_t^c \rangle_1 = \mathbf{W}_c \times \langle \boldsymbol{x}_t \rangle_1 + \mathbf{U}_c \times \langle \boldsymbol{h}_{t-1} \rangle_1 + \boldsymbol{b}_c$, while the other parties P_i compute $\langle \boldsymbol{net}_t^{in} \rangle_i = \mathbf{W}_{in} \times \langle \boldsymbol{x}_t \rangle_i + \mathbf{U}_{in} \times \langle \boldsymbol{h}_{t-1} \rangle_i$, $\langle \boldsymbol{net}_t^c \rangle_i = \mathbf{W}_c \times \langle \boldsymbol{x}_t \rangle_i + \mathbf{U}_c \times \langle \boldsymbol{h}_{t-1} \rangle_i$.

3.b: Parties run $\mathsf{PrivSigm}(\langle \boldsymbol{net}_t^{in} \rangle_1, \ldots, \langle \boldsymbol{net}_t^{in} \rangle_n)$ to learn $\langle \boldsymbol{g}_t \rangle_1, \ldots, \langle \boldsymbol{g}_t \rangle_n$ respectively.

3.c: Parties run $\mathsf{PrivTanh}(\langle \boldsymbol{net}_t^c \rangle_1, \ldots, \langle \boldsymbol{net}_t^c \rangle_n)$ to learn $\langle \tilde{\boldsymbol{c}}_t \rangle_1, \ldots, \langle \tilde{\boldsymbol{c}}_t \rangle_n$ respectively.

4: *State update*:

4.a: In parallel to the below, each pair of parties run the two-party multiplication protocol π_{mul}^{2P}. Party P_i inputs $\{\langle \boldsymbol{f}_t \rangle_i, \langle \boldsymbol{c}_{t-1} \rangle_i\}$; denote P_i's output by $\langle \Phi_t \rangle_i$ such that $\sum_{\ell=1}^n \langle \Phi_t \rangle_\ell = (\sum_{\ell=1}^n \langle \boldsymbol{f}_t \rangle_\ell) \circ (\sum_{\ell=1}^n \langle \boldsymbol{c}_{t-1} \rangle_\ell)$.

4.b: Similarly, each pair of parties run π_{mul}^{2P} once more. Party P_i inputs $\{\langle \boldsymbol{g}_t \rangle_i, \langle \tilde{\boldsymbol{c}}_t \rangle_i\}$ and learns $\langle \Theta_t \rangle_i$ such that $\sum_{\ell=1}^n \langle \Theta_t \rangle_\ell = (\sum_{\ell=1}^n \langle \boldsymbol{g}_t \rangle_\ell) \circ (\sum_{\ell=1}^n \langle \tilde{\boldsymbol{c}}_t \rangle_\ell)$.

4.c: P_i calculates $\langle \boldsymbol{c}_t \rangle_i = \langle \Phi_t \rangle_i + \langle \Theta_t \rangle_i$.

5: *Output gate*:

5.a: P_1 computes $\langle \boldsymbol{net}_t^{out} \rangle_1 = \mathbf{W}_o \times \langle \boldsymbol{x}_t \rangle_1 + \mathbf{U}_o \times \langle \boldsymbol{h}_{t-1} \rangle_1 + \boldsymbol{b}_o$, while the other parties P_i compute $\langle \boldsymbol{net}_t^{out} \rangle_i = \mathbf{W}_o \times \langle \boldsymbol{x}_t \rangle_i + \mathbf{U}_o \times \langle \boldsymbol{h}_{t-1} \rangle_i$.

5.b: Parties run $\mathsf{PrivSigm}(\langle \boldsymbol{net}_t^{out} \rangle_1, \ldots, \langle \boldsymbol{net}_t^{out} \rangle_n)$ to learn $\langle \boldsymbol{o}_t \rangle_1, \ldots, \langle \boldsymbol{o}_t \rangle_n$ respectively.

5.c: Parties run $\mathsf{PrivTanh}(\langle \boldsymbol{c}_t \rangle_1, \ldots, \langle \boldsymbol{c}_t \rangle_n)$ to learn $\langle \Delta_t \rangle_1, \ldots, \langle \Delta_t \rangle_n$ respectively.

5.d: Each pair of parties run π_{mul}^{2P} in parallel where party P_i inputs $\{\langle \boldsymbol{c}_t \rangle_i, \langle \Delta_t \rangle_i\}$ and learns $\langle \boldsymbol{h}_t \rangle_i$ such that $\sum_{\ell=1}^n \langle \boldsymbol{h}_t \rangle_\ell = (\sum_{\ell=1}^n \langle \boldsymbol{c}_t \rangle_\ell) \circ (\sum_{\ell=1}^n \langle \Delta_t \rangle_\ell)$.

1) LHE-Based Generation: A linear homomorphic encryption (LHE) is a cryptosystem that provides homomorphism on ciphertexts and is often applied to manipulate secure computation over private data. Here we take Paillier [28] as the underlying cryptosystem and to show how LHE-based generation done. Interested readers could refer to [34] for a relatively complex topic about homomorphic encryption. In the most general setting of Paillier, it consists of two algorithms, namely: encryption $\mathsf{Enc}_{pk}(\cdot)$ and decryption $\mathsf{Dec}_{sk}(\cdot)$, as well as supporting the computation of $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2)) = m_1 + m_2$. Therefore, the LHE-based generation is executed as follows:

Firstly, to compute the shares of the product $\langle x \rangle_A \times \langle y \rangle_B$, P_A generates a pair of Paillier key $\{pk, sk\}$ and send public key

Algorithm 4 The Online Phase of Privacy Preserving Seq2seq Transformation PrivSEQ2SEQ

Input: Each P_i inputs sequence $\langle \boldsymbol{x} \rangle_i = \{\langle \boldsymbol{x}_1 \rangle_i, \ldots, \langle \boldsymbol{x}_T \rangle_i\}$

Output: All parties obtain the output sequence $\hat{\boldsymbol{y}} = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{T'}\}$

1: Parties run $\mathsf{PrivLSTM}_1(\langle \boldsymbol{x} \rangle_1, \ldots, \langle \boldsymbol{x} \rangle_n)$ to learn $\langle \boldsymbol{h} \rangle_1, \ldots, \langle \boldsymbol{h} \rangle_n$ respectively.

2: Each party P_i locally sets the initial state for the decoder as $\langle \boldsymbol{s}_0 \rangle_i = \langle \boldsymbol{h}_T \rangle_i$, i.e., the final state of encoder at the last iteration.

3: Parties carry out the decoder by $\mathsf{PrivLSTM}_2(\langle \boldsymbol{h} \rangle_1, \ldots, \langle \boldsymbol{h} \rangle_n) \rightarrow \{\langle \boldsymbol{s} \rangle_1, \ldots, \langle \boldsymbol{s} \rangle_n\}$. Specifically, for each iteration $t' \in \{1, \ldots, T'\}$, the hidden states are updated as follows:

3.a: **for** each $t \in \{1, \ldots, T\}$ **do**:

- In parallel to the below, each pair of parties run the mini-batch two-party multiplication protocol π^{2P}_{mul} for the product of two shared vectors. Party P_i inputs $\{\{\langle \boldsymbol{s}^T_{t'-1} \rangle_i, \langle \boldsymbol{h}_t \rangle_i\}$; denote P_i's output by $\langle e_{t't} \rangle_i$.
- The parties run $\mathsf{Rec}(\langle e_{t't} \rangle_1, \ldots, \langle e_{t't} \rangle_n)$ to obtain $e_{t't}$ that is equal to $(\sum_{\ell=1}^{n} \langle \boldsymbol{s}^T_{t'-1} \rangle_\ell) \times (\sum_{\ell=1}^{n} \langle \boldsymbol{h}_t \rangle_\ell)$.

Finally, all of the parties get $e_{t'1}, \ldots, e_{t'T}$. Now, party P_i could locally compute $\alpha_{t't} = \frac{\exp(e_{t't})}{\sum_{k=1}^{T} \exp(e_{t'k})}$ (for $t \in 1, \ldots, T$), $\langle \boldsymbol{c}_{t'} \rangle_i = \sum_{t=1}^{T} \alpha_{t't} \times \langle \boldsymbol{h}_t \rangle_i$.

3.b: The first party P_1 concatenates $\boldsymbol{y}_{t'-1}$ and $\langle \boldsymbol{c}_{t'} \rangle_1$ as $\langle \tilde{\boldsymbol{y}}_{t'} \rangle_1 = [\boldsymbol{y}_{t'-1}; \langle \boldsymbol{c}_{t'} \rangle_1]$ while the other parties set $\langle \tilde{\boldsymbol{y}}_{t'} \rangle_i = \langle \boldsymbol{c}_{t'} \rangle_i$. Note that $\boldsymbol{y}_0$ is the publicly known identifier.

3.c: Parties engage in each iteration of $\mathsf{PrivLSTM}_2$ with the couple of input and previous hidden state of $\{\{\langle \tilde{\boldsymbol{y}}_{t'-1} \rangle_1, \langle \boldsymbol{s}_{t'-1} \rangle_1\}, \ldots, \{\langle \tilde{\boldsymbol{y}}_{t'-1} \rangle_n, \langle \boldsymbol{s}_{t'-1} \rangle_n\}\}$ to update the new hidden state $\langle \boldsymbol{s}_{t'} \rangle_1, \ldots, \langle \boldsymbol{s}_{t'} \rangle_n$ respectively. At this time, $\langle \boldsymbol{y}_{t'} \rangle_i$ is directly assigned by $\langle \boldsymbol{s}_{t'} \rangle_i$ which is computed jointly from LSTM's output gate.

4: P_i locally performs the full connection operation as $\langle \hat{\boldsymbol{y}} \rangle_i = \mathbf{W}_y \times \langle \boldsymbol{y} \rangle_i + \boldsymbol{b}_y$.

5: The parties run $\mathsf{Rec}(\langle \hat{\boldsymbol{y}} \rangle_1, \ldots, \langle \hat{\boldsymbol{y}} \rangle_n)$ to obtain $\hat{\boldsymbol{y}} = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{T'}\}$.

pk to P_B. Then P_A encrypts $\langle x \rangle_A$ using the LHE and submits the ciphertext $c_A = \mathsf{Enc}_{pk}(\langle x \rangle_A)$ to P_B. P_B performs the scalar multiplication and addition on the ciphertext, i.e., $c_B = c_A^{\langle y \rangle_B} \cdot \mathsf{Enc}_{pk}(\bar{r})$, where $\bar{r}$ is a uniform random number. Finally, P_A decrypts it and sets $\langle z \rangle_A = \mathsf{Dec}_{sk}(c_B)$, while P_B sets his/her share as $\langle z \rangle_B = -\bar{r}$. Remark that the number in the NLP task are rational number but the exponentiation in Paillier is specified as integer within the range of Paillier's modulus, therefore, we need to represent the numbers in fixed-point format and handle their operations based on some truncation protocols, interested reader could refer to [35]–[37] for higher precision.

2) OT-Based Generation: Oblivious transfer (OT), introduced by Gilboa [38], is another approach to calculate the shares of product $\langle x \rangle_A \times \langle y \rangle_B$. As its description, there are two entities sender and receiver engaging the protocol where sender provides two secret value $\{\omega_0, \omega_1\}$ and receiver picks one of them $\omega_{\mathscr{C}}$, $\mathscr{C} \in \{0, 1\}$, the security of this protocol requires that sender yields no information about the chosen bit $\mathscr{C}$ and receiver doesn't know another value $\omega_{1-\mathscr{C}}$. When applying in the generation of multiplication triple in $l-1$ bit length, $l-1$ invocations of 1-out-of-2 OTs are needed, implemented using an efficient OT extension. The protocol works as follows: assume that the sender is P_A with the input of $\langle x \rangle_A$ and that the receiver is P_B with the input of $\langle y \rangle_B$. Let the binary representation of $\langle y \rangle_B$ be $\langle y \rangle_B = y_\epsilon \ldots y_1 . \hat{y}_1 \ldots \hat{y}_\epsilon$ where ϵ is a publicly known precision which is must larger than the largest length of number's integer part. The y_j denotes one bit in the integer part of $\langle y \rangle_B$ and $\hat{y}_j$ indexes its fractional part. Then for each bit the two parties run a 1-out-of-2 OT protocol where the sender's inputs are $(r_j, (r_j + \langle x \rangle_A))$ (or $(\hat{r}_j, (\hat{r}_j + \langle x \rangle_A))$), and the receiver's input is y_j (or $\hat{y}_j$), where r_j or $(\hat{r}_j)$ is a random $2^{2\epsilon}$ bit float number. Denote the receiver's output as $q_j = r_j + \langle x \rangle_A \cdot y_j$ (or $\hat{q}_j = \hat{r}_j + \langle x \rangle_A \cdot \hat{y}_j$). It is easy to verify that

$$\langle x \rangle_A \times \langle y \rangle_B = \left(\sum_{k=1}^{\epsilon} 2^{k-1} \cdot q_k + \sum_{k=1}^{\epsilon} 2^{-k} \cdot \hat{q}_k \right) + \left(\sum_{k=1}^{\epsilon} -2^{k-1} \cdot r_k + \sum_{k=1}^{\epsilon} -2^{-k} \cdot \hat{r}_k \right)$$

These will therefore be the two outputs of the multiplication protocol, i.e.,

$$\langle z \rangle_A = \sum_{k=1}^{\epsilon} -2^{k-1} \cdot r_k + \sum_{k=1}^{\epsilon} -2^{-k} \cdot \hat{r}_k$$

$$\langle z \rangle_B = \sum_{k=1}^{\epsilon} 2^{k-1} \cdot q_k + \sum_{k=1}^{\epsilon} 2^{-k} \cdot \hat{q}_k$$

Keller *et al.* [12] applied OT and consistency check for a malicious secure multiplication triple generation. More recently in 2019, Doerner *et al.* [23] presented mini-batch executing protocols for shared vectors and matrices, which are known to achieve both security and efficiency. Thus, we apply an approach similar to that of [23] to minimize the operational costs in our proposed SecureNLP system.

3) Third Party-Aided Generation: Recall that both LHE and OT incur significant operational costs, e.g., LHE requires significant modular exponentiation operations, while OT requires significant data transmission. Therefore, we turn to pre-calculation for the SecureNLP system, by relying on a third party, $\mathscr{T}$ responsible for the generation of these random multiplicative triples in advance. In practice, a valid user who controls a light client can easily perform this mission. It is reasonable because that the large number of multiplication

triples required during a NLP task complicates the tracing of their utilization. Furthermore, the secure channels are available between the third party $\mathscr{T}$ and SecureNLP's parties $P_1, \ldots, P_n$. These multiplication triples can now be generated in a trusted manner without requiring computationally expensive operations or transmissions, which can significantly improve the system's efficiency. Despite its sightly weakened secure multi-party model, this third party-aided approach is probably the most promising development option for existing NLP frameworks.

VI. Security Proof

Based on the UC-model [39], [40], we provide a theoretical proof on the security of our SecureNLP system. As defined in Section III-A, we want to prove that our system can guarantee security in the presence of semi-honest adversary. In the semi-honest security model, assume that there is an adversary $\mathcal{A}$ who is able to control at most $n-1$ parties (it is trivial once $\mathcal{A}$ has corrupted all the parties). Let $I \subset \mathbb{N}$ denotes the corrupted set and $|I| < n$. Let $J = \mathbb{N} \setminus I$ be the set of honest parties. Throughout the proof, P_i $(i \in I)$ points to one of the corrupted parties, and P_j $(j \in J)$ denotes a honest party. The security definition follows approaches in prior works such as [31], [41].

Definition 1: For any polynomial-time adversary $\mathcal{A}$, if there exists a simulator S who can build a simulated world where the view of $\mathcal{A}$ is computationally indistinguishable from the view in the real world, then the protocol is secure.

Lemma 1: If all the building blocks are simulatable, then the whole system will have a complete simulation.

Lemma 2: $A\ x \cdot r$ is uniformly random if r is a uniformly random element unknown to the adversary and is completely independent from x.

Lemma 3: $A\ x \pm r$ is uniformly random if r is a uniformly random element unknown to the adversary and is completely independent from x.

Lemma 4: π_{mul}^{2P}, π_{mul}^{nP} and the protocols built by their linear combinations are secure [23], [30].

On the basis of above lemmas, we will prove that it is achievable for the simulator $\mathcal{S}$ to make $\mathcal{A}$'s view in simulated world being computationally indistinguishable from that in the real world.

Theorem 1: The protocol PrivSigm can guarantee semi-honest security even on the condition that an adversary has corrupted any $t < n$ parties.

Proof: The view of PrivSigm for corrupted party P_i $(i \in I)$ is $view_i = \{\langle u\rangle_i, [\rho]_i, \langle\xi\rangle_i, [k]_i, \langle\phi\rangle_i, \langle\chi\rangle_i, \chi, \langle v\rangle_i\}$, where $[\rho]_i$ is uniformly randoms. From Lemma 2, $k_i = e^{\langle u\rangle_i} \times [\rho]_i$ is still uniformly random values. Owning to the fact that the intermediate values $\langle\xi\rangle_i$ and $\langle\phi\rangle_i$ are outputs from π_{mul}^{nP}, they are also all uniformly randoms on the basis of Lemma 4. According to Lemma 3, the results of $\langle\chi\rangle_i = \langle\phi\rangle_i - \langle\xi\rangle_i$, $\chi = \sum_{\ell=1}^{n}\langle\chi\rangle_i$ are still uniform. Then, $\langle v\rangle_i$ expressed as the multiplication of two random values, i.e., $\frac{\langle\phi\rangle_i}{\chi}$ is also uniformly random. Consequently, $view_i$ is simulatable and it is unable to find a probabilistic polynomial-time algorithm to distinguish $view_i$ and the simulated view for P_i. Thus, our PrivSigm is secure in the semi-honest model. □

Theorem 2: The protocol PrivTanh can guarantee semi-honest security even on the condition that an adversary has corrupted any $t < n$ parties.

Proof: Similar to the proof of PrivSigm, the view for a corrupted party P_i $(i \in I)$ during an execution of PrivTanh is $view_i = \{\langle u\rangle_i, [\rho]_i, \langle\xi\rangle_i, [k]_i, \langle\phi\rangle_i, \langle\chi\rangle_i, \chi, \langle v\rangle_i\}$. Since that $[\rho]_i$ is independently chosen value and $k_i = e^{2\times\langle u\rangle_i} \times [\rho]_i$, both of them are uniformly random based on Lemma 2. Meanwhile, $\langle\xi\rangle_i$ and $\langle\phi\rangle_i$ are obtained through two independent sessions of π_{mul}^{nP} that has been proven secure in the semi-honest model. Then, according to Lemma 3 and Lemma 2, the results of $\langle\chi\rangle_i = \langle\phi\rangle_i + \langle\xi\rangle_i$, $\chi = \sum_{\ell=1}^{n}\langle\chi\rangle_i$, $\langle v\rangle_i = \frac{\langle\phi\rangle_i - \langle\xi\rangle_i}{\chi}$ are still uniformly. Therefore, it is easy for the simulator $\mathcal{S}$ to generate a view that is computationally indistinguishable from $view_i$. Thus, our PrivTanh can be proven security in the semi-honest model. □

Theorem 3: The protocol PrivLSTM can guarantee semi-honest security even on the condition that an adversary has corrupted any $t < n$ parties.

Proof: According to Algorithm 3, There are three gates composing one layer of LSTM network. The interactive protocols for the forget gate and input gate are only based on PrivSigm and PrivTanh, who have been proved to be prefectly similatable as above. For the state update, the view for a corrupted party P_i $(i \in I)$ is $view_i^1 = \{\langle \boldsymbol{f}_t\rangle_i, \langle \boldsymbol{c}_{t-1}\rangle_i, \langle\Phi_t\rangle_i, \langle \boldsymbol{g}_t\rangle_i, \langle \tilde{\boldsymbol{c}}_t\rangle_i, \langle\Theta_t\rangle_i, \langle \boldsymbol{c}_t\rangle_i\}$, where $\langle\Phi_t\rangle_i$ and $\langle\Theta_t\rangle_i$ are learned from two independent sessions of sub-protocol π_{mul}^{nP} with the inputs of $\{\langle \boldsymbol{f}_t\rangle_i, \langle \boldsymbol{c}_{t-1}\rangle_i\}$ and $\{\langle \boldsymbol{g}_t\rangle_i, \langle \tilde{\boldsymbol{c}}_t\rangle_i\}$ respectively. Thus, they are uniformly random on the basis of Lemma 4 and their linear combination $\langle \boldsymbol{c}_t\rangle_i = \langle\Phi_t\rangle_i + \langle\Theta_t\rangle_i$ is also uniformly random. As a consequence, $view_i^1$ is simulatable and computationally indistinguishable with the simulated view. In the same way, the simulator can also generate a polynomial-time indistinguishable view for the output gate, i.e. $view_i^2 = \{\langle \boldsymbol{net}_t^{out}\rangle_i, \langle \boldsymbol{o}_t\rangle_i, \langle\Delta_t\rangle_i, \langle \boldsymbol{h}_t\rangle_i\}$ for the corrupted party P_i. Thus, our PrivLSTM can be proven security in the semi-honest model according to Lemma 1. □

Theorem 4: The protocol PrivSEQ2SEQ can guarantee semi-honest security even on the condition that an adversary has corrupted any $t < n$ parties.

Proof: In Section V-B, we know that there are two LSTM networks for the forward algorithm of privacy-preserving seq2seq model. The encoder can be directly achieved by one PrivLSTM sub-protocol whose security has been proved in Theorem 3. As for the decoder, before entering into each iteration of the PrivLSTM, the views of the corrupted party P_i $(i \in I)$ will be $view_i = \{\langle \boldsymbol{s}_{t'-1}^T\rangle_i, \langle \boldsymbol{h}_t\rangle_i, \langle e_{t't}\rangle_i, e_{t't}, \alpha_{t't}, \langle \boldsymbol{c}_{t'}\rangle_i, \langle \tilde{\boldsymbol{y}}_{t'}\rangle_i\}$. It's trivial to see that they are uniformly random and can be simulated by the simulator $\mathcal{S}$, due to the fact that $\langle e_{t't}\rangle_i$ is output after executing $\frac{n(n-1)}{2}$ sub-protocol π_{mul}^{2P}, and the other intermediate values are linear combinations of $\langle e_{t't}\rangle_i$. Besides, the final output are calculated locally by P_i and P_j, $output_i$ is also simulatable by the simulator $\mathcal{S}$. Therefore, our protocol PrivSEQ2SEQ is secure in the semi-honest model according to Lemma 1. □

TABLE I
RUNTIME FOR ONLINE AND OFFLINE PHASES

Number of parties	Online		Offline
	PrivSigm	PrivTanh	
single party	9.1e-08 s	1.07e-08 s	$\perp$[1]
2	0.0218 s	0.0225 s	33.7521 s
4	0.0684 s	0.0691 s	33.8113 s
8	0.1589 s	0.1589 s	33.6672 s
16	0.343 s	0.3491 s	33.9324 s
32	0.7024 s	0.7041 s	34.0185 s
64	1.4237 s	1.424 s	33.7125 s

[1] The original sigmoid and tanh are calculated by single party locally that can do without pre-computation instead.

VII. CORRECTNESS AND EFFICIENCY ANALYSIS

In this section, we use experiments to demonstrate the correctness and efficiency of SecureNLP and its underpinning protocols, PrivSigm, PrivTanh. To experiment the performance of SecureNLP, we adopt the date from the standard translation evaluation database WMT [42]. We implement SecureNLP with Python and C++, and execute it on laptop with an Intel(R) Core(TM) i7-6700 CPU @3.40GHz and 8.00GB of RAM in the LAN setting. Our implementation is based on the libraries of MXNET [43] and EMP-Toolkit [44].

A. Performance of Private Sigmoid and Tanh

In SecureNLP, the operations in the RNN are both reduced to the private sigmoid and tanh, i.e., PrivSigm and PrivTanh. Since the bandwidth of our LAN is limited, we follow the idea of π_{mul}^{nP} in [23] but replace the two-rational multiplication (operations like $x_1 \cdot x_2$ for two secret values) used in π_{mul}^{nP} by the LHE-based approach and apply the optimzation algorithm in [45]. Therefore, the generations of Paillier key pairs and other pre-calculation materials are the only calculation tasks left during the offline phases for private sigmoid and tanh (which would be constant even the number of parties increases). The online phases include the computation of sigmoid and tanh functions jointly by multiple parties, from 2 to 64 parties. Note that we also evaluate the running time in the single party setting to provide a comparison of our SecureNLP. As shown in Table I and Fig. 5, the running times of online phases go up in the number of parties. It is quite clear that as the number of parties increase, the computation costs will increase exponentially. It is because that for n parties, one private sigmoid or one private tanh needs $O(\log n)$ time complexity (each π_{mul}^{nP} protocol consists of n^2 two-rational multiplication sub-protocols but can be optimized by executing in $O(\log n)$ rounds in parallel.) However, most of them are in a matter of "milliseconds" matters. In particular, when we set the parties' number as 2 for example, the runtime is less than 21.8 ms and 22.5 ms, respectively. As for the set up phase, each party just need to generate the pre-calculation materials that is not affected by the number of parties, thus the offline overhead for PrivSigm and PrivTanh only have a tiny difference.

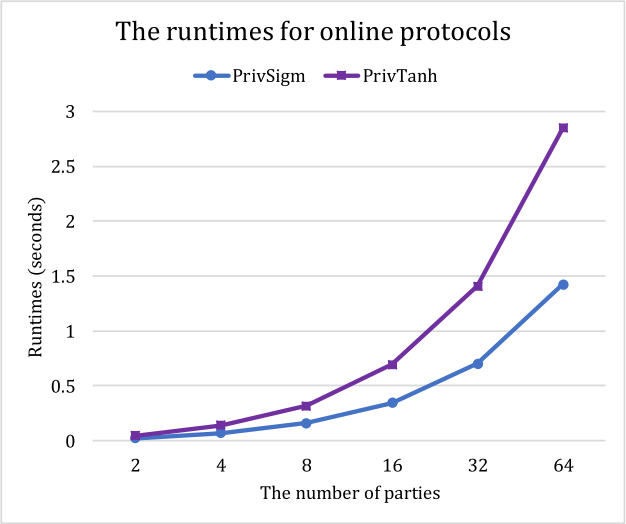

Fig. 5. Runtime for online phase of building blocks.

TABLE II
RUNTIME AND COMMUNICATION OVERHEADS OF MULTIPLICATION TRIPLES' GENERATION

Generation Protocols	Runtime[1]	Communication
LHE-based	2108.18 ms	32 KB
OT-based	375.68 ms	320 KB
Third Party-Aided	0.520 ms	$\perp$[2]

[1] Each runtime is defined to be the cost for the generation of 64 multiplication triples.
[2] Third party generates the auxiliary multiplication triples independently and has no need of communications with others.

B. Performance of SecureNLP

As for the private two-party multiplication π_{mul}^{2P} used in SecureNLP (operations like $(x_1+x_2)\cdot(y_1+y_2)$ for two secret values additively shared by two parties), we implemented it based on the third party-aided approach in Section V-C3. Thus some random numbers and random multiplication triples will be generated by the third party within the offline phases. However, the third party-aided setting is not completely decentralized, we further analyze the three different approaches for the generation of multiplication triples, i.e., LHE-based generation, OT-based generation, and third party-aided generation. For LHE-based approach, we set $\epsilon = 8$ and apply the optimized Paillier scheme with the modulus as a 2048-bit length biprime. Note that when using π_{mul}^{2P} based on Paillier, one private multiplication between two parties requires 4 Paillier exponentiation and sending 2 Paillier ciphertexts. The OT-based protocol is based on KOS [46] that requirs η base OTs for security parameter η (each of the OT protocol costs 3 exponentiation for the sender and 2 for the receiver and sending 4 group elements) as analyzed in [26]. Thus in SecureNLP, for one private multiplication between two parties, we count $2 \cdot \epsilon \cdot 2.5 = 40$ exponentiation for each parties. Table II clearly shows the communication/computation tradeoff between OT-based, Paillier-based, third party-aided variants. Finally, the experimental results also show that the

TABLE III
RUNTIME FOR DIFFERENT BATCH SIZES BETWEEN TWO PARTIES

Batch Size[1]	Runtime	Communication
1	0.128 s	2.55 KB
16	1.781 s	40.75 KB
32	3.547 s	81.5 KB
64	7.08 s	163 KB

[1] We keep some parameters constant, e.g., the longest length of a float number in NLP mission is 32 bit and all numbers of hidden units are pre-set as 1.

TABLE IV
THE RUNTIMES AND MESSAGE SIZES WITH DIFFERENT NUMBER OF PARTIES

Number of Parties [1]	Runtimes	Message Size
single party	0.0248 s	$\perp$[2]
2	7.08 s	0.159 MB
4	21.992 s	0.995 MB
8	50.862 s	4.458 MB

[1] Here, the parties are running the private NLP task with batch size = 1 and hidden units = 64.
[2] Single party executes the original NLP task which does not require any message transmission.

accuracies of OT-based and Paillier-based approaches with the previous $\epsilon = 8$ are approximately to 1 comparing to the functions in the single party setting.

Before the experiments, we first define a 47-input and 39-output vocabulary of French-English pairs for training of the SecureNLP system that consists of two LSTM networks for encoder and decoder. For efficiency and practicality, we employ the NumPy library of Python to concurrently operate the matrix or vector computations. All of the following experiments use the same configuration, expect the one for evaluation.

1) Accuracy: We obtain the evaluation result of 1.000 from BLEU (Bilingual Evaluation Understudy [47]) if the output tokens have been in the training dictionary which is consistent with the networks in the single party setting. Essentially, this is due to the fact that the accuracies of private sigmoid and tanh protocols are approximately equal to those of single-party functions. Thus, our system can be applied to other natural language processing tasks relying on the sigmoid and tanh activation functions. More importantly, to make the system fully decentralized could simply replaces the offline phases based on LHE or OT.

2) Efficiency: We test the cost of the runtime and communication overheads with different batch size. The number of parties is set as 2. As shown in Table III, the runtime and communication overhead is almost linearly with the batch size of NLP tasks. Furthermore, we evaluate the runtime and communication overheads of working one NLP mission with the batch-size 64 in Table IV, Fig. 6 and Fig. 7. It is quite clear that as the number of parties increases, both the computation and communication costs increase exponentially

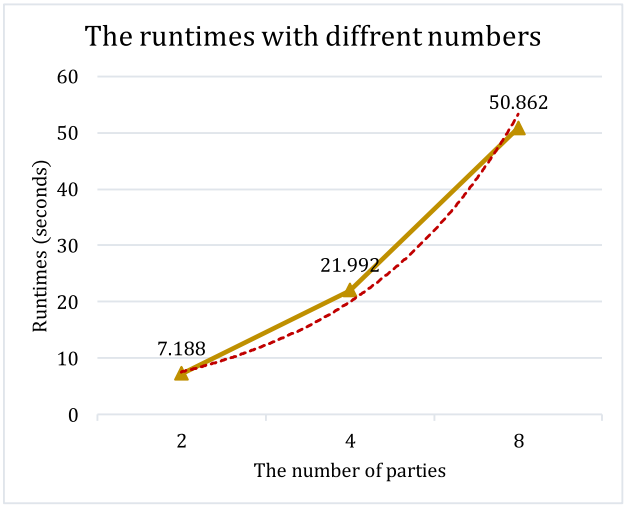

Fig. 6. Runtime for seq2seq mission with different party numbers. The red line indicates the trend in runtime as the number of parties increases.

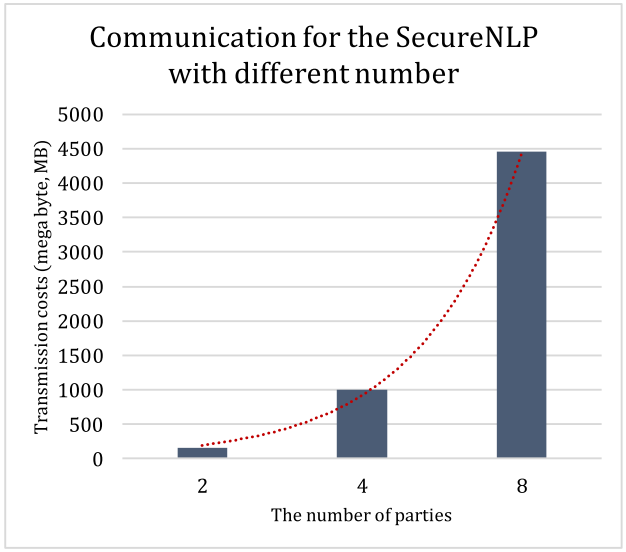

Fig. 7. Communication costs for seq2seq mission with different party numbers. The red line indicates the trend in communication overhead as the number of parties increases.

since that for n parties, one private sigmoid or one private tanh are $O(\log n)$ depth (impacting the time complexity) and $O(\sqrt{n})$ width (impacting the communication complexity). Here, the runtime in online phase mainly include interactions within the private sigmoid and Tanh protocols, while the generation of random rationals are performed in advance by the parties and generation of multiplication triples are performed by the third party in the offline phase.

Finally, we further find out the heavest operations is the private sigmoid and tanh, whose precentages are in the neighborhood of 59% and 40% (as shown in Fig. 8). It can been inferred that the linear operations, which can be performed locally, have no notably impacts on the performance and our designed SecureNLP is blocked by the activation functions. How to optimize these two private protocols will be a open problem that we may explore for in the future.

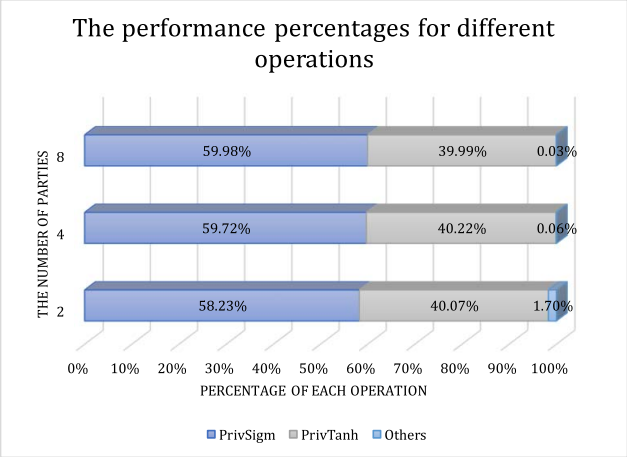

Fig. 8. Performance in percentage, for different operation during the seq2seq mission.

VIII. Conclusion

NLP systems are working with more data sources as our world becomes more digitalized. Therefore, there is also a corresponding need to ensure data privacy when executing multi-source NLP tasks, and hence the need for privacy-preserving multi-party NLP system (i.e., no trusted authority). Therefore, in this paper, we presented SecureNLP. We started our exploring by two private building blocks for the non-linear activation functions, i.e., PrivSigm and PrivTanh. Then, we used these two building blocks to design a series of interactive protocols for privacy-preserving RNN-based seq2seq with the attention model. These protocols allow us to ensure that all parties engage in the seq2seq transformation, without compromising the privacy of the data. We then demonstrated the security and efficiency of the proposed approach.

However, multi-party multiplication may still be computationally expensive for some lightweight, inexpensive devices. Therefore, one future research agenda is to design an optimized version with more lightweight operations, such as those based on the elliptic curve cryptosystem.

References

[1] Tractica. (2017). Natural language processing market to reach $22.3 billion by 2025. Website.[Online]. Available: https://www.tractica.com/newsroom/press-releases/natural-language-processing-market-to-reach-22-3-billion-by-2025/

[2] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, Oct. 2017, pp. 1175–1191.

[3] K. Xu, H. Yue, L. Guo, Y. Guo, and Y. Fang, "Privacy-preserving machine learning algorithms for big data systems," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jun. 2015, pp. 318–327.

[4] Y. Zhao, Y. Yu, Y. Li, G. Han, and X. Du, "Machine learning based privacy-preserving fair data trading in big data market," *Inf. Sci.*, vol. 478, pp. 449–460, Apr. 2019.

[5] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, Jun. 2018.

[6] Y. Jiang *et al.*, "SecureLR: Secure logistic regression model via a hybrid cryptographic protocol," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 16, no. 1, pp. 113–123, Jan. 2019.

[7] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 2046–2058, Dec. 2013.

[8] X. Liu, R. H. Deng, K.-K.-R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2401–2414, Nov. 2016.

[9] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci. (SFCS)*, Nov. 1982, pp. 160–164.

[10] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptol.*, vol. 22, no. 2, pp. 161–188, Apr. 2009.

[11] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology—CRYPTO*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Germany: Springer, 2012, pp. 643–662.

[12] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 830–842.

[13] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[14] X. Liu, R. H. Deng, Y. Yang, H. N. Tran, and S. Zhong, "Hybrid privacy-preserving clinical decision support system in fog–cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 825–837, Jan. 2018.

[15] X. Liu, R. Lu, J. Ma, L. Chen, and B. Qin, "Privacy-preserving patient-centric clinical decision support system on Naïve Bayesian classification," *IEEE J. Biomed. Health Inform.*, vol. 20, no. 2, pp. 655–668, Mar. 2016.

[16] L. Liu *et al.*, "Privacy-preserving mining of association rule on outsourced cloud data from multiple parties," in *Information Security and Privacy*. Cham, Switzerland: Springer, 2018, pp. 431–451.

[17] X. Liu, R. Deng, K.-K.-R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Trans. Cloud Comput.*, early access, Feb. 5, 2018, doi: 10.1109/TCC.2018.2799219.

[18] J. Hua, G. Shi, H. Zhu, F. Wang, X. Liu, and H. Li, "CAMPS: Efficient and privacy-preserving medical primary diagnosis over outsourced cloud," *Inf. Sci.*, vol. 527, pp. 560–575, Jul. 2020.

[19] X. Liu, R. Deng, K.-K.-R. Choo, and Y. Yang, "Privacy-preserving reinforcement learning design for patient-centric dynamic treatment regimes," *IEEE Trans. Emerg. Topics Comput.*, early access, Jan. 30, 2019, doi: 10.1109/TETC.2019.2896325.

[20] Z. Ma, Y. Liu, X. Liu, J. Ma, and K. Ren, "Lightweight privacy-preserving ensemble classification for face recognition," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5778–5790, Jun. 2019.

[21] Y. Yu, H. Li, R. Chen, Y. Zhao, H. Yang, and X. Du, "Enabling secure intelligent network with cloud-assisted privacy-preserving machine learning," *IEEE Netw.*, vol. 33, no. 3, pp. 82–87, May 2019.

[22] T. Li, J. Li, X. Chen, Z. Liu, W. Lou, and T. Hou, "NPMML: A framework for non-interactive privacy-preserving multi-party machine learning," *IEEE Trans. Dependable Secure Comput.*, early access, Feb. 4, 2020, doi: 10.1109/TDSC.2020.2971598.

[23] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 399–414.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2014, pp. 3104–3112.

[25] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734.

[26] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, Jan. 2018, pp. 1837–1854.

[27] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 980–997.

[28] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 1999, pp. 223–238.

[29] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, "A lightweight privacy-preserving CNN feature extraction framework for mobile sensing," *IEEE Trans. Depend. Sec. Comput.*, early access, Apr. 26, 2019, doi: 10.1109/TDSC.2019.2913362.

[30] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Int. Cryptol. Conf. Adv. Cryptol.*, 1991, pp. 420–432.
[31] Y. Lindell, "How to simulate it—A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Cham, Switzerland: Springer, 2017, pp. 277–346.
[32] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2015, pp. 1–15.
[33] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, Jan. 2018, pp. 1179–1194.
[34] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, Sep. 2018.
[35] J. Algesheimer, J. Camenisch, and V. Shoup, "Efficient computation modulo a shared secret with application to the generation of shared safe-prime products," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2002, pp. 417–432.
[36] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2010, pp. 35–50.
[37] O. Catrina, "Round-efficient protocols for secure multiparty fixed-point arithmetic," in *Proc. Int. Conf. Commun. (COMM)*, Jun. 2018, pp. 431–436.
[38] N. Gilboa, "Two party RSA key generation," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 1999, pp. 116–129.
[39] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 136–145.
[40] R. Canetti, A. Cohen, and Y. Lindell, "A simpler variant of universally composable security for standard multiparty computation," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2015, pp. 3–22.
[41] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2008, pp. 192–206.
[42] (May 27, 2020). *WMT*. [Online]. Available: http://www.statmt.org/wmt14/translation-task.html
[43] (May 27, 2020). *MXNET: A Flexible and Efficient Library for Deep Learning*. [Online]. Available: http://mxnet.incubator.apache.org/
[44] X. Wang, A. J. Malozemoff, and J. Katz. (2016). *EMP-Toolkit: Efficient MultiParty Computation Toolkit*. [Online]. Available: https://github.com/emp-toolkit
[45] C. Jost, H. Lam, A. Maximov, and B. J. Smeets, "Encryption performance improvements of the Paillier cryptosystem," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 864, Sep. 2015.
[46] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2015, pp. 724–741.
[47] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics (ACL)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318.

Qi Feng received the bachelor's and master's degrees from the School of Computer Science, Wuhan University, Wuhan, China, in 2016 and 2018, respectively, where she is currently pursuing the Ph.D. degree with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering. Her research interest includes cryptographic protocols.

Debiao He (Member, IEEE) received the Ph.D. degree in applied mathematics from the School of Mathematics and Statistics, Wuhan University, Wuhan, China, in 2009. He is currently a Professor at the School of Cyber Science and Engineering, Wuhan University, Wuhan. His main research interests include cryptography and information security, in particular, cryptographic protocols. He has published over 100 research papers in refereed international journals and conferences, such as the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and Usenix Security Symposium. He was a recipient of the 2018 IEEE SYSTEMS JOURNAL Best Paper Award and the 2019 IET Information Security Best Paper Award. His work has been cited more than 7000 times at Google Scholar. He is in the editorial board of several international journals, such as *Journal of Information Security and Applications*, *Frontiers of Computer Science*, and *Human-centric Computing and Information Sciences*.

Zhe Liu (Senior Member, IEEE) received the B.S. and M.S. degrees from Shandong University in 2008 and 2011, respectively, and the Ph.D. degree from the Laboratory of Algorithmics, Cryptology and Security, University of Luxembourg, Luxembourg, in 2015. He was a Researcher with SnT, University of Luxembourg, Luxembourg. He is currently a Professor at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include computer arithmetic and information security. He has coauthored over 70 research peer-reviewed journal and conference papers. His Ph.D. thesis has received the prestigious FNR Awards 2016–Outstanding Ph.D. Thesis Award for his contributions in cryptographic engineering on IoT devices.

Huaqun Wang received the B.S. degree in mathematics education from Shandong Normal University, China, in 1997, the M.S. degree in applied mathematics from East China Normal University, China, in 2000, and the Ph.D. degree in cryptography from the Nanjing University of Posts and Telecommunications in 2006. He is currently a Professor at the Nanjing University of Posts and Telecommunications, China. His research interests include applied cryptography, network security, and cloud computing security.

Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA). In 2016, he was named the Cybersecurity Educator of the Year - APAC (Cybersecurity Excellence Awards are produced in cooperation with the Information Security Community on LinkedIn), and in 2015, he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen-Nuremberg. He was a recipient of the 2019 IEEE Technical Committee on Scalable Computing (TCSC) Award for Excellence in Scalable Computing (Middle Career Researcher), the 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty, the Outstanding Associate Editor of 2018 for IEEE ACCESS, the British Computer Society's 2019 Wilkes Award Runner-up, the 2019 *EURASIP Journal on Wireless Communications and Networking* (JWCN) Best Paper Award, the Korea Information Processing Society's *Journal of Information Processing Systems* (JIPS) Survey Paper Award (Gold) 2019, the IEEE Blockchain 2019 Outstanding Paper Award, the IEEE TrustCom 2018 Best Paper Award, the ESORICS 2015 Best Research Paper Award, the 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, the Fulbright Scholarship in 2009, the 2008 Australia Day Achievement Medallion, and the British Computer Society's Wilkes Award in 2008. He is also a fellow of Australian Computer Society, and the Co-Chair of the IEEE Multimedia Communications Technical Committee's Digital Rights Management for Multimedia Interest Group.