VerifyTL: Secure and Verifiable Collaborative Transfer Learning

Zhuoran Ma, Jianfeng Ma, Member, IEEE, Yinbin Miao, Ximeng Liu, Senior Member, IEEE, Wei Zheng, Kim-Kwang Raymond Choo, Senior Member, IEEE, and Robert H. Deng, Fellow, IEEE

Abstract—Getting access to labeled datasets in certain sensitive application domains can be challenging. Hence, one may resort to transfer learning to transfer knowledge learned from a source domain with sufficient labeled data to a target domain with limited labeled data. However, most existing transfer learning techniques only focus on one-way transfer which may not benefit the source domain. In addition, there is the risk of a malicious adversary corrupting a number of domains, which can consequently result in inaccurate prediction or privacy leakage. In this paper, we construct a secure and Verifiable collaborative Transfer Learning scheme, VerifyTL, to support two-way transfer learning over potentially untrusted datasets by improving knowledge transfer from a target domain to a source domain. Furthermore, we equip VerifyTL with a secure and verifiable transfer unit employing SPDZ computation to provide privacy guarantee and verification in the multi-domain setting. Thus, VerifyTL is secure against malicious adversary that can compromise up to \( n - 1 \) out of \( n \) data domains. We analyze the security of VerifyTL and evaluate its performance over four real-world datasets. Experimental results show that VerifyTL achieves significant performance gains over existing secure learning schemes.

Index Terms—Convolutional neural network, dishonest majority, malicious security, spdz, transfer learning.

I. INTRODUCTION

WITH the increasing deployment of Internet of Things (IoT) devices and digitalization of our society, the amount of digital data generated and collected will also increase significantly. This also contributes to renewed interest in Artificial Intelligence (AI), such as deep learning techniques. For example, Convolutional Neural Network (CNN) [1] has been widely used to facilitate image processing, facial recognition and fingerprint identification. The construction of data-driven CNN model typically requires intensive data resources for analysis and recognition. However, sharing data across systems may not be easy in practice, for example due to security and privacy concerns [2]. In addition, labeled datasets that can be used in AI model training may also be limited in certain sensitive application domains.

Transfer learning [3] can potentially be used to overcome such a limitation, by transferring knowledge learned on one data-rich domain (i.e., source domain) to another data-starved domain (i.e., target domain). However, as shown in Fig. 1, existing transfer learning mechanisms have a number of limitations. A key challenge in conventional transfer learning is the privilege of data-starved domains, which obtain the transferred knowledge from data-rich domains with no payout. It is a real struggle for a data-rich domain to perform the “selfless” behavior (i.e., knowledge transfer) in transfer learning, as data collection, curation, labeling, etc., come at a cost. With the difficulty of data shortage, it is impractical to find the “selfless” data-rich domains in transfer learning. Therefore, it is also necessary to provide a peer-to-peer architecture, where each domain contributes individual knowledge to construct the global model collaboratively.

Motivated by the aforementioned challenges in transfer learning, Collaborative Learning (CL) [4, 5] (also called federated learning) is a potential solution to construct the global CNN model in the peer-to-peer architecture. Instead of sharing training data, CL constructs the global model by aggregating locally-trained model parameters on domain-held data. Without
the central server, CL avoids the single point of failure and communication bottlenecks [6]. However, existing CL mechanisms still face the following issues in practical scenarios, which can degrade the performance of the global model.

A. Practical Scenarios

As depicted in Fig. 1, multiple mutually-distrusting data domains are involved in a scenario of newly-developing applications, such as financial analysis and medical diagnosis. \(D_1\) is a data-rich domain, which contains a big volume of data with authorization from its users. \(D_2\) and \(D_3\) are two data-poor domains belonging to the same institution \(I\). For its own benefits, \(I\) is willing to train a global model with other domains (i.e., \(D_1\)) belonging to other institutions, which allows its domains (i.e., \(D_2\) and \(D_3\)) honestly submit local parameters to others. However, with the competitive interests, \(I\) tries to corrupt the collaboration with other domains by comprising the transfer learning process, such as malicious computations and returning wrong results. After obtaining the intermediate results from others, a group of domains of \(I\) privately construct their model with the profits. For self-serving reasons, the vicious competition will lead the negative transfer [7], [8] for other domains. There are malicious behaviors in many environments, including business, financial and political applications. Involved institutions and individuals are willing to actively cheat for competitive interests, but seek to high-performance transfer learning for self-interest.

In the above scenario, there are two issues that hinder the development of transfer learning by simply using CL scheme. The first issue is privacy concerns. Locally-trained model parameters contain data privacy, which may be vulnerable to inference attacks including membership attacks [9] and reconstruction attacks [10]. To avoid the disclosure of the training data [11], existing CL schemes exploit cryptographic techniques including secure Multi-Party Computation (MPC), homomorphic encryption and differential privacy for data privacy. Existing Privacy-Preserving CL (PPCL) schemes design secure computations to privately construct the global model with encrypted locally-trained model parameters, which is based on honest-but-curious assumptions. In PPCL, all data domains are curious to others’ private information, but honestly implement the protocol for training. However, the strong assumption is unrealistic for the practical scenarios [12]. Hence, there may exist a malicious adversary [13], which attempts to corrupt the changing set of data domains to alter their behaviors. Corrupted domains can attempt to compromise the protocol, for example by executing a malicious computation, and changing the computation results in secure computations with negative transfer [7], [8]. Thus, a malicious adversary can launch malicious learning with dishonest majority [14] by maliciously tuning transfer learning, resulting in transfer learning behaving badly on specific attacker-chosen inputs. With the existence of malicious domains, it cannot achieve the accuracy of the global model. Unfortunately, directly adopting the state-of-the-art maliciously-secure MPC protocol such as SPDZ [15] is difficult to implement the complicated computation protocols in transfer learning such as non-linear activation functions (i.e., Sigmoid and ReLU), which lead to significant overheads to achieve strong security. The second issue is to handle skewed data. In transfer learning, data-rich domains hold numerous training samples, while data-starved domains hold poor training samples. The data volume distribution is not uniform across different domains, with which existing CL schemes fail to achieve high performance with CNN models [16]. Besides, data-starved domains hold limited local data samples, with which local objectives differ greatly from the global objective [17]. Thus, it is difficult to update local data-driven CNN with tiny-sized training samples, which can reduce the convergence rate and involve repetitive training iterations. Even worse, the performance of the collaborative model may be degraded.

B. Our Contributions

To address the above issues, this paper proposes a novel secure and verifiable collaborative transfer learning with the strong guarantee of malicious security, which implements the high-performance and privacy-preserving transfer learning against malicious adversaries, hereafter referred to as VerifyTL. In VerifyTL, even if \(n - 1\) out of \(n\) domains fully misbehave and collude, the honest domain can still guarantee data privacy. A summary of our contributions is as follows:

- **Malicious security.** VerifyTL supports malicious security, in which a malicious adversary can corrupt \(n - 1\) out of \(n\) data domains. Each data domain only trusts itself to prevent the corruption of dishonest majority. VerifyTL designs the SPDZ-based transfer learning scheme, which not only protects data privacy, but also verifies the correctness of the final result with a Message Authentication Code (MAC) to prevent malicious behaviors.

- **High-performance collaborative transfer learning.** We present a collaborative transfer learning framework in the peer-to-peer architecture, which redesigns transfer learning with multiple data-poor domains to address the issues of skewed data. Specifically, we deploy a secure and collaborative transfer learning unit, which supports multi-domain settings.

- **Implementations.** We evaluate VerifyTL using four real-world datasets. Compared with original learning implemented CNN training without transfer learning, the performance gains with VerifyTL are up to 19.7% for USPS data, 20.59% for MNIST data, 34.65% for Fashion MNIST data, and 68.25% for CIFAR-10 data.

In the next two sections, we will review the related literature and introduce relevant background materials. In Section 4, we will introduce the system model of VerifyTL, the threat model of malicious security, and design goals. In Section 5, we present the proposed VerifyTL, prior to evaluating its security and performance in Sections 6 and 7. In the last section, we conclude this paper.

II. RELATED WORK

**Transfer Learning.** Transfer learning has the potential to address the data scarcity problem of data-starved domains (i.e.,
target domains), which transfers knowledge learned from data-rich domains (i.e., source domains) [3,18]. Earlier approaches mainly focus on transferring the training data from one or more source domains to another [19], [20]. However, such approaches either incur significant communication costs during the transmission of large amounts of data from the source domain or do not support heterogeneous transfer among different feature distributions. The existing schemes such as those presented in [21], [22] used a TrAdaBoost approach, which reuses training data of source domains for implementing knowledge transfer. However, TrAdaBoost requires access to training data on both source and target domains. Consequently, the target domain can learn the training dataset of the source domain(s). Oquab et al. [23], [24] proposed a CNN-based transfer learning scheme that transfers image representations learned with CNNs on large-scale annotated datasets to other tasks with limited training data. Shin et al. [25] designed a transfer learning method that transfers fine-tuning CNN models pre-trained from natural image datasets to medical image tasks for image diagnosis. Kendall et al. [26] presented a principal approach to multi-task deep learning, which weighs multiple loss functions by considering the homoscedastic uncertainty of each source task. However, these schemes only support one-way transfer learning, i.e., knowledge is transferred from a source domain to a task domain. With the privilege of target domains, it is impractical to find a “selfless” source domain with generous knowledge sharing. Hence, two-way transfer learning methods such as those presented in [27], [28] use a cross-stitch network with CNN models for multi-task learning. These methods enable dual knowledge transfers across domains by utilizing cross-connections from one task to another and vice versa. However, these two-way transfer learning methods are confined to the two-domain setting, but not the multi-domain setting. Unfortunately, locally-trained model parameters may be revealed, for example by successfully carrying out an inference attack to reconstruct the training data [9], [11]. In other words, there is a risk of information leakage.

**Privacy-Preserving Collaborative Learning.** To avoid the above challenges, Privacy-Preserving Collaborative Learning (PPCL) [29] is a workable solution with a peer-to-peer architecture, which tolerates network latency and single point of failures. In PPCL, each domain exchanges locally-trained model parameters to construct the global model instead of data outsourcing. As local model parameters also contain sensitive information, the adversary can still infer the properties of data from the exchanged model parameters [30]. Hence, existing PPCL schemes [31], [32] exploit cryptographic tools to encrypt local model parameters for privacy guarantee, for example by utilizing MPC technologies [31], [33], [34]. Mohassel et al. [33] designed a general framework for privacy-preserving machine learning for linear regression, logistic regression and neural networks. Sav et al. [31] proposed a privacy-preserving collaborative training and evaluation of neural networks in the n-party case, which adopts the multiparty lattice-based cryptography to protect the secret inputs and intermediate results. Liu et al. [35] designed a secure transfer learning scheme using fully homomorphic encryption and secret sharing to guarantee user privacy. These schemes are secure against passive adversaries under the assumption of honest-but-curious entities, where these entities are required to follow predefined protocols. However, in real-world applications, it is not realistic to blindly trust that all entities will strictly follow the protocols. For example, there is a risk of a dishonest majority of data-rich domains that are unwilling to share his/her knowledge and deviates from the predefined protocols during the transfer learning process. In such a scenario, the honest-but-curious assumption will no longer hold; thus, such approaches are potentially vulnerable to the setting of dishonest majority [35], [36], [37]. To remove the unrealistic honest-but-curious assumption, the concept of malicious security is introduced, which can prevent the dishonest majority from deviating the predefined protocols [38], [39]. For example, Zheng et al. [34] presented competitive learning for linear models, which is based on SPDZ [15] to implement malicious security, where SPDZ is a practical MPC protocol extended to the dishonest setting. Sharma et al. [40] designed SPDZ-based transfer learning to implement one-way transfer learning with unreliable entities for malicious security. However, the above schemes only transfer knowledge from the source domain to the target domain in one way, which is clearly unsuitable in data-poor domains as all domains lack learned knowledge.

**Limitations:** i) In the scenario of transfer learning, there exists data-rich domains and data-starved domains. Data is unevenly distributed among data domains, with which the performance of CL drops significantly with skewed data. ii) Besides, without the central server, the n-party MPC protocols based on secret sharing are inefficient, as each party requires n − 1 interactions for secret sharing (refer to Section 7.5). iii) Existing privacy-preserving collaborative learning schemes based on MPC rely on the honest-but-curious assumption, where each participant honestly implements secure calculation, and will not collude with others. It is possible to terminate MPC-based protocols when a majority of entities are dishonest and collude with each other. Besides, constructing transfer learning in the collaborative architecture requires complicated activation functions based on MPC, which costs burdensome computational and communication overheads. A comparative summary is presented in Table I.

### III. Preliminaries

We will now briefly describe CNN, transfer learning, and the SPDZ protocol [42] in Sections 3.1 to 3.3.

---

**TABLE I**

| Approach          | Fun$_1$ | Fun$_2$ | Fun$_3$ | Fun$_4$ | Fun$_5$ | Fun$_6$ |
|-------------------|---------|---------|---------|---------|---------|---------|
| [21]              | SVM     | X       | ✓       | X       | X       | X       |
| [24]              | CNN     | X       | ✓       | ✓       | ✓       | ✓       |
| [27]              | CNN     | ✓       | ✓       | ✓       | ✓       | ✓       |
| [41]              | CNN     | ✓       | ✓       | ✓       | ✓       | Semi-honest ✓ |
| [35]              | Deep learning | ✓       | ✓       | ✓       | Semi-honest ✓ |
| [34]              | Linear model | ✓       | ✓       | ✓       | Malicious ✓ |
| [40]              | CNN     | ✓       | ✓       | ✓       | ✓       | Malicious ✓ |
| VerifyTL          | CNN     | ✓       | ✓       | ✓       | ✓       | Malicious ✓ |

Notes: Fun$_1$: Machine learning model; Fun$_2$: Whether supporting collaborative transfer learning; Fun$_3$: Whether supporting multiple parties or not; Fun$_4$: Whether achieving lightweight transmission or not; Fun$_5$: Semi-honest or Malicious security model; Fun$_6$: Whether supporting verification or not.
A. Convolutional Neural Network (CNN)

We adopt a CNN as the base model \( N \) that consists of convolution layers, pooling layers and fully connected layers. Let \( x_0 \) and \( x^L \) = \( y \) respectively be the input and the desired output, where \( L \) is the number of layers and \( x^l \) is the activation map of layer \( l \in \{0, \ldots, L\} \).

- Convolution layer Conv: A Conv layer inputs feature maps \( x^{l-1} \) and adopts the sliding convolutional kernels \( ker \) for feature extraction. Given an input \( x^{l-1} \in \mathbb{R}^{h_l \times w_l \times c_l} \) in 3rd-order tensors (i.e., an array of matrices) with the height \( h_l \), width \( w_l \) and channels \( c_l \). A \( ker \) maps \( x^{l-1} \) to a weighted-sum \( x^l \) as defined in \( x^l = f(W^l x^{l-1}) \), where \( W^l \) is a weight set of the \( l \)-th Conv layer.
- Pooling layer Pool: A Pool layer reduces the data dimensions and trainable parameters in the network, and the neurons in this layer are the outputs of a cluster of neurons at the previous layer.
- Activation function ReLU: The activation function is denoted as a Rectified Linear Unit ReLU(\( x \)) = \( \max(0, x) \), which significantly accelerates the training phase and prevents overfitting.
- Full connection layer Full: A Full layer fully connects all its neurons to each neuron at another layer. Given an input \( x^{l-1} \), the \( l \)-th full connection layer outputs \( x^l = \text{ReLU}(W^l x^{l-1} + b^l) \), where \( b^l \) is a bias term.

B. Transfer Learning

Transfer learning is a machine learning technique that focuses on acquiring knowledge over data domains (i.e., source domains) and repurposing it on a related data domain (i.e., target domain). Generally, transfer learning comprises the following three steps:
- Extract knowledge. A source model is first pre-trained over a source domain, prior to extracting knowledge from training data and repurposing for the target domain.
- Transfer knowledge. A source domain transfers extracted knowledge to a target domain for the construction of a target model.
- Tune target model. The target model needs to be refined over the transferred knowledge and the target domain’s training data for model optimization.

C. SPDZ Protocol

SPDZ protocol, a state-of-the-art MPC protocol, is designed to mitigate malicious adversaries with secret sharing-based MACs, and tolerate corruption of the majority of parties. More specifically, the SPDZ protocol is divided into online and offline phases. The offline phase performs all computationally expensive public-key operations to create and pre-share triples. The online phase only involves lightweight primitives. The advantages of SPDZ are summarized as follows.
- Privacy. Given a plaintext \( x \), it is converted into \( n \) additive shares \( x^{(i)} \in \mathbb{Z}_{2^n} \), where \( x \equiv \sum x^{(i)} \mod 2^n \), \( \kappa \in \{8, 16, 32, 64, 128, \ldots \} \) is the security parameter. The privacy of these shares \( x^{(i)} \) is guaranteed by the additive secret sharing.
- Verifiability. The correctness of all inputs and outputs in SPDZ is verified by the MAC-check mechanism [36] with additive secret shares of MACs over the ring of integers \( \mathbb{Z}_{2^n} \). For \( n \) parties, each party \( D_i \) owns an additive share \( \alpha_i \in \mathbb{Z}_{2^n} \) of the fixed MAC key \( \alpha \), i.e., \( \alpha = \alpha_1 + \alpha_2 + \cdots + \alpha_n \). Here, we define \( x \in \mathbb{Z}_{2^n} \) as \( [\cdot] \)-shared when a party holds a tuple \( (x^{(i)}, \gamma(x)^{(i)}) \), where \( \gamma(x)^{(i)} \) is an additive share of the corresponding MAC value \( \gamma(x) \) as

\[
\gamma(x) = \sum \gamma(x)^{(i)} \mod 2^n = \alpha x. \tag{1}
\]

IV. SYSTEM AND THREAT MODELS, AND DESIGN GOALS

In this section, we will describe the system and threat models, and the design goals.

A. System Model

We consider a scenario, where training data are independently and identically distributed among \( n \) data domains \( D = \{D_1, D_2, \ldots, D_n\} \). To construct a high-performance model, \( D_{i \in [1,n]} \) is willing to transfer individual knowledge from local data with each other. As depicted in Fig. 2, the system model consists of \( n \) data domains including both data-rich domains and data-poor domains, where \( n \) data domains are all interconnected to execute the collaborative transfer learning. There may exist corrupted domains \( D^* \) corrupted by the malicious adversary. The corrupted data domains \( D^* \) may undermine the learning phase by behaving maliciously to learn the privacy of other domains. Even if \( D_{i \in [1,n]} \) wishes to execute collaborative transfer learning, they do not trust each other.

With the majority of corrupted domains, the entities in our system model perform the following steps to protect private information and prevent malicious computations for malicious security. First, \( D_{i \in [1,n]} \) pre-trains a CNN model \( N \) over local data, which is adopted to extract knowledge. The extracted knowledge is denoted as data representations, where is then split into \( n \) shares and broadcasted to other \( n - 1 \) domains (Step 1). During the collaborative transfer learning phase, a transfer unit
realizes secure collaborative transfer learning among \( n \) domains, and verifies the computation process to prevent \( n - 1 \) corrupted domains (Step 2). Then, the transferred representations are returned to other data domains for tuning each local model (Step 3). Steps 1-3 are iterated over multiple data domains until a local CNN model reaches convergence (Step 4).

B. Threat Model

In our threat model, a set of mutually distrustful data domains \( \mathcal{D} \) needs to securely implement an agreed computation protocol over their secret inputs. The protocol should be securely executed to implement malicious security. It indicates that a changing number of corrupted domains cannot learn additional information, or even lead to incorrect results. To faithfully simulate the adversarial setting, the threat model is defined in the presence of malicious adversaries [43].

**Malicious security model.** In the active adversarial setting, a malicious adversary \( A \) can choose any subset from data domains \( \mathcal{D} \) for corruption [42]. \( A \) can corrupt an arbitrary proportion of data domains \( \mathcal{D}^* \subset \mathcal{D} \), even \( |\mathcal{D}^*| = n - 1 \). Generally, \( A \) can execute the following adversarial behaviours.

- \( A \) attempts to infer sensitive information (i.e., local model parameters and intermediate results) of \( \mathcal{D} \).
- \( A \) can influence the corrupted data domains to arbitrarily deviate from the agreed protocol by tampering inputs, launching malicious computations and returning incorrect outputs.

Formally, we define an ideal functionality \( F \) for securely executing secure functions in VerifyTL. The definition of malicious security is formally defined as follows.

**Definition 1.** Let \( F \) be a \( n \)-party functionality, and let \( \prod \) be a \( n \)-party protocol for computing \( F \). In the presence of malicious adversary, the protocol \( \prod \) is secure to compute \( F \) if the ideal model has a non-uniform Probabilistic Polynomial Time (PPT) simulator \( S \) for every non-uniform PPT adversary \( A \) in the real model, such that for the inputs \( I \),

\[
\text{IDEAL}_{F, I, S}(x) \equiv \text{REAL}_{\prod, I, A}(x). \tag{2}
\]

where \( \text{IDEAL}_{F, I, S}(x) \) denotes the joint output of the honest data domains \( \mathcal{D}_i \), and \( S \) from the ideal world execution of \( F \). \( \text{REAL}_{F, I, A}(x) \) denotes the joint output of the honest data domains \( \mathcal{D}_i \) and \( A \) from the ideal world execution of \( \prod \).

**Remark.** We assume that all data domains provide benign inputs to participate in the transfer learning for a high-performance global model for their own benefits. However, due to competitive interests, corrupted data domains may execute malicious computations in the secure transfer process, aiming at destroying the effectiveness of the trained model of honest data domains. Besides, during the local training, we cannot control which plaintexts a data domain could choose to input for secure transfer learning. In the local training over plaintexts, it may involve poisoning attacks [44, 45], i.e., data domains submit carefully-crafted local model parameters before secure outsourcing, which is beyond the scope of this paper. In VerifyTL, we focus on malicious behaviors during secure computations.

C. Design Goals

In VerifyTL, all-but-one data domains can be colluded, where a domain honestly follows the pre-defined protocols, but up to \( n - 1 \) domains may execute malicious computations for negative transfer, and share observed information among them to infer data privacy. To achieve malicious security over multiple mutually-distrusting data domains, VerifyTL is designed to realize the following goals:

- **Privacy preservation.** Extracted local knowledge and intermediate calculation results contain sensitive information on feature representations of local data, with which data distribution and original data can be deduced. For data confidentiality, any data domains should not learn any sensitive information (including the private inputs and intermediate results) from the execution process, even in the presence of \( n - 1 \) corrupted domains.
- **Self-verification.** Considering that data domains are mutually-untrusted, VerifyTL should verify the correctness of execution process. To against adversarial behaviours, \( D_i \) executes a self-verified mechanism to check the integrity of inputs and computations under the setting of dishonest majority.
- **Collaborative transfer.** Training data are unevenly distributed among \( D_{\forall i \in [1,n]} \). To achieve the high-performance model, we design a collaborative transfer with skewed data, in which \( D_{\forall i \in [1,n]} \) exchanges local knowledge for mutual benefits, and tunes its CNN model over the transferred knowledge.

IV. PROPOSED VERIFYTL

Here, we first summarize a technical overview of VerifyTL, and then design the secure and verifiable transfer unit to implement VerifyTL.

A. Technical Overview

The main motivation behind collaborative transfer learning is that a data-rich domain has no profits during transfer learning. Thus, we utilize the collaborative transfer learning to realize two-way transfer, which can contribute multi-domain transferred knowledge to tune CNNs. The core of VerifyTL relies on the following observations:

- Data representations on CNNs contain extracted knowledge of original datasets.
- According to the correlation extent between two domains, each data domain can set different contribution degree to tune its CNN model over transferred representations from other domains.
- A malicious adversary can corrupt any data domains, which leads to privacy leakage and malicious computation over dishonest majority.

In this section, we design the underlying countermeasures based on above observations to achieve VerifyTL:

- Data representations contain sensitive information of training data, and thus it is necessary to provide security guarantees.
is authenticated as a commitment share, such that the fixed MAC key mechanism to prevent malicious behaviours. More details are shown in SPDZ technique that is a state-of-the-art MPC protocol \[36\]. \( D_{\text{src}} \) owns a MAC key share \( \alpha \in \mathbb{Z}_n \) of the fixed MAC key \( \alpha \), where \( \alpha = \sum_{i \in [1,n]} \alpha_i \mod \kappa \). Each secret share \( x^{(i)} \in X^{(i)} \) is authenticated as a commitment share, such that

\[
\sum_{i \in X} X^i = \text{MAC value of } x
\]

Each data domain holds a tuple \((x^{(i)}, \gamma(x)^{(i)})\)

The notation definitions are listed in Table \(2\). The details are described in the following sections.

In each data domain, a degree vector \( \Theta_i \) is used to control different contribution degrees of other domains to implement flexible transfer learning.

Collaborative transfer units, i.e., two-domain unit (cross unit) and multi-domain unit (weave unit) are proposed to provide secure and verifiable computation for the collaborative transfer learning against dishonest behaviours.

Fig. 3 illustrates the main process of VerifyTL. Assume that there are \( n \) data domains, each of which owns a local training dataset. All data domains agree on the same CNN architecture in advance. Each data domain initializes a CNN model on training data, a pooling layer in a CNN model extracts activation maps as the inputs of secure collaborative transfer learning. The SPDZ-based cross and weave units are responsible for maintaining secure and verifiable collaborative transfer. To address the issue of privacy leakage, we propose a \( \Pi_{\text{SPDZ}} \) protocol to transfer protected data representations and intermediate computation results under the settings of two-domain and multi-domain, respectively (Step \(1\)-\(3\)). To avoid the threat of malicious behaviors, we design a self-verifed mechanism to verify the correctness of inputs and computation results (Step \(3\)). The notation definitions are listed in Table \(2\). The details are described in the following sections.

**Table II Note Description**

| Notations | Descriptions |
|-----------|-------------|
| \( x^{(i)} \)-shared | Each data domain holds a tuple \((x^{(i)}, \gamma(x)^{(i)})\) |
| \( D \) | The data domain set with the size of \( n \) |
| \( X^l \) | Activation map of \( l \)-th layer |
| \( N(0, 1) \) | The distribution of zero mean and unit standard deviation |
| \( X^{l} \) | Transferred activation map of \( l \)-th layer |
| \( p \) | Precision |
| \( L \) | Number of layers |
| \( h_l, w_l, c_l \) | the height, width and channels of \( X^l \) |
| \( \Theta \) | Degree vector |
| \( N_{\text{elt}} \) | \( \Theta \) of the \( i \)-th element at a certain location in all maps |
| \( W_i \) | CNN network of the \( i \)-th data domain |
| \( \gamma(x) = \alpha \otimes \gamma(x) \) | MAC value of \( x \) |
| \( \otimes \) | SPDZ multiplication computation over integers |

**B. Construction of VerifyTL**

Here, we design the transfer unit to train networks over data representations transferred among multiple data domains, which implements collaborative transfer learning over activation maps after a pooling layer.

1) **Setup:** Representation Extraction (Step \(1\)): We adopt a CNN as the base model \( \mathcal{N} \) that handles individual training data, which consists of convolution layers, pooling layers and fully connected layers. \( D_{\text{src}} \) pre-trains individual CNN model with the same architecture over its training data. Let \( X^0 \) and \( X^L = y \) respectively be the input and the desired output, where \( L \) is the number of layers and \( X^l \) is the output of a layer \( l \in \{1, \ldots, L\} \).

At the \( l \)-th layer of the network, \( D_{\text{src}} \) utilizes activation maps [46], [47] extracted after a convolutional layer and pooling layer as data representations of local data. The overwhelming majority of modern CNN architectures achieve activation maps through a ReLU, which imposes a hard constraint on the intrinsic structure of the maps. The extracted activation map at the \( l \)-th pooling layer are denoted as a 3rd-order tensor \( A^l = \left[ h_l \right] \times \left[ w_l \right] \times c_l \) with the height \( h_l \), width \( w_l \) and channels \( c_l \).

**Quantization:** In a CNN network, activation maps are normalized with batch normalization \[48\], and the distribution of each activation map is \( N(0, 1) \). However, activation maps cannot be directly encoded and operated in SPDZ libraries and thus require pre-process. We adopt an approximation method \[49\], \[50\] to convert floating-point numbers to fixed-point numbers with a precision \( p \), where \( p \) is the number of bits of approximation precision and the upper bound of the approximation \( [1 - 2^p, -1 + 2^p] \). For example, given a message \( m_1 \) and \( m_2 \), the encoded numbers are defined as \( m_1' = Q(m_1, p) = [m_1, 2^p] \) and \( m_2' = Q(m_2, p) = [m_2, 2^p] \). Specifically, the result of multiplication operation in SPDZ can change the precision of \( m_1' \times m_2' \) to \( 2^p \) while the result of an addition operation \( m_1' + m_2' = (m_1 + m_2)2^p \) in SPDZ is uninfluenced. This is because the SPDZ computation runs over encoded numbers, multiplication operations lead to the expand of precision as \( m_1' \times m_2' = m_12^p \times m_22^p = m_1m_22^{2p} \). Therefore, it is necessary to keep a precision consistent with a truncation \( T(m_1m_2, p) = \lfloor \frac{m_1m_2}{2^p} \rfloor \) after each multiplication operation, where \( T(x, p) = \min(\max([\frac{x}{2^p}], -1 + 2^p), 1 - 2^p) \).

**Encryption and Commitment** (Step \(3\)): \( D_{\text{src}} \) implements secret sharing to encrypt extracted knowledge \( X^l \). A message \( x \in X^l \) is split into \( n \) random shares such that

\[
x^{(n)} = x - \sum_{j=1}^{n-1} r_j, \quad x^{(j)} = r_j,
\]

where \( r_j \in [1,n] \) is a random number, \( \kappa \) is the security parameter.

To authenticate each secret share \( x^{(i)} \), we use the MAC key mechanism to prevent malicious behaviours. More details are shown in SPDZ technique that is a state-of-the-art MPC protocol \[36\]. \( D_{\text{src}} \) owns a MAC key share \( \alpha_i \in \mathbb{Z}_n \) of the fixed MAC key \( \alpha \), where \( \alpha = \sum_{i \in [1,n]} \alpha_i \mod \kappa \). Each secret share \( x^{(i)} \in X^{(i)} \) is authenticated as a commitment share, such as
that
\[ \gamma(x)^{(i)} = \alpha \cdot x^{(i)} \mod \kappa, \]
\[ \text{s.t., } \gamma(x) = \sum_{i \in [1,n]} \alpha_i \cdot x = \sum_{i \in [1,n]} \gamma(x)^{(i)}. \]

Then, \( D_i \in [1,n] \) broadcasts secret shares \( \mathcal{X}^{(i)} \) and their commitments \( \gamma(\mathcal{X})^{(i)} \) to others.

2) Secure and Verifiable Transfer Unit: As demonstrated in Fig. 4, the collaborative transfer consists of representation traverse, collaborative transfer and self-verified mechanism (Step 3-5).

Representation traverse. After receiving others’ representation shares \( \mathcal{X}^{(i)} \) (\( j \in [1,n]/i \)), \( D_i \) traverses these representation shares \( \mathcal{X}^{(i)} \) to obtain \( h_1 \times w_1 \times l_1 \) vectors \( \mathcal{Y}^{(i)} \) such that
\[ \mathcal{Y}^{(i)}_{a,b,c} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}] \]
\[ \text{s.t., } a \in [1,h_1], b \in [1,w_1], c \in [1,l_1], x \in \mathcal{X}^j. \]

The vector \( \mathcal{Y}^{(i)}_{a,b,c} \) is a set of element shares \( x^{(i)} \) from at the same location \( (a,b,c) \) in \( D_i \)’s representation \( \mathcal{X}^{(i)} \).

Collaborative transfer. We present a collaborative unit to transfer knowledge among \( n \) data domains with the presence of a dishonest majority. The key idea is to combine as more knowledge as possible to transfer among \( D_i \). We define a collaborative degree vector \( \Theta = (\theta_1, \theta_2, \ldots, \theta_n) \) according to the size of local data. For example, the collaborative degree of \( D_i \in [1,n] \) is denoted as
\[ \theta_i = \left[ \frac{|X_i|}{|X|} \right] \cdot p, \text{ s.t., } \theta_i \in [0,p), \]
where \( |X_i| \) is the size of local data of \( D_i \), \( |X| \) is the total size of training data. Specifically, the higher values of \( \theta_i \), the more contribution of \( D_i \) is shared in the collaborative transfer.

The collaborative transfer is defined as
\[ \mathcal{I}^{(i)} = \Theta \odot \mathcal{Y}^{(i)}_{a,b,c} = \left[ \theta_1, \ldots, \theta_n \right] \odot \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}^T. \]
The process involves SPDZ-based inner product \( \odot \), the execution process is calculated as
\[ \mathcal{I}^{(i)} \leftarrow \sum_{j \in [1,n]} \theta_j \cdot x_j^{(i)} \mod \kappa. \]

Meanwhile, the commitment of element shares are updated as
\[ \gamma(\mathcal{I})^{(i)} \leftarrow \sum_{j \in [1,n]} \theta_j \cdot \gamma(x_j)^{(i)} \mod \kappa. \]

In this way, \( D_i \) obtains the transferred knowledge \( \mathcal{I}^{(i)} \) and the corresponding commitments \( \gamma(\mathcal{I})^{(i)} \), then broadcasts them to \( D_j \in [1,n]/i \) to open the shares.

Self-verified mechanism. Once receiving secret shares of the transferred knowledge \( \mathcal{I}^{(i)} \) (\( j \in [1,n]/i \)), \( D_i \) can open the final transferred knowledge to tune local model \( \mathcal{N}^{\text{ei}}. \) However, with the existence of dishonest majority, \( A \) can attempt to corrupt the inputs and SPDZ-based computation by replacing inconsistent attacker-chosen inputs and implementing incorrect computations. It is unconfirmed that these shares \( \mathcal{I}^{(i)} \) and the opened value \( \mathcal{I} \) are consistent. Before returning an opened value, it is necessary to verify these shares and opened value with commitments and MAC keys. The self-verified mechanism is executed as follows.

- \( D_i \) agrees on a \( n \)-dimensional random vector \( Y \leftarrow \mathbb{Z}_n^\kappa. \)
- \( D_i \in [1,n] \) opens individual transferred representations such that
\[ \mathcal{I} \leftarrow \sum_{i \in [1,n]} \mathcal{I}^{(i)} \mod \kappa. \]
- To verify the integrity of \( \mathcal{I} \leftarrow \mathbb{Z}^{h_1 \times w_1 \times c_1}, D_i \in [1,n] \) computes a tensor \( M_i \leftarrow \mathbb{Z}^{h_1 \times w_1 \times c_1} \) of proofs by using the local MAC key share \( \alpha_i \) and commitments \( \gamma(\mathcal{I})^{(i)} \). The proof \( m_{a,b,c}^{a,b,c} \in M_i \) is computed as
\[ m_{a,b,c}^{a,b,c} = r \cdot (\alpha_i \cdot \mathcal{I}^{(i)} - \gamma(\mathcal{I}^{(i)}) \mod \kappa. \]
\[ \text{s.t. } a \in [1,h_1], b \in [1,w_1], c \in [1,l_1], \]
where \( r \in \mathbb{Z}_n \) is a random number, \( \mathcal{I}^{a,b,c} \in \mathcal{I}^{(i)} \), \( \gamma(\mathcal{I}^{a,b,c})^{(i)} \in \gamma(\mathcal{I})^{(i)} \).
- \( D_i \in [1,n] \) broadcasts the proof \( M_i \), then executes the following computation over the received proofs such that
\[ m_{a,b,c}^{a,b,c} = \sum_{i \in [1,n]} m_{a,b,c}^{a,b,c} \mod \kappa \]
\[ \text{s.t. } a \in [1,h_1], b \in [1,w_1], c \in [1,l_1], m_{a,b,c}^{a,b,c} \in M_i. \]
- If \( D_i \in [1,n] \) honestly follows the predefined protocol, then the proof is obtained as
\[ m_{a,b,c}^{a,b,c} \Rightarrow r \cdot \sum_{i \in [1,n]} (\alpha_i \cdot \mathcal{I}^{(i)} - \gamma(\mathcal{I}^{(i)})) \mod \kappa \]
\[ \Rightarrow r \cdot (\sum_{i \in [1,n]} \alpha_i \cdot \mathcal{I}^{(i)} - \gamma(\mathcal{I}^{(i)})) \mod \kappa \]
\[ \Rightarrow r \cdot (\alpha \cdot \mathcal{I}^{(i)} - \gamma(\mathcal{I}^{(i)})) \mod \kappa \]
\[ \Rightarrow r \cdot (\mathcal{I}^{(i)} - \mathcal{I}^{(i)}) = 0. \]
Consequently, $D_{\forall i \in [1,n]}$ checks the integrity as

$$\text{Verify} = \begin{cases} \text{Accept}, & \text{if } m = 0, \\ \text{Abort}, & \text{otherwise}. \end{cases}$$

If the self-verified mechanism fails, then the computation process aborts. Otherwise, all $D_{\forall i \in [1,n]}$ can obtain the transferred knowledge $\overline{K}$, with which the local model $N_{\forall t,i}$ is fine-tuned with Stochastic Gradient Descent (SGD) method to minimize the loss function $\mathcal{L}(N_{\forall t,i})$.

3) **Tune model:** The objective of weave transfer learning is to minimize joint losses $\mathcal{L}(\overline{W})$ over $n$ domains, which is defined as

$$\arg\min_{\overline{W}} \mathcal{L}(\overline{W}) = \sum_{i=1}^{n} \mathcal{L}(W_i) \quad \text{s.t.} \quad W_i \in N_{\forall t,i},$$

$D_{\forall i \in [1,n]}$ adopts forward and backward propagation methods to fine-tune $N_{\forall t,i}$. The specified process is shown in [51].

**Forward pass.** Upon obtaining the transferred $\overline{X}_t^{l,n}$, each domain $D_i$ implements $\overline{X}_t^{l+1} = f(W_t^l \overline{X}_t^{l,n})$ and outputs prediction $\overline{X}^L$ after $L$ layers.

**Backward pass.** For minimizing loss function $\mathcal{L}$ measuring the difference between predictions $\overline{X}^L$ and ground-truth labels $y$, the objective function can be optimized by the back-propagation method. During the tune process of trained networks, the global object is divided into local optimization over a single domain $D_i$. The gradients of data representations are back-propagated, the derivative of loss function $\mathcal{L}$ in a transfer unit are defined as

$$\frac{\partial \mathcal{L}}{\partial \overline{X}_t^{l,i}} = \left[ \frac{\partial \mathcal{L}}{\partial \overline{X}_t^{l,i}} \right] \left[ \begin{array}{c} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \frac{\partial \mathcal{L}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_n} \end{array} \right].$$

To construct the collaborative transfer learning over multiple domains, $D_i$ is required to implement local optimization by building its CNN model $N_{\forall t,i}$.

---

**VI. SECURITY ANALYSIS**

In this section, we demonstrate the security of VerifyTL using the standard real/ideal world paradigm. We first give the security definition, and evaluate whether VerifyTL satisfies the privacy requirements in Section 4.2 under the following security definitions.

**A. Security Definition**

We follow the real/ideal world paradigm formalized in [42], [52], the security of a protocol $\prod$ is defined as the indistinguishability between the real-model executed by an adversary $\mathcal{A}$ and an ideal functionality with a simulator $\mathcal{S}$. For each of the constructions, we provide the simulation proof for the case of dishonest majority of $D^*$ (i.e., corrupted domains). Here, the simulator $\mathcal{S}$ and the adversary $\mathcal{A}$ are constructed to play the role of $D_i$ by interacting with the other domain $D_j$.

**Real-world model REAL:** The $n$-party protocol $\prod$ is executed over data domains $D$. Each data domain $D_i$ provides the public inputs $\text{Input}^{\text{p}}_i$ and secret inputs $\text{Input}^{\text{s}}_i$, then the public output $\text{Output}^{\text{p}}_i$ and secret output $\text{Output}^{\text{s}}_i$ are produced with random masks $r_i \in \mathbb{Z}_n$ ($i \in [1,n]$).

**Ideal-world model IDEAL:** The functionality $\mathcal{F}$ is executed as a probabilistic $n$-party function in Probabilistic Polynomial Time (PPT), which is defined as $\mathcal{F}(\kappa, \text{Input}^\ast_1, \text{Input}^\ast_2, \ldots, \text{Input}^\ast_n, \text{Input}^\ast_p, \text{r})$, and $\text{r}$ is a set of random masks. In IDEAL, all domains send individual inputs to a trusted third party $T$ that executes $\mathcal{F}$ over these inputs and returns ($\text{Output}^{\text{p}}_i, \text{Output}^{\text{s}}_i$) to $D_i$.

**B. Security Proofs**

Based on above security definition, we theoretically prove that the security of VerifyTL scheme is computationally indistinguishable in following theorems.

1) **Encryption and Commitment Protocol:** The ideal functionality $\mathcal{F}_{\text{Enc}}$ executing encryption protocol $\prod_{\text{Enc}}$ is demonstrated in Fig. 5.

**Theorem 1.** Let $\prod_{\text{Enc}}$ be the encryption and commitment protocol that securely realizes the functionality $\mathcal{F}_{\text{Enc}}$ in the presence of the malicious adversary.

**Proof.** We present the construction for the case for corrupted domains $D^*$, which is demonstrated in Fig. 6.

We denote $\text{IDEAL}_{\Sigma_{\text{Enc}}}$ and $\text{REAL}_{\Sigma_{\text{Enc}}}$ as the views of $D_i$ and $D^*$, respectively. $\Sigma_{\text{Enc}}$ extracts the local activation maps after the building of a CNN model. These activation maps are adopted for interactive data representations.

During the process of $\prod_{\text{Enc}}$, each domain implements secret sharing over individual extracted representations $\mathcal{X}$, and then these shares are broadcasted to other domains. Since the values of these shares are masked with $n-1$ random numbers $r^k \in \mathbb{Z}_n$, e.g., a value $x = \sum_{k=1}^n x^{(k)} \mod \kappa$, where $x^{(k)} = x - r^k \mod \kappa$, $x^{(n)} = x - \sum_{k=1}^{n-1} x^{(k)} \mod \kappa$, the actual values of the inputs cannot be recovered with the protection of random mask $r^k$. It is obvious that the views of the adversary $\mathcal{A}_{\text{Enc}}$ are indistinguishable in both real and ideal models, as represented
Protocol $\prod_{\text{Enc}}$

**Adversary $A_{\text{Enc}}$**

- $A_{\text{Enc}}$ receives $\alpha$ from $D^*$.
- $A_{\text{Enc}}^{D_i}$ selects $n-1$ random values $x_i, r_k \in [1, n]$, and computes
  $$x_n = x - \sum_{k=1}^{n-1} r_k, x(k) = r_k.$$
- $A_{\text{Enc}}$ commits each random number with public MAC key $\alpha$ and local MAC key $\alpha^*$, which is computed as
  $$\gamma(x(k)) = \alpha \cdot x(k) \mod \kappa$$
  or
  $$\gamma(x(k)) = \alpha^* \cdot x(k) \mod \kappa.$$
- $A_{\text{Enc}}$ outputs $(X_i(k), \gamma(X_i(k)), k \in [1, n])$.

**2) Secure and Collaborative Transfer Protocol:** The ideal functionality $F_{\text{Trans}}$ executing the secure and collaborative transfer protocol $\prod_{\text{Trans}}$ is demonstrated as Fig. 7.

**Theorem 2.** Let $\prod_{\text{Trans}}$ be a secure and collaborative transfer protocol that securely computes a functionality $\sum_{\text{Trans}}$ among multiple data domains (i.e., honest domains $D_i$ and corrupted domains $D^*$) in the presence of the malicious adversary.

**Proof.** We demonstrate the construction for the case for corrupted domains $D^*$, which is presented in Fig. 8.

We denote $\text{IDEAL}_{\text{Enc}}^{\text{S}_{D_i}}(\alpha_i, X_i, X_i^k, k \in [1, n]) \equiv \text{REAL}_{\text{Enc}}(\alpha, X_i, X_i^k)$.

During these phases, $S_{\text{Trans}}$ simulates several times of SPDZ multiplication "$\circ$", where all inputs and intermediate parameters are masked with statistically indistinguishable uniformly random numbers $r \in Z_\kappa$. Thus, the real distributions of these inputs and intermediate parameters in the simulator $S_{\text{Trans}}$ are statistically indistinguishable from the view of $\text{REAL}$. All the inputs and intermediate parameters are $[-]$-shared before being broadcasted to the other domain. The privacy of the inputs and intermediate results can be protected, as the views of $\text{REAL}_{\text{Trans}}$ are indistinguishable from the view of $\text{REAL}$. For each input share $[\alpha_i]$, $S_{\text{Trans}}$ generates MAC shares $\gamma(\alpha_i)$ to verify whether secret shares and MAC shares satisfy the invariant $\alpha_i \cdot \gamma(\alpha_i) = \sum_{i=1}^{n} \gamma(\alpha_i)$.

Hence, $A_{\text{Trans}}$ cannot commit forged intermediate results $\sum_k^i$ without $A$.

$S_{\text{Trans}}$ opens the final result over shares with the self-verified mechanism. $S_{\text{Trans}}$ receives the global MAC key $\alpha$, then splits $\alpha$ into multiple random shares, each of which is sent to $D_i$. For each input share $[\alpha_i]$, $S_{\text{Trans}}$ generates MAC shares $\gamma(\alpha_i)$ to verify whether secret shares and MAC shares satisfy the invariant $\alpha_i \cdot \gamma(\alpha_i) = \sum_{i=1}^{n} \gamma(\alpha_i)$. $A_{\text{Trans}}$ also provides additive shares and MAC shares for the verification of a final result. If the validation fails, then $D_i$ aborts the computation. Otherwise, $S_{\text{Trans}}$ obtains the final result for the adversary $A_{\text{Trans}}$.

However, all honest domains execute the self-verified mechanism over shares $x(i)$ of the final result and corresponding MAC.
key shares \(\alpha_i\), where \(x = \sum x^{(i)} \mod 2^\kappa\), \(\alpha = \sum \alpha^{(i)}\) and \(\gamma(x)_i = \alpha_i \cdot x^{(i)}\). Each data domain verifies the opened result \(x\) by judging if this is true:

\[
\gamma(x) = \sum_i \gamma(x)_i = \sum_i \alpha^{(i)} x^{(i)} \mod \kappa = \alpha x.
\]

Since \(S_{\text{Trans}}^D\) can not only change the content of secret shares, but also modify an additive share of the corresponding MAC value \(\gamma(x)\), which is computed by using additive shares of MAC key \(\alpha_i\) on each data domain \(D_i\). \(\prod_{\text{Trans}}\) can enable each domain to correctly compute in the transfer unit. If the inputs and opened values don’t pass the self-verified mechanism, then all domains will abort computation. Otherwise, the final computation results are returned to the weave unit. Therefore, the simulator \(S_{\text{Trans}}^D\) has completed the simulation process, where \(S_{\text{Trans}}^D\) successfully simulates IDEAL without leaking original values of activation maps \(X\) for all data domains \(D_i\) in \(D\). Thus, it indicates the indistinguishability between this Real and real model REAL based on above analysis, which is represented as \(\text{IDEAL}_{\prod_{\text{Trans}}} S_{\text{Trans}}^D (X, \gamma(X), m_i, \alpha_i, D_i) \equiv \text{REAL}_{\prod_{\text{Trans}}} (X, \gamma(X), m_i, \alpha, D_i)\).

Based on the above analysis, we conclude that our protocol \(\prod_{\text{Trans}}\) can securely implement under the setting of malicious adversary, which satisfies privacy requirements in VerifyTL.

VII. PERFORMANCE EVALUATION

In this section, we discuss experimental setup and evaluate VerifyTL on real-world datasets, and we compare VerifyTL with existing solutions.

A. Experimental Setup

The experiments were executed in Java and were implemented on two six-core 2.80GHz machines with Inter i5-8400H processor, 16GB RAM, running Ubuntu, and VerifyTL is evaluated in parallel. The LAN setting is implemented with two machines hosted in the same region with an average bandwidth of 974 Mbps and an average latency of 3.1 ms. We begin the experiments by introducing training datasets. The communication among different data domains relies on TCP with authenticated channels (through TLS).

Network architecture. We evaluated our methods over four different image classification tasks trained in the collaborative learning architecture. The details are shown in Table III.

- We use NetworkI with for USPS data [53], which contains one convolution layer Conv, one pooling layer Pool, and one full connection layer Full.
- We adopt NetworkII (i.e., LeNet-5 [51]) for MNIST data [54], which consists of \(L=7\) layers such as two Conv layers, two Pool layers, and one Full layer.
- We employ NetworkIII with three Conv layers, two Pool layers, and one Full layer for Fashion MNIST data [55].
- We adopt NetworkIV (i.e., AlexNet [56]) with five Conv layers, three pooling layers Pool and three Full layers for CIFAR-10 data [57].

Parameters. We set up the parameters in VerifyTL with a security parameter \(\kappa = 2^{128}\), precision \(p = 2^8\) and the size of data domains \(n\) varies in the range \([2,10]\). All domains adopt the same CNN model \(Net\), where batch_size=128, learning_rate = 0.01, dropout=0.8.

B. Effectiveness

We tested our evaluation over four kinds of network architectures (NetworkI, II, III, IV). We adopt the 10-fold cross validation technique for CNN accuracy. There are 10 data domains (i.e., \(n=10\)), the size of training samples on each domain is 1K, the secure and verifiable transfer units are adopted after all pooling layers to guarantee accuracy. Figs. 9(a-d) describe the variation of the test accuracy with a secure and collaborative transfer unit, which is adopted after each Pool layer.

Accuracy improvement. The original learning is implemented on a single data domain without any transfer unit. We notice that the accuracies on all NetworkI-IV of VerifyTL have the significant improvement than those without secure and verifiable transfer units. This is because that more data representations are transferred among data domains with the increase of \(n\). Thus, more knowledge can be involved to improve accuracy on each data domain. Besides, the above experiments demonstrate that VerifyTL reaches the significant accuracy improvement compared with the original learning without transfer units. It is
consistent with our scheme that VerifyTL is applicable to different kinds of CNN architectures. For Network I, VerifyTL adopts a secure and verifiable transfer unit on USPS data after Pool2. As depicted in Fig. 9(a), VerifyTL achieves the test accuracy of 81.41%, while the original learning reaches the test accuracy of 61.71% after 120 iterations. For Network II, the original learning reaches the test accuracy of 70.40% on MNIST data after 35 iterations. VerifyTL adopts two secure and verifiable transfer units after Pool2 & Pool4. As shown in Fig. 9(b), VerifyTL has a significant accuracy improvement by up to 20.59%. For Network III, VerifyTL adopts two secure and verifiable transfer unit after Pool2 & Pool4. As described in Fig. 9(c), VerifyTL achieves the test accuracy of 92.21% on Fashion MNIST data. While the original learning is hard to converge with limited accuracy, whose the test accuracy of 57.56% after 150 iterations. For Network IV, VerifyTL uses three secure and verifiable transfer units after Pool2, Pool4 & Pool8. As described in Fig. 9(d), VerifyTL converges the test accuracy of 89.67% on CIFAR-10 data. While the original learning reaches the test accuracy of 21.42% after 150 iterations.

**Plaintext comparison.** Figs. 9(a-d) show the accuracy comparison between VerifyTL and the proposed scheme over plaintexts. For Network I, VerifyTL achieves the test accuracy of 81.41%, while VerifyTL implemented with plaintexts reaches the test accuracy of 80.79% after 120 iterations. For Network II, VerifyTL reaches the test accuracy of 90.89%, while VerifyTL implemented with plaintexts achieves 91.11%. For Network III, VerifyTL converges a test accuracy of 92.21%, which has an accuracy difference of −0.76% compared with VerifyTL implemented with plaintexts after 150 iterations. For Network IV, VerifyTL has the accuracy difference of −0.62% compared with VerifyTL implemented with plaintexts after 200 iterations. We note that the accuracy of VerifyTL is similar to that of plaintexts with negligible accuracy difference. This is because VerifyTL enables the privacy and verification over secret shares by adopting the approximation method to convert a rational number to the integer field, which may incur computation errors.

### C. Efficiency

1) **Theoretical Analysis:** We analyze the computational complexity and communication complexity of a secure and verifiable transfer unit.

- **Computational complexity.** In a secure and verifiable transfer unit, computational complexity of the collaborative transfer phase depends on the size of vectors $V_i$ and $Θ$. In the collaborative transfer phase, $D_i ∈ [1, n]$ involves one time SPDZ-based inner product $⊙$ that calls $n$ multiplication operation, which costs $O(n)$ in a SPDZ multiplication with linear operations. As the size of transferred activation maps is $A^l ← \mathbb{R}^{h_l \times w_l \times c_l}$, it involves $h_l \times w_l \times c_l$ times collaborative transfer, thus a secure and verifiable transfer unit costs $O(n * h_l * w_l * c_l)$ time for each domain $D_i ∈ [1, n]$.

#### Table III

| Dataset     | Network Architecture |
|-------------|----------------------|
| USPS        | Conv2 $28 \times 28 \times 1$ $→$ Pool2 $24 \times 24 \times 20$ $→$ Pool3 $12 \times 12 \times 10$ $→$ Pool4 $10 \times 1 \times 1$ |
| MNIST       | Conv2 $28 \times 28 \times 3$ $→$ Pool2 $24 \times 24 \times 6$ $→$ Pool3 $12 \times 12 \times 6$ $→$ Pool4 $8 \times 8 \times 12$ $→$ Pool5 $4 \times 4 \times 1$ $→$ Pool6 $1 \times 1 \times 1$ |
| Fashion MNIST | Conv2 $32 \times 32 \times 3$ $→$ Pool2 $28 \times 28 \times 6$ $→$ Pool3 $14 \times 14 \times 6$ $→$ Pool4 $10 \times 10 \times 16$ $→$ Pool5 $8 \times 8 \times 10$ $→$ Pool6 $4 \times 4 \times 1$ $→$ Pool7 $1 \times 1 \times 1$ |
| CIFAR-10    | Conv2 $227 \times 227 \times 3$ $→$ Pool2 $255 \times 255 \times 3$ $→$ Pool3 $13 \times 13 \times 256$ $→$ Pool4 $13 \times 13 \times 384$ $→$ Pool5 $13 \times 13 \times 384$ $→$ Pool6 $6 \times 6 \times 256$ $→$ Pool7 $4 \times 4 \times 256$ $→$ Pool8 $4 \times 4 \times 256$ $→$ Pool9 $1 \times 1 \times 1$ |

Notes. *Full* denotes the fully connection layer, *Conv* denotes the convolution layer, and *Pool* denotes the pooling layer. The size of the output and unit on each layer is denoted as $h_l \times w_l \times c_l$. The kernel size is the size of a convolution kernel on a *Conv* layer. The kernel size is the size of a pooling unit on a *Pool* layer.

![Fig. 9. Accuracy of VerifyTL.](image-url)
TABLE IV
RUNNING TIME AND COMMUNICATION OVERHEAD

| Transfer Unit | Network I | Network II | Network III | Network IV |
|---------------|-----------|------------|-------------|------------|
|               | After Pool₂ | After Pool₂ | After Pool₄ | After Pool₂ | After Pool₂ | After Pool₄ | After Pool₄ | After Pool₈ |
| Running time (s) | 11.86 | 3.56 | 0.79 | 4.84 | 1.64 | 288.13 | 178.17 | 37.96 |
| Comm. overhead(MB) | 1.258 | 3.615 | 0.803 | 4.921 | 1.673 | 292.833 | 181.012 | 38.495 |

**Notes.** The batch_size=50, and n = 2. “Commu.” is the abbreviation for “Communication”.

- **Communication complexity.** A secure and verifiable collaborative transfer unit has communication complexity \(\mathcal{O}(n^2(h_l \times w_l \times c_l))\): Since the communication complexity relies on the size of activation maps and the number of data domains, each data domain communicates \(\mathcal{O}(n(h_l \times w_l \times c_l))\) data items, where the size of a transferred activation map is \(h_l \times w_l \times c_l\).

2) **Experimental Analysis:** Table IV demonstrates the execution time and communication overheads of the following sections of VerifyTL, which is an average over 100 trials.

Fig. 10(a) shows the running time of the secure and verifiable collaborative transfer with the semi-honest model and malicious security model, respectively. We observe that the running time increases with the growth of data domains \(D\). Since a bigger size of vector \(\mathcal{X}\)l and \(\Theta\) is involved, more times of secure computations are executed. Also, the secure and verifiable collaborative transfer unit costs more overhead in our malicious security model to guarantee the verification of computation results. This is expected, as verification process is required to spend execution time. It creates a tradeoff between security and efficiency as a malicious model provides a higher security level than a semi-honest. When \(n = 10\), batch_size=50, the self-verified mechanism costs 4.76 s. The increase of verification time is within an acceptable limit for implementing the malicious security model. Besides, we discover that the execution time of a transfer unit is affected by the size of inputs. The running time of a transfer unit after Pool₂ is more than that of it after Pool₄. The reason is that the input of a transfer unit after a Pool₂ layer is \(\mathcal{X}^{Pool₂}\) with the size of \(12 \times 12 \times 6\), while the input of a transfer unit after a Pool₄ layer is \(\mathcal{X}^{Pool₄}\) with the size of \(4 \times 4 \times 12\) thus more elements are involved in a transfer unit after Pool₄ for SPDZ computation. When \(n = 10\), a secure and verifiable transfer unit after Pool₂ costs 5.93 s, while a secure and verifiable transfer unit after Pool₄ costs 2.14 s.

Fig. 10(b) depicts the running time of VerifyTL with four different networks. NetworkI has the simplest architecture, while NetworkIV has the most sophisticated one. We notice the running time on VerifyTL with NetworkIV is greater than that of the proposed scheme on NetworkI – III. After 10 iterations, VerifyTL with Network costs 2.79 min, VerifyTL with NetworkII costs 0.53 min, VerifyTL with NetworkIII costs 0.70 min, while VerifyTL with NetworkIV costs 42.8 min. The reason is the large amount of data required to be processed for CNN training and a secure and verifiable transfer unit. The more complex the network, the more data needs to be processed.

Fig. 10(c) depicts the influence of the size of data domains on the training time. We notice that the training time of VerifyTL is increased as the growth of data domains. It represents that more activation maps are transferred to tune more local CNN model with the increase of \(D\). After 150 training iterations, the training time is 19.69 min with \(n = 5\), the training time is 17.98 min with \(n = 4\), the training time is 15.09 min with \(n = 3\), while the training time is 11.19 min with \(n = 2\).

In Fig. 10(d), we notice that the running time of VerifyTL is larger than that of the proposed scheme over plaintexts, where \(n = 2\). After 150 iterations, the training time is 12.87 min and that of plaintexts is 7.99 min. This is because VerifyTL implements a transfer unit over secret shares with SPDZ computation to guarantee privacy and verification.

D. **Comparative Evaluations**

We select the transfer learning baselines for comparisons, which only supports the transfer learning between two data domains, i.e., \(n = 2\). We set the skewed data setting, in which the size of the training data of \(D₁\) is 2K, while the size of training data of \(D₂\) is 6K.
Secure transfer learning [35] adopts logistic regression for MNIST and Fashion MNIST data, which is based on additive homomorphic encryption and secret sharing with the 80-bit security level, where batch size = 50, learning rate = 0.005, dropout = 0.8.

Cross-stitch [27] is a transfer learning scheme without privacy preservation. It adopts NetworkII for transfer learning, where batch size = 50, learning rate = 0.02, dropout = 0.8. We select the following baselines for comparisons, where n = 5. We set the skewed data setting, in which the size of training data of D1, D2, D3, D4 is 1K, while the size of training data of D5 is 6K.

Original learning adopts NetworkII without transfer learning units, which is implemented on a single data domain.

Homomorphic-encryption collaborative learning [5] is a distributed architecture with a server, which adopts NetworkII as the collaborative model architecture over MNIST data and Fashion MNIST data, respectively. The batch size = 50, learning rate = 0.02, dropout = 0.8. The server performs secure aggregation based on the Paillier cryptosystem with the 80-bit security level. It requires only the weighted averages of the local-updated gradients computed by SGD.

Helen [34] is a malicious secure competitive learning for linear models based on SPDZ. Helen adopts the logistic regression classifier as the global model over MNIST and Fashion MNIST data. The batch size = 50, learning rate = 0.02, dropout = 0.8.

Based on Table V, we conclude that VerifyTL provides a stronger security model and achieves outstanding accuracy results that can rival with other approaches. The accuracy of the original learning with a NetworkII model without any collaborative transfer units is compared with VerifyTL. VerifyTL implements a significant accuracy improvement and provides privacy and verification with an acceptable training time. Besides, VerifyTL performs better than federated learning [4] in both security, efficiency and effectiveness. In federated learning [4], a data domain is required to securely outsource trainable parameters at each layer to a semi-honest central server. There are total 3,646 parameters in a NetworkI model, which costs huge computation overhead for secure outsourcing. The reason for the computation overhead is that [4] is based on Paillier cryptosystem, which involves more expensive exponent arithmetic to guarantee the privacy by encrypting the large size of transmitted CNN parameters during each training epoch. Unfortunately, federated learning cannot support malicious security, of which the correctness of behaviours among distributed data domains and the central server cannot be guaranteed. Once a malicious adversary corrupts n – 1 data domains, it will lead to incorrect training to undermine the accuracy. The computational overhead of Helen [34] relies on the feature dimension and data size. Helen is limited with a condition, where the feature dimension d must be much smaller than the data size of each domain m, i.e., d ≪ m. In Helen, the feature dimension of both MNIST and Fashion MNIST data is d = 28 * 28 = 784, the data size of each domain is m = 1K, which cannot satisfy the condition d ≪ m. Hence, the computation overhead of Helen is significantly influenced by d, which leads to a heavy cost for encryption, verification, and secure learning.

Also, compared with transfer learning schemes [27], [35], VerifyTL maintains outstanding accuracy and extends transfer learning from the two-domain setting to the multi-domain setting with strong privacy preservation. The transfer learning scheme [27] runs over plaintexts without privacy preservation. The secure transfer scheme [35] is based on additive homomorphic encryption and secret sharing using Beaver’s triples into two-party secure multiplication, where the generation of Beaver’s triples has a heavy cost. Besides, the computational overhead of the scheme [35] relies on the feature dimension and data size. For the high dimensional data (MNIST, Fashion MNIST), the scheme [35] has a huge computational overhead for secure transfer learning.

### VIII. Conclusion

In this paper, we proposed a secure and verifiable collaborative transfer learning (VerifyTL) scheme. The scheme facilitates the collaborative transfer over extracted knowledge among multiple data domains in a strong privacy-preserving manner, and allows verification against dishonest majority for implementing malicious security. We mathematically proved the security of VerifyTL, as well as evaluating its performance using four real-world datasets USPS, MNIST, Fashion MNIST and CIFAR-10, i.e., the performance gains with VerifyTL up to > 60% compared with original learning implemented CNN training on a single domain.

| Data size | Method     | Accuracy | Training time | Threat model  | Reason for the computation overhead |
|-----------|------------|----------|---------------|---------------|-------------------------------------|
| n = 2     | Cross-stitch [27] | 87.4%    | 0.14 h        | Network II    | Secure transfer scheme without privacy preservation. It adopts NetworkI for transfer learning, where batch size = 50, learning rate = 0.005, dropout = 0.8. |
|           | Secure transfer | 76.4%    | 9.51 h        | Logistic regression | Semi-honest security |
|           | VerifyTL | 91.34%    | 0.21 h        | Network II | Malicious security |
| n = 5     | Original learning | 74.6%    | 0.32 h        | Network II | — |
|           | federated learning [4] | 92.3%    | 14.8 h        | Network II | Semi-honest security |
|           | Helen [34] | 75.3%    | 7.8 h         | Logistic regression | Malicious security |
|           | VerifyTL | 98.2%    | 1.31 h        | Network II | Malicious security |

Notes: 300 training iterations.
REFERENCES

[1] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face recognition: A convolutional neural-network approach,” IEEE Trans. Neural Netw., vol. 8, no. 1, pp. 98–113, Jan. 1997.

[2] E. Unice, “General data protection regulation,” 2018. [Online]. Available: https://gdpr-info.eu/

[3] S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[4] S. Santner, S. Wiedemann, K.-R. Muller, and W. Samek, “Robust and communication-efficient federated learning from non-tid data,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[5] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, “Privacy-preserving deep learning via additively homomorphic encryption,” IEEE Trans. Inf. Forensics Secur., vol. 13, no. 5, pp. 1333–1345, May 2018.

[6] G. Xu, G. Li, S. Guo, T. Zhang, and H. Li, “Privacy-preserving decentralized deep learning with multiparty homomorphic encryption,” CoRR, 2022, arXiv:2207.04604.

[7] L. Gui, R. Xu, Q. Lu, J. Du, and Y. Zhou, “Negative transfer detection in transductive transfer learning,” Int. J. Mach. Learn. Cybern., vol. 9, no. 2, pp. 185–197, 2018.

[8] T. Gu, B. Dolan-Gavitt, and S. Garg, “Communication: Identiﬁying vulnerabilities in the machine learning model supply chain,” 2017, arXiv:1708.06733.

[9] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in Proc. IEEE Symp. Secur. Privacy, IEEE, 2017, pp. 3–18.

[10] H. Oh and Y. Lee, “Exploring image reconstruction attack in deep learning computation oﬄoading,” in Proc. Int. Workshop Deep Learn. Mobile Syst. Appl., ACM, 2019, pp. 19–24.

[11] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, “Beyond inferring class representatives: User-level privacy leakage from federated learning,” in Proc. IEEE Conf. Comput. Commun., IEEE, 2019, pp. 2512–2520.

[12] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein, “Senate: A maliciously-secure MPC platform for collaborative analytics,” in Proc. USENIX Secur. Symp., 2021, pp. 2129–2146.

[13] A. Daliskov, D. Escudero, and M. Keller, “Fantastic four: Honest-majority four-party secure computation with malicious security,” in Proc. USENIX Secur. Symp., 2021, pp. 2183–2200.

[14] M. Z. A. Bhiyani and J. Wu, “Collusion attack detection in networked systems,” in Proc. IEEE Cyber Sci. Technol. Congr., IEEE, 2016, pp. 286–293.

[15] T. Araki et al., “Generalizing the SPDZ compiler for other protocols,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., ACM, 2018, pp. 880–895.

[16] Q. Li, B. He, and D. Song, “Model-contrastive federated learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 10713–10722.

[17] M. Kamp, J. Fischer, and J. Vreeken, “Federated learning from small worked systems,” in Proc. IEEE Symp. Secur. Privacy, IEEE, 2018, pp. 35–52.

[18] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, “Helen: Maliciously secure cooperative learning for linear models,” in Proc. IEEE Symp. Secur. Privacy, IEEE, 2019, pp. 724–738.

[19] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, “A secure federated transfer learning framework,” IEEE Intell. Syst., vol. 35, no. 4, pp. 70–82, June-Aug., 2020.

[20] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, “Practical covertly secure MPC for dishonest majority–or: Breaking the SPDZ limits,” in Proc. Eur. Symp. Res. Comput. Secur., Springer, 2013, pp. 1–18.

[21] Y. Chen, J. Wang, C. Yu, W. Gao, and X. Qin, “FedHealth: A federated transfer learning framework for wearable healthcare,” 2019, arXiv:1907.09473.

[22] Y. Aumann and Y. Lindell, “Security against covert adversaries: Efﬁcient protocols for realistic adversaries,” in Proc. Theory Cryptogr. Conf., Springer, 2007, pp. 137–156.

[23] C. Hazay and Y. Lindell, “Efﬁcient protocols for set intersection and pattern matching with security against malicious and coalvert adversaries,” in Proc. Theory Cryptogr. Conf., Springer, 2008, pp. 155–175.

[24] S. Sharma, X. Chaoping, Y. Liu, and Y. Kang, “Secure and efﬁcient federated transfer learning,” 2019, arXiv:1910.13271.

[25] M. Salem, S. Taheri, and J.-S. Yuan, “Utilizing transfer learning and homomorphic encryption in a privacy preserving and secure biometric recognition system,” Computers, vol. 8, no. 1, 2019, Art. no. 3.

[26] C. Hazay and Y. Lindell, “A note on the relation between the deﬁnitions of security for semi-honest and malicious adversaries,” IACR Cryptol. EPRINT Arch., vol. 2010, pp. 550, 2011.

[27] M. Aumann and Y. Lindell, “Security against covert adversaries: Efﬁcient protocols for realistic adversaries,” J. Cryptol., vol. 23, no. 2, pp. 281–343, 2010.

[28] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, “Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,” in Proc. IEEE Symp. Secur. Privacy, IEEE, 2018, pp. 99–35.

[29] O. Sicu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, “Towards rich feature discovery with class activation maps augmentation for person re-identiﬁcation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 1389–1398.

[30] Y. Zhu, Y. Zhou, H. Xu, Q. Ye, D. Doerrmann, and J. Jiao, “Learning instance activation maps for weakly supervised instance segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 3116–3125.

[31] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015, arXiv:1502.03167.

[32] N. Agrwal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, “QUOTIENT: Two-party secure neural network training and prediction,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., ACM, 2019, pp. 1231–1247.
[50] P. Mohassel and Y. Zhang, “SecureML: A system for scalable privacy-preserving machine learning,” in *Proc. IEEE Symp. Secur. Privacy*, IEEE, 2017, pp. 19–38.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[52] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, “High-throughput secure three-party computation for malicious adversaries and an honest majority,” in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*. Springer, 2017, pp. 225–255.

[53] J. J. Hall, “A database for handwritten text recognition research,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, May 1994.

[54] Y. LeCun, C. Cortes, and C. J. Burges, “MNIST database,” 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist

[55] Zalando, “Fashion MNIST database,” 1998. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

[56] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, “High-throughput secure three-party computation for malicious adversaries and an honest majority,” in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*. Springer, 2017, pp. 225–255.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[58] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” *Tech. Rep.*, 2009.

Zhuoran Ma received the BE degree from the School of Software Engineering, Xidian University, Xi’an, China, and PhD degree from the Department of Cyber Engineering from Xidian University, in 2022. She is currently a lecturer with the Department of Cyber Engineering in Xidian University. Her research interests include data security and federated learning.

Jianfeng Ma (Member, IEEE) received the PhD degree in computer software and telecommunication engineering from Xidian University, Xi’an, China, in 1988 and 1995, respectively. From 1999 to 2001, he was a research fellow with the Nanyang Technological University of Singapore. He is currently a professor and a PhD supervisor with the Department of Computer Science and Technology, Xidian University, Xi’an, China. He is also the director of the Shaanxi Key Laboratory of Network and System Security. His current research interests include information and network security, wireless and mobile computing systems, and computer networks.

Yinbin Miao received the BE degree with the Department of Telecommunication Engineering from Jilin University, Changchun, China, in 2011, and the PhD degree with the Department of Telecommunication Engineering from Xidian University, Xi’an, China, in 2016. He is currently a lecturer with the Department of Cyber Engineering in Xidian University, Xi’an, China. His research interests include information security and applied cryptography.

Xi meng Liu (Senior Member, IEEE) received the BSc degree in electronic engineering from Xidian University, Xi’an, China, in 2010 and the PhD degree in cryptography from Xidian University, China, in 2015. Now he is a full professor with the College of Mathematics and Computer Science, Fuzhou University. Also, he was a research fellow with the School of Information System, Singapore Management University, Singapore. He has published more than 200 papers on the topics of cloud security and big data security including papers in *IEEE Transactions on Computers*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Services Computing*, *IEEE Internet of Things Journal*, and so on. He awards “MINjiang Scholars” distinguished professor, “Qishan Scholars” in Fuzhou University, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and big data security. He is a member of the ACM, CCF.

Wei Zheng received the BE degree from the Department of Information Security from Nanchang University, Nanchang, China, in 2020. He is currently working toward the ME degree with the Department of Cyber Engineering from Xidian University, Xi’an, China. His research interests include information security and applied cryptography.

Kim-Kwang Raymond Choo (Senior Member, IEEE) received the PhD degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship with the University of Texas at San Antonio. He is the founding co-editor-in-chief of ACM distributed ledger technologies: Research & practice, and the founding chair of IEEE Technology and Engineering Management Society Technical Committee (TC) on Blockchain and Distributed Ledger Technologies. He is also the recipient of the 2022 IEEE Hyper-Intelligence TC Award for Excellence in Hyper-Intelligence Systems (Technical Achievement award), the 2022 IEEE TC on Homeland Security Research and Innovation Award, the 2022 IEEE TC on Secure and Dependable Measurement Mid-Career Award, the 2019 IEEE TC on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher).

Robert H. Deng (Fellow, IEEE) has been a professor with the School of Information Systems, Singapore Management University, since 2004. His research interests include data security and privacy, multimedia security, and network and system security. He was an associate editor of the *IEEE Transactions on Information Forensics and Security* from 2008 to 2012. He is currently an associate editor of the *IEEE Transactions on Dependable and Secure Computing* and *Security and Communication Networks* (John Wiley). He is the cochair of the Steering Committee of the ACM Symposium on Information, Computer and Communications Security. He received the University Outstanding Researcher Award from the National University of Singapore in 1999 and the Lee Kuan Yew fellow for research excellence from Singapore Management University, in 2006. He was named Community Service Star and Showcased Senior Information Security Professional by (ISC)² under its Asia-Pacific Information Security Leadership Achievements Program, in 2010.