Functional Encryption Based Approaches for Practical Privacy-Preserving Machine Learning

by

Runhua Xu

M.S. in Computer Science, Beihang University, China, 2014

B.E. in Software Engineering, Northwestern Polytechnical University, China, 2011

Submitted to the Graduate Faculty of
the School of Computing and Information in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2020

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Runhua Xu

It was defended on

August 3^{rd} 2020

and approved by

Dr. James Joshi, School of Computing and Information, University of Pittsburgh

Dr. Prashant Krishnamurthy, School of Computing and Information, University of Pittsburgh

Dr. Balaji Palanisamy, School of Computing and Information, University of Pittsburgh

Dr. Nathalie Baracaldo, IBM Almaden Research Center

Dissertation Director: Dr. James Joshi, School of Computing and Information, University of Pittsburgh

Functional Encryption Based Approaches for Practical Privacy-Preserving Machine Learning

Runhua Xu, PhD

University of Pittsburgh, 2020

Machine learning (ML) is increasingly being used in a wide variety of application domains. However, deploying ML solutions poses a significant challenge because of increasing privacy concerns, and requirements imposed by privacy-related regulations. To tackle serious privacy concerns in ML-based applications, significant recent research efforts have focused on developing privacy-preserving ML (PPML) approaches by integrating into ML pipeline existing anonymization mechanisms or emerging privacy protection approaches such as differential privacy, secure computation, and other architectural frameworks. While promising, existing secure computation based approaches, however, have significant computational efficiency issues and hence, are not practical.

In this dissertation, we address several challenges related to PPML and propose practical secure computation based approaches to solve them. We consider both two-tier cloud-based and three-tier hybrid cloud-edge based PPML architectures and address both emerging deep learning models and federated learning approaches. The proposed approaches enable us to outsource data or update a locally trained model in a privacy-preserving manner by employing computation over encrypted datasets or local models. Our proposed secure computation solutions are based on functional encryption (FE) techniques. Evaluation of the proposed approaches shows that they are efficient and more practical than existing approaches, and provide strong privacy guarantees. We also address issues related to the trustworthiness of various entities within the proposed PPML infrastructures. This includes a third-party authority (TPA) which plays a critical role in the proposed FE-based PPML solutions, and cloud service providers. To ensure that such entities can be trusted, we propose a transparency and accountability framework using blockchain. We show that the proposed transparency framework is effective and guarantees security properties. Experimental evaluation shows that the proposed framework is efficient.

Table of Contents

List of Tables

List of Figures

Preface

This dissertation could not have been possible without the help and support of many people over the years. I would like to express my deep gratitude to many people who have been indispensable in my journey of Ph.D. training. First and foremost, I am deeply thankful to my advisor Dr. James Joshi for his supervision. He has been patiently mentoring me and supporting my research work, enriching my academic activities, and expanding my academic vision. I would like to thank the dissertation committee members, Dr. Prashant Krishnamurthy, Dr. Balaji Palanisamy, and Dr. Nathalie Baracaldo for their invaluable feedback and insightful suggestions on my dissertation work. Part of this dissertation work was done while I was interning at the *IBM Almaden Research Center*, and here I also would like to express my deep gratitude to and acknowledge the support and help of the team members of *AI Security and Privacy Solutions*, Dr. Yi Zhou, Dr. Ali Anwar, and Dr. Heiko Ludwig. In particular, I am especially grateful to my internship mentor, Dr. Nathalie Baracaldo for her continuous guidance and support. I am also grateful for and acknowledge the support of the National Science Foundation (NSF) grants (DGE #1438809 and OAC #1642117) that helped me pursue my Ph.D. studies.

Great thanks also go to my fellow colleagues at *LERSAIS* and the *School of Computing and Information*, and my friends in Pittsburgh, for all the happy times during these years. I also would like to express my endless gratitude to my family, Shan, Xiang, Juan, Xia, Xian, whose love and support have always been the cornerstone in pursuit of my dreams. Last but not least, I am very grateful for the fundamental education and training in the past several years that prepared me to continue my Ph.D. degree here at Pitt.

1.0 Introduction

Machine learning (ML) is increasingly being applied in a wide variety of application domains. Especially, emerging deep neural networks (DNN) (a.k.a, deep learning (DL)) models have shown significant model accuracy and performance improvement in many application areas such as computer vision, natural language processing, speech, or audio recognition, etc., [105, 76, 147]. Federated learning (FL) (a.k.a collaborative learning) is another emerging ML technique that enables training a high-quality centralized model while training data remains distributed over multiple decentralized devices [120, 100]. FL has shown its promise in various application domains including healthcare, vehicular networks, and smart manufacturing, [188, 155, 82]. Although these models have shown huge success in the AI-powered or ML-driven applications, they still face several challenges such as (i) lack of powerful computational resources, and (ii) availability of huge volumes of available data to be used for training ML models. For example, a well-performing deep neural networks model relies on a huge volume of training data and high-powered computational resources to support both the training and inference phases.

To address the need of powerful computing resources with higher performance CPUs and GPUs, larger memory storage, etc., existing commercial ML-related infrastructure service providers such as Amazon, Microsoft, Google, and IBM have devoted significant amounts of their efforts toward building *infrastructure as a service* (IaaS) or *machine learning as a service* (MLaaS) platforms with appropriate rental fees for clients that do not have such powerful computing resources. Clients can employ such an ML-related IaaS to manage and train their ML models and provide data analytics and prediction services in their applications or to their customers directly.

Another key challenge is the availability of data to train ML models. With larger training datasets, we have a possibility of training ML models that have better accuracy. Hence, there is a need for collecting large volumes of data potentially from multiple sources. However, the collection and use of datasets raise serious privacy concerns because of privacy-sensitive information they may contain, as is evident from recent data breaches [150, 176]. An adversary may also be able to infer private information from an ML model; for instance, the adversary may infer that a patient's data has been included in the training of an HIV-related ML model (a.k.a, the membership inference attack [167, 19, 90, 110, 130]). Furthermore, existing regulations such as the Health Insurance Portability and Accountability Act (HIPPA) and more recent ones such as the European General Data Protection Regulation (GDPR), Cybersecurity Law of China, New York SHIELD Act, California Consumer Privacy Act (CCPA), etc., place various restrictions on the availability and use of privacy-sensitive data. Such privacy concerns and regulatory restrictions pose a significant challenge in training a well-performing ML model and can hinder the use of ML models for real-world applications.

To tackle the increasing privacy concerns related to using ML in applications where users' privacy-sensitive data such as electronic health/medical records, location information, etc., are stored and processed,

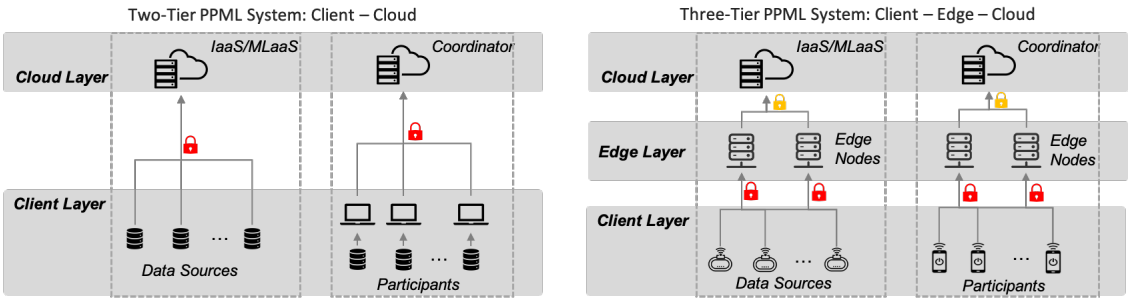


Figure 1.1: Two typical architectures of PPML systems.

it is important to devise well-designed privacy-preserving machine learning (PPML) approaches. Towards this, there are significant efforts focused on PPML research that aims at integrating existing anonymization mechanisms or newly designed privacy-preserving approaches and ML. Further discussion on existing related work is presented in Chapter 2. Existing privacy-preserving approaches have been proposed to address some of these privacy issues, and integrated into ML approaches; however, these approaches still have their own limitations. For instance, integration of *differential privacy* in ML models has been shown to lead to potential loss of model [1]. Similarly, the use of general secure multi-party computation (garbled circuits-based) approaches incurs high communication overhead because of large volumes of intermediate data such as garbled tables of circuit-gates that need to be transmitted during the execution of the protocols [151].

In this dissertation, we explore secure computing approaches and show that they are promising for generating PPML models with privacy guarantees. Towards this, we propose efficient and practical PPML approaches based on *functional encryption techniques*. In these proposed approaches, data is protected using encryption and computational ML tasks are carried out over such encrypted data. We also explore the issues related to the trustworthiness of the proposed PPML infrastructure and propose our solutions.

In the rest of the chapter, we first discuss the motivation and challenges for the proposed research and outline the key research tasks.

1.1 Motivation and Challenges

Early research related to PPML can be traced back to privacy-preserving data mining research several years ago, as discussed and analyzed in [9, 114]. PPML is currently an active area of research because of (i) rapid developments in ML research, including in DNN or federated learning (FL), and their increasing adoption in applications; and (ii) significant privacy concerns users have with regards to the use of ML models, and accompanying stricter privacy protection regulations and acts, which restrict the use of privacy-

sensitive data for training ML models. In addition, there also has been significant progress in privacy research that has produced various privacy protection approaches that can be used to address PPML challenges.

To tackle increasing privacy concerns of deploying ML techniques, more recent research work on PPML is being proposed from broader research communities of machine learning, distributed systems, security, and privacy. In this dissertation, we mainly focus on secure computation techniques - a fundamental component in PPML systems - to allow a third-party to acquire the result of a computation (e.g., the trained model) over privacy-sensitive data without disclosing private information to the third-party. In particular, we use functional encryption techniques as the underlying cryptographic framework to address PPML challenges by considering various ML approaches (mainly DNN and FL) and architectures (i.e., two-tier and three-tier PPML-enabled systems as depicted in Figure 1.1) as follows:

Two-Tier PPML Architecture. A two-tier PPML architecture includes a cloud layer containing IaaS, MLaaS or coordinating servers, and a client layer including participants that are primarily data sources and/or those who own data as well as partial computational resources. In general, the raw data or the locally trained model is protected by a cryptography scheme and the subsequent processes like aggregate computation and training computation are carried over the encrypted data or model parameters. Here we focus on two areas of ML.

Privacy-Preserving Federated Learning: FL has been shown as a promising architectural ML approach that enables collaborative training of models among multiple participants - under the orchestration of a coordinator - without sharing any of their raw training data. FL can thus provide basic privacy protection as data remains local to users' own domains. To enhance privacy guarantee, additional privacy-preserving approaches such as differential privacy (DP) and secure multi-party computation (SMC) are being integrated within an FL framework to protect each participant's model updates. However, integrating a DP mechanism in ML can introduce some loss in utility (i.e., reduces the accuracy of a trained ML model), while integrating SMC techniques in ML can introduce significant communication or computational overheads. The main challenge here is how to enhance privacy guarantees for FL while taking into account model accuracy, communication efficiency, training efficiency, and support of dynamic participants, etc.. In particular, we propose to address the following key research questions in this dissertation:

- How to provide enhanced privacy protection in FL to protect each participant's model updates while providing acceptable model accuracy?
- How to ensure that privacy-preserving approaches support efficient aggregation with regards to computational and communication overheads?
- How to ensure that such an approach can support a group of participants that provide datasets, where each participant is able to join or leave the group even during the learning phase?
- How to ensure that a privacy-preserving approach incorporates various data partitioning cases, where each participant may only have a part of horizontally and/or a vertically partitioned dataset.

Privacy-Preserving Deep Neural Networks: Training DNN models require high-performance computing resources; however, such machines are not available for most enterprises, especially, the small-scale businesses. Thus, existing ML-related *IaaSs* are employed to train ML models to enhance their businesses. However, such services do not appropriately address privacy concerns associated with processing of the privacy-sensitive data. Here, the main challenge is how to enhance privacy protection in an existing two-tier IaaS-based architecture using appropriate privacy-preserving approaches; in particular, such an approach should provide protection of privacy-sensitive data through the use of a cryptosystem, and training of an ML model is done over encrypted data. To be specific, the following key research questions will be addressed in this dissertation:

- How can we train a DNN over encrypted privacy-sensitive data? We note that most of the existing similar approaches only work in the inference phase of an ML model in a third-party IaaS;
- While training such a privacy-preserving DNN model, how can we ensure it is efficient with regards to training time and communication overheads so as to make it practically deployable?
- How can we ensure that such a privacy-preserving technique is scalable with regards to including multiple data sources, and multiple cloud services while considering various dataset partitioning cases?

Three-Tier PPML Architecture. A three-tier PPML includes a cloud layer and a client layer as in the two-layer PPML architecture, but it additionally contains an edge layer including several edge nodes that can preprocess data sent by the devices (e.g., sensors or mobile devices) in the client layer in a privacy-preserving manner. Similar to the two-tier PPML system, the raw data or the local ML models are also protected by some crypto schemes. The preprocessing at the edge and final processing at the cloud are both done over the encrypted data. The three-tier PPML architecture tries to take advantage of the recently emerging promising edge computing paradigm where edge nodes are placed closer to mobile devices or sensors so as to deliver highly responsive and scalable cloud services, and to provide preprocessing services in the Internet of Things (IoT) ecosystem. However, such a three-tier architecture brings new challenges. In particular, we propose to address the following key issues in this dissertation:

- In contrast to cloud data centers where cloud servers are managed through strict and regularized policies, edge nodes may not have the same degree of regulatory and monitoring oversight. Thus, edge nodes may not be as trusted as the cloud data centers; so an appropriate privacy protection mechanism is needed to ensure privacy leakage is prevented at the edge layer.
- Existing crypto-based secure computation techniques are not applicable in such a three-tier edge computing architecture. In particular, existing homomorphic encryption (HE) based PPML approaches are computationally inefficient to be deployed at the edge layer of IoT devices with limited capabilities. While more efficient functional encryption (FE) based PPML has been proposed in the literature, they have not been aimed at such three-tier edge computing-based architecture, so they cannot be directly employed.

  In essence, compared to general-purpose garbled-circuits based secure computation approaches, emerging

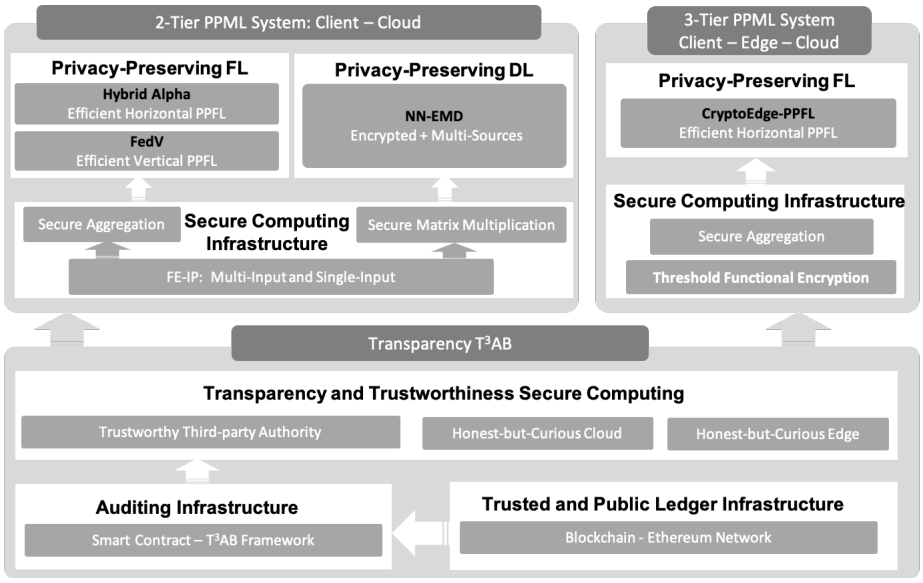


Figure 1.2: Overview of research tasks in this dissertation.

crypto-based secure computation techniques (i.e., computing over the encrypted data) have been shown to be more promising in terms of achieving a strong privacy guarantee as the data is protected by an encryption algorithm. Further, it is easier to integrate them with other types of approaches to form a customized hybrid approach. Our proposed secure computation based approaches in this dissertation are essentially hybrid in nature based FE techniques.

Another critical issue is the trustworthiness of various entities involved in PPML. For instance, an *IaaS* server is a *coordinating* server discussed above, which is assumed to be *honest-but-curious*. In addition, *edge nodes* in a three-tier ML architecture may also be *semi-trusted*. Besides, part of crypto-based secure computation approaches rely on a critical component, namely, the *third-party authority* (TPA), that needs to be *fully trusted*. However, for practical deployment of PPML such trust assumptions (e.g., fully trusted TPA) can be a problem. Thus, mechanisms are needed to remove such trust assumptions and ensure that all the entities are *honest* and can be trusted during the training or inference phases of a PPML approach.

1.2 Overview of Research Tasks

In this dissertation, as illustrated in Figure 1.2, we carry out the following four research tasks: (i) development of an efficient and secure aggregation technique for privacy-preserving FL frameworks and an efficient secure computation approach for privacy-preserving DNN models over an encrypted local model or

raw datasets in a two-tier PPML architecture; (ii) design of an efficient secure aggregation technique for training FL models in three-tier PPML architecture to leverage the benefits of edge layer; (iii) construction of mechanisms to make secure computation based PPML infrastructures as proposed in the tasks (i) and (ii) more transparent to increase its trust. We present an overview of proposed research tasks below.

1.2.1 Practical Secure Computation in a Two-Tier PPML System

For two-tier PPML architecture, we focus on practical secure computation techniques for privacy preserving FL (PPFL) and privacy-preserving DNN (PPDNN).

Efficient Secure Aggregation in PPFL. In FL, there are several *participants* each training its own ML model locally based on its own local training dataset. A *coordinator* helps to coordinate and aggregate the model update parameters from each *participant* to generate a more accurate global model. To prevent inference attacks where an *honest-but-curious* coordinator may infer private information from a local model uploaded by a participant, a secure aggregate computation approach is needed to protect the privacy of each participant's local model. However, existing secure aggregate computation approaches have limitations in terms of computational efficiency or complex protocol interactions. To address these issues, in this research task, we propose secure aggregation approaches that improve the efficiency of a PPFL framework while providing a strong privacy guarantee. Based on the types of partitioning of datasets, we divide the research task into two sub-tasks:

- Development of a secure aggregate computation approach for horizontal PPFL: A *horizontal FL* is one where datasets are horizontally partitioned, i.e., each participant has a complete set of features of each sample in its training dataset, and hence each participant is able to train a complete local model independently. We propose a new secure aggregate computation approach for horizontal PPFL that improves efficiency over existing approaches while protecting the privacy of each participant's local model.
- Development of a secure aggregate computation approach for vertical PPFL: A *vertical FL* represents federated learning over vertically partitioned datasets, where each participant has a partial set of features of each sample in its training dataset. As a result, each participant is able to train only a "partial" local model instead of a complete one independently and the aggregation operation is not as straightforward as in a horizontal PPFL. In this sub-task, we propose a vertical PPFL approach that addresses the challenge of computing over these "partial" models efficiently.

Efficient Secure Computation in PPDNN. We propose several secure computation approaches for PPDNN to support computational tasks related to training a DNN model over an encrypted dataset using a client-server or a client-cloud architecture. In such a case, a client that has privacy-sensitive data but limited computational resources can train a DNN model by employing a third-party IaaS without leaking its privacy-sensitive data. Existing secure computation approaches, such as ones based on homomorphic encryption [129], that can be used to support training a DNN model over encrypted data have efficiency

problems, making it difficult for practical deployment. In this task, we propose an innovative approach to construct secure computation protocols based on a functional encryption family. The proposed privacy-preserving approach supports both the training and the inference phases of a DNN model. In particular, our proposed PPDNN supports training neural networks over encrypted datasets from multiple sources. In the above-mentioned two-tier *client-cloud* or *client-server* architecture, the third-party ML-related IaaS platform is responsible for training the DNNs, while the clients provide their encrypted privacy-sensitive training datasets. As we know, the performance of a DNN model relies on the size of a training dataset. In general, a single client or data source may not be able to provide an adequate amount of training data. In this task, we develop an efficient secure computation approach that supports training DNNs over multiple encrypted datasets from multiple data sources, where each dataset may be partitioned vertically or horizontally.

1.2.2 Practical Secure Computing for Three-Tier PPML System

In this dissertation, we also propose a secure aggregation approach applicable for a three-tier PPML architecture that leverages the benefits of emerging edge computing paradigm to support a broader set of applications. In particular, such a system includes the following entities:

- *IoT* devices that can easily encrypt raw data;
- *untrusted* or *semi-trusted* edge nodes that have the capability of processing encrypted data (i.e., partially decrypt the encrypted raw data) without learning any privacy-sensitive information from the encrypted data;
- a cloud data center that can construct the final aggregation results, but still cannot learn any privacy-sensitive information.

We propose a three-tier PPML framework that includes a novel and efficient secure computation approach to support practical secure aggregation over encrypted data at the edge. In general, the cloud center can specify a random number of edge nodes to collaboratively *preprocess* the encrypted data where the processed data is still in ciphertext form, without leaking any information to the edge nodes. Then the cloud can decrypt the preprocessed ciphertexts to compose the final aggregation results in plaintext. The proposed secure computation approach can dramatically reduce the overall decryption time and save computational resources for the cloud, as the edge nodes are allowed to do some preprocessing over the encrypted data. Our proposed three-tier PPML framework is based a novel threshold functional encryption (TFE) scheme.

1.2.3 Trustworthy Infrastructure for Secure Computation

In this dissertation, we also propose an approach to build a trustworthy infrastructure for the above-mentioned proposed secure computation approaches. As introduced in research tasks 2 and 3, the proposed secure computation solutions are based on a functional encryption family that relies on a third-party authority (TPA) that is responsible for providing key services such as initializing and distributing public-key

parameters, and generating functional private keys. The proposed approach assumes that TPA is fully trusted, but for practical deployment of such approaches simply trusting the TPA is not adequate. Besides, the IaaS platform and the coordinator are also assumed to be *honest-but-curious*, which is a common assumption in most PPML frameworks. It is also critical that we have mechanisms in place to ensure that all these entities indeed behave honestly or can be trusted. Towards this, we propose a transparency framework to ensure that the TPA infrastructure is trustworthy. The proposed transparency approach ensures that the operations of these entities are monitored and audited by participants based on the logged pieces of information.

1.3 Summary and Dissertation Outline

In this chapter, we have presented the background, motivation, and challenges related to secure computation based PPML. We have also overviewed the research tasks carried out as part of this dissertation to address the PPML challenges identified.

The outline of this dissertation is as follows: In Chapter 2, we present a literature review related to the proposed tasks. We present the proposed secure computing approaches for PPML in Chapter 3, and for training DNNs using encrypted multi-sourced datasets in Chapter 4. In Chapter 5, we present the proposed solution for practical secure aggregation issues in edge-enabled architecture. We present the proposed transparency framework using blockchain in Chapter 6. In Chapter 7 we present conclusions and future work.

2.0 Literature Review

In this chapter, we present a literature review of related work in privacy-preserving machine learning (PPML) that are related to our proposed approaches. We discuss several representative PPML solutions proposed in the literature, as summarized in Table 2.1, from three dimensions: *affected phase* of an ML pipeline and *design goals*. In general, privacy-preserving techniques such as differential privacy and various types of secure computation techniques have most commonly been employed for PPML. Furthermore, recently proposed secure computation approaches for PPML that rely on emerging cryptosystems such as homomorphic encryption and functional encryption have shown promise for achieving strong privacy guarantees. More detailed discussion is presented in the rest of the chapter.

2.1 Affected Phases of Privacy-Preserving Approaches

Typical phases of an ML system include *production phase* and *consumption phase*. The production phase mainly focuses on how to train an ML model using the collected training data or local model update from the data producer, while the consumption phase works on how to consume those trained models, such as making use of inference service. So existing privacy-preserving machine learning solutions target *privacy-preserving training*, *privacy-preserving inference*, or *both*.

From the perspective of computation, there is no strict boundary between a privacy-preserving training and a privacy-preserving inference as computation in the inference procedure could be viewed as a simplified version of training procedure without training labels [54]. For instance, the training phase of a neural network model could be viewed as a process where a set of data is continuously fed into the designed network for multiple training epochs, while the inference phase could be treated as one epoch of computation for one data sample to generate a prediction label. Formally, given a set of training samples $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$, where $\boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}$, the goal of a MM model (for simplicity, assume a linear model) is to learn a fit function denoted as

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} + b, \tag{2.1}$$

where $\boldsymbol{w} \in \mathbb{R}^m$ is a set of model parameters, and b is the intercept. To find proper model parameters, usually, we need to minimize the regularized training error given by

$$E(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\boldsymbol{x}_i)) + \alpha R(\boldsymbol{w}), \tag{2.2}$$

where $\mathcal{L}(\cdot)$ is a loss function that measures model fit and $R(\cdot)$ is a regularization term (a.k.a. penalty) that penalizes model complexity; α is a non-negative hyperparameter that controls the regularization strength. Regardless of various choices of $\mathcal{L}(\cdot)$ and $R(\cdot)$, stochastic gradient descent (SGG) is a common optimization

Table 2.1: Representative proposal of privacy-preserving machine learning systems

Proposals	Affected Phase	Designing Goals
[38, 91]	Training	Differential privacy data release
[1, 72, 109]	Training	Differential private SGD
[166]	Training	Distributed selective SGD
[120, 100, 138, 154]	Training	Federated learning
[125]	Both	Secure computation delegation
[128, 151]	Both	GC-based secure computing
[73, 78, 75, 86, 31, 139, 92]	Inference	HE-based secure computing
[27]	Inference	customized SC by HE
[22]	Training	FL/secret sharing/encryption
[129]	Both	HE-based secure computing
[191]	Both	FE-based secure computing
[173]	Training	FL/DP/THE-based secure computing
[189]	Training	FL/DP/FE-based secure computing

method for unconstrained optimization problems as discussed above. A simple SGD method needs to *iterate* over the training samples and for each sample it needs to *update* the model parameters according to the following update rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} E \leftarrow \boldsymbol{w} - \eta[\alpha \nabla_{\boldsymbol{w}} R + \nabla_{\boldsymbol{w}} \mathcal{L}] \tag{2.3}$$

$$b \leftarrow b - \eta \nabla_b E \leftarrow b - \eta[\alpha \nabla_b R + \nabla_b \mathcal{L}] \tag{2.4}$$

where η is the learning rate which controls the step-size in the parameter space. Given a trained model $(\boldsymbol{w}_{\text{trained}}, b_{\text{trained}})$, the goal of the inference phase is to predict a label $\hat{y}$ with target sample $\boldsymbol{x}$ as follows:

$$\hat{y} \leftarrow f_{\boldsymbol{w}_{\text{trained}}, b_{\text{trained}}}(\boldsymbol{x}) \tag{2.5}$$

In general, the task of privacy-preserving training is more challenging than the task of privacy-preserving inference. Most of the existing privacy-preserving training solutions indicate the inclusion of privacy-preserving inference even though they may not explicitly state that.

2.1.1 Privacy-Preserving Training

The goal of a privacy-preserving training is to prevent the leakage of privacy sensitive information in *training data*. Essentially, the key factors related to training are *data* and *computation*, and consequently, existing proposed approaches try to tackle the challenge of privacy-preserving training from the following two aspects: (i) how to distill/filter the training data such that the processed data includes less or no privacy-sensitive information; or (ii) how to process or compute over the training data in a privacy-preserving manner.

From the perspective of *data*, existing privacy-preserving training approaches either (i) adopt traditional anonymization mechanisms such as k-anonymity[171], l-diversity[118] and t-closeness [111] to remove identifier and quasi-identifier information in the training data before sending it out for training, or (ii) employ

ϵ-differential privacy [59, 62, 61] mechanism to add privacy budget (noise) into the dataset to avoid leakage of private information against a set statistical queries.

For instance, the approach proposed in [67] tries to provide k-anonymity for data mining algorithms, while the approaches proposed in [107, 98] focus on the utility metric and provide a suite of anonymization algorithms to produce an anonymous view based on ML workloads. On the other hand, recently, differential privacy mechanism has shown its promise in emerging deep learning models that rely on training on a large dataset. For example, the early work in [1] proposes a differentially private stochastic gradient descent (often abbreviated SGD) approach to train a privacy-preserving DL model. The approach proposed in [121] demonstrates that it is possible to train large recurrent language models with user-level differential privacy guarantees with only a negligible cost in predictive accuracy. Parameter-transfer based meta-learning (i.e., applications including few-shot learning, federated learning, and reinforcement learning) often requires task-owners to share model parameters, because of which privacy leakage becomes possible. Proposed approaches for privacy-preserving meta-learning such as in [72, 189] try to tackle the problem of private information leakage in FL by employing an algorithm to achieve client-sided differential privacy. The method proposed in [109] formalizes the notion of task-global differential privacy and proposes a differentially private algorithm for gradient-based parameter transfer that satisfies the privacy requirement while retaining provable transfer learning guarantees in convex settings.

Emerging area of secure computation over encrypted data in the cryptography community is showing another promising approach to protecting the privacy of training data. Unlike a traditional anonymization technique or a differential privacy mechanism that aims at protecting against inference or de-anonymization attacks, such as those demonstrated in [183, 146, 167, 144], wherein an adversary may have additional background knowledge, the encryption based approaches can provide strong privacy guarantees, referred to as *confidential-level privacy* in this chapter; these approaches have received more and more attention in recent studies such as in [173, 191, 189, 69, 83, 39, 73, 31, 129]; in these approaches, the training data or the shared model updates are protected by cryptosystems.

From the perspective of *computation*, existing privacy-preserving training approaches are also correspondingly divided into two approaches: (i) cases where the training data is processed by traditional anonymization mechanisms or differential privacy mechanism; in such cases, computation involved in training is as normal as in non-PPML model training; (ii) cases where the training data is protected via cryptosystems to achieve a confidential-level privacy; here, the privacy-preserving (i.e., referred to as crypto-based) training computation is more complex compared to that in normal non-PPML model training. Crypto-based training approaches rely on recently proposed cryptographic schemes, mainly, homomorphic encryption schemes [70, 175, 28, 119, 6] and functional encryption schemes [25, 74, 4, 14, 13, 2, 3]. Unlike in a normal training process, it is worth noting that there is an extra step - *data conversion* - in crypto-based training approaches because most of these cryptosystems are built on the integer group while most of the training data is in floating-point number format, especially, after normalization. Note that normalization is a very common

preprocessing method in most of ML approaches [189, 191, 83, 129]. The data conversion step includes a pair of encoding and decoding operations. The encoding step is commonly adopted to convert the floating-point numbers into integers so that the training data can be encrypted and then used in a crypto-based training. On the contrary, the decoding step is applied to the trained model or the result of crypto-based training to recover the floating-point numbers. Obviously, the data conversion procedure indicates that there is a potential precision loss during a crypto-based training. We will discuss the details of the potential impact of data conversion step in Section 2.2.

2.1.2 Privacy-Preserving Inference

Several existing PPML approaches that focus on privacy-preserving training indicate that the proposed solution may also support privacy-preserving inference, as illustrated in [128, 151, 125, 191, 129]. We also observe that most of the proposals that apply cryptographic approaches mainly use homomorphic encryption and its related schemes, such as in [73, 78, 75, 86, 31, 139, 92, 27]. These approaches only target the inference phase, as these cryptosystems are not efficient enough to be applied in the training phase as it involves complex and massive computation.

Part of privacy-preserving inference approaches also focus on the privacy-preserving model query or publication; in this case, a model user is separate from a model owner. And a key concern here is how to prevent adversaries (i.e., a curious model user) from inferring private information of the model owner from the model itself, in particular, when an adversary has been allowed to iteratively query the inference service. To address these issues, a naive privacy-preserving solution is to limit the number of queries for a model user.

In addition to these, existing prevention methods can be categorize into three approaches:

- a *private aggregation of teacher ensembles (PATE)* approach [136, 138], wherein the knowledge of an ensemble of "teacher" models is transferred to a "student" model, with intuitive privacy provided by training teachers on disjoint datasets and strong privacy guaranteed by noisy aggregation of teachers' answers;
- *model transformation* approach such as MiniONN [115], where an existing model is transformed into an oblivious neural network supporting privacy prediction with reasonable efficiency;
- *model compression* approach, especially, applied in emerging DL models with a large set of model parameters, where knowledge distillation methods [88, 143] are adopted to compress the trained DL models. Even though the main goal of knowledge distillation is to reduce the size of a DL model, such a method also brings additional privacy-preserving features [137, 178]. Intuitively, the distillation procedure not only removes the redundant information in a model but also reduces the probability that an adversary can infer potential private information in the model through iterative queries.

2.2 Design Principle of Privacy-Preserving Approaches

In this section, we discuss privacy-preserving approaches in a more fine-grained manner, namely, the design principles used in these approaches; here, we focus how these approaches tackle the following issues:

- How is privacy-sensitive data released?
- How is privacy-sensitive data processed?
- Does a PPML architecture prevent the disclosure of private information?

Correspondingly, we introduce the following four types of the privacy-preserving approaches: (i) data publishing based approaches; (ii) data processing based approaches; (iii) architecture-based approaches; and (iv) hybrid approaches that may combine or integrate previous approaches.

2.2.1 Data Publishing Based Approaches

In general, a data publishing based privacy-preserving approach includes the following:

- *De-identification-based Approaches* that partially *remove* the identifiers in the raw data;
- *Perturbation-based Approaches* that partially *perturb* the statistical result of the raw data;
- *Crypto-based Approaches* that totally *encrypt* the raw data.

De-identification-based Approaches: De-identification based approaches essentially remove some information that can lead to identification. Here, techniques such as k-anonymity[171], l-diversity[118] and t-closeness [111] are applied to the raw privacy-sensitive data to remove privacy-sensitive information so as to protect from potential inference attacks. Specifically, a k-anonymity mechanism can ensure that the information for each person contained in the released dataset cannot be distinguished from at least k-1 other individuals whose information is also released in the dataset. To achieve that, k-anonymity approaches define *identifiers* and *quasi-identifiers* for each data attribute, and then remove the identifiers and partially hide the *quasi-identifiers* information. The l-diversity mechanism introduces the concept of *equivalence classes*, where an equivalence class satisfies l-diversity if there are at least l "well-represented" values for a sensitive attribute. A dataset satisfies l-diversity if every equivalence class of the dataset satisfies l-diversity [171, 118]. Essentially, as an extension of the k-anonymity mechanism, l-diversity mechanism reduces the granularity of the data representation and additionally maintain the diversity of sensitive fields by adopting techniques like generalization and suppression such that given any records it can be mapped to at least $k-1$ other records in the dataset. t-closeness is further refinement of l-diversity by introducing additional restriction on the distribution value over an *equivalence class*. An equivalence class satisfies t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of all attributes in the whole dataset is no more than a threshold t. Similarly, a dataset satisfies t-closeness if all equivalence classes satisfy t-closeness.

Perturbation-based Approaches: Perturbation based approaches mainly refer to using ϵ-*differential privacy* techniques [59, 62, 61]. As per [60, 62], differential privacy can be defined as follows: a randomized mechanism $\mathcal{M}: \mathcal{D} \rightarrow \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ϵ, δ)-*differential privacy* if for any two adjacent input $d, d^{'} \in \mathcal{D}$ and for any subset of outputs $S \subseteq \mathcal{R}$, it holds that

$$\Pr[\mathcal{M}(d) \in S] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(d^{'}) \in S] + \delta \tag{2.6}$$

The additive term δ allows for the possibility that plain ϵ-differential privacy is broken with probability δ (which is preferably smaller than $1/|d|$). Usually, a paradigm of an approximating a deterministic function $f: \mathcal{D} \rightarrow \mathbb{R}$ with a differentially private mechanism is via *additive noise* calibrated to function's *sensitivity* S_f, which is defined as the maximum of the absolute distance $|f(d) - f(d^{'})|$. The representative and common additive noise mechanisms for real-valued functions are Laplace mechanism and Gaussian mechanism defined, respectively, as follows:

$$\mathcal{M}_{\text{Gauss}}(d; f, \epsilon, \delta) = f(d) + \mathcal{N}(\mu, \sigma^2) = f(d) + \mathcal{N}(0, \frac{2\ln(1.25/\delta)}{\epsilon^2} \cdot S_f^2) \tag{2.7}$$

$$\mathcal{M}_{\text{Lap}}(d; f, \epsilon) = f(d) + Lap(\mu, b) = f(d) + Lap(0, \frac{S_f^2}{\epsilon}) \tag{2.8}$$

Typical uses of differential privacy enabled PPML i along two directions: (i) directly adopting above-mentioned additive noise mechanism on the raw dataset in the case of publishing data, as illustrated in [38, 91]; or (ii) transforming the original training method into differentially private training method so that the trained model has ϵ-differential privacy guarantee in the case of publishing model, as illustrated in [1, 72, 109].

Sketching is an approximate and simple approach for data stream summary, by building a probabilistic data structure that serves as a frequency table of events, like counting Bloom filters. Recent theoretical advances [7, 123] have shown that differential privacy is achievable on sketches with additional modifications. For instance, the work in [16] focuses on the privacy-preserving collaborative filtering, a popular technique for a recommendation system, by using sketching techniques to implicitly provide the differential privacy guarantees by taking advantage of the inherent randomness of the data structure used. Most recent work as reported in [113] proposes a novel sketch-based framework for distributed learning; this approach involves compressing the transmitted messages via sketches to simultaneously achieve communication efficiency and provable privacy benefits.

In summary, traditional anonymization mechanisms and differential privacy mechanisms are designed to tackle general data publishing problems. More recently, differential privacy has been widely adopted in privacy-preserving DL and privacy-preserving FL approaches, such as in [1, 121, 72, 109, 173, 189]. Furthermore, *differential privacy* shows its promise in helping generate synthetic data [59, 94, 172] and emerging generative adversarial networks (GANs) [187, 65].

Cryptography-based Approaches: The third approach mainly refers to *cryptography* based techniques that totally *obfuscate* the raw data to achieve a stronger privacy guarantee (i.e., confidential-level pri-

vacy) compared to the traditional anonymization mechanisms and differential privacy mechanisms. Existing cryptographic approaches for data publishing for the purpose of training an ML can be seen along the following two directions: (i) applying traditional symmetric encryption schemes such as AES, which is associated with garbled-circuits based secure multi-party computation protocols [127, 180, 142]; (ii) applying emerging cryptosystems such as homomorphic encryption schemes [70, 175, 28, 119, 6] or functional encryption schemes [25, 74, 4, 14, 13, 2, 3]. These include necessary algorithms to carry out computation over encrypted data; here, only a party with the issued key is able to acquire the computation results. The typical PPML approaches such as thos proposed in [151, 149] could be classified to the first direction of the crypto-based data publishing approach, while more and more recent works such as proposed in [173, 191, 189, 69, 83, 39, 73, 31, 129] focus on the direction (ii).

Essentially, crypto-based data publishing approaches cannot work independently and usually are associated with related secure computation approaches, as it is expected to let the data receiver only learn the data processing result rather than the original data. These crypto-based approaches provide a promising candidate for data publishing, but the emphasis of crypto-based approaches is on computation over the encrypted data. More details related to that will be presented in the next section.

2.2.2 Data Processing Based Approaches

Based on different data/model publishing methods, data processing approaches can be as follows: *normal training* and *training using secure computation*. As discussed in Section 2.2.1, if the data is published using traditional anonymization or differential privacy mechanisms the training process is similar to that in non-PPML. Here, by privacy-preserving data processing based approach we mainly refer to the secure computation based approach that can be adopted in both training and inference phases.

Secure computation problems and the corresponding solutions were initially proposed by Andrew Yao in 1982 in a garbled-circuits protocol for two-party computation [192]. The primary goal of secure computation is to enable two or more parties to evaluate an arbitrary function over both their inputs without revealing anything to either party except for the output of the function. These secure computation approaches could be basic secure two-party computation (2PC) or more general secure multi-party computation (MPC) for multiple parties. These protocols include two types of security guarantees considering different adversarial models: *semi-honest* (passive) security and *malicious* (active) security. We refer the reader to [49, 84] for more details related to secure multi-party computation.

Here, we discuss existing secure computation techniques that have been used in PPML from the perspective of the underlying designed principle. In general, these secure computation techniques include the following: (i) pairwise blinding using perturbation or dining cryptographer networks (DC-net) or (verifiable) secret sharing; (ii) garbled-circuits with oblivious transfer; and (iii) emerging cryptographic schemes.

Pairwise Blinding based Approaches: One category of secure computation approach is the pairwise blinding using perturbation techniques, where the private values are blinded/masked with randomized values.

One type of perturbation is additive perturbation. For instance, in a generic additive perturbation based privacy-preserving summation [156] - a function-specific (i.e., the aggregation function) secure multi-party computation - the coordinator gives its input x_0 by adding a randomized perturbation r, and then each participant adds its input x_i on $x_0 + r$ and passes the result to the next participant. Finally, the coordinator acquires $\sum x_i + r$ and removes its randomized perturbation r to acquire the aggregated result. Another type of perturbation in pairwise blinding is multiplicative perturbation where the random projection or random rotation techniques are used to perturb the values. Besides, anonymous communication could also be the candidate method in the pairwise blinding based approaches. For example, DC-nets [34] or mix-nets [35] is a class of anonymous communication network, where a single participant at a time can send an anonymous message, which can be viewed as a restricted case of secure aggregation. Adopting DC-nets or mix-nets, a trusted coordinator can collect input from each party in an anonymous manner and then compute the function results, which provides some privacy-preserving feature because the coordinator cannot learn the source of each function input. Furthermore, secret sharing techniques can also be adopted in pairwise blinding based secure computation approaches. For instance, suppose that the coordinator is issued a randomized secret s, and each participant is issued secret sharing s_i to integrate into its input x_i as a perturbation. Then, the coordinator can aggregate $\sum x_i$ by removing the recovered secret s.

In short, most pairwise blinding based approaches can be viewed as a lightweight approach compared to other secure computation related approaches. As illustrated in existing proposals [9, 96, 96, 29, 56, 22], these classes of secure computing approaches are mostly adopted in traditional data mining area [9, 96] and have not been used much in recently proposed PPML approaches. Specifically, approaches proposed in [29, 56] focus on the k-means clustering machine learning algorithms. The solution in [29] proposes two types of pairwise blinding methods, namely, a *division protocol* and a *random value protocol* to perform two-party division and to sample uniformly at random from an unknown domain size. The approach proposed in [56] utilizes additive secret sharing as a cryptographic primitive to implement a secure multiparty computation protocol to do privacy-preserving clustering. The approach proposed in [22] employs a t-of-n secret sharing scheme with additional decisional Diffie–Hellman(DDH) based key agreement and authenticated encryption to construct a protocol for securely computing sums of vectors with low communication overhead, robustness to failures, which requires only one server with limited trust.

Garbled Circuits based Approaches: Garbled-circuits and oblivious transfer techniques constitute another type of foundation to build secure computation solutions. For simplicity, we use the garbled-circuits based 2PC protocol as an example to illustrate that. The basic idea is that one party (a.k.a, the garbled-circuit *generator*) prepares a circuit computing function that includes garbled gates that are encrypted via traditional symmetric encryption scheme such as AES, and then the other party (a.k.a, the garbled-circuit *evaluator*) obliviously computes the output of the circuit without learning any intermediate information. Specifically, the function f is transferred to a Boolean circuit composed of huge amounts of different types of garbled gates (e.g., AND-gate, OR-gate, and XOR-gate). Suppose that an AND-gate g^{AND} is associated

Table 2.2: Example of the AND-garbled gate table

b_i	b_j	$g_{\text{AND}}(b_i, b_j)$	garbled output		permuted garbled output
0	0	0	$E_{w_i^0, w_j^0}(w_k^0)$	$\Rightarrow$	$E_{w_i^0, w_j^0}(w_k^0)$
1	0	0	$E_{w_i^1, w_j^0}(w_k^0)$	$\Rightarrow$	$E_{w_i^1, w_j^0}(w_k^1)$
0	1	0	$E_{w_i^0, w_j^1}(w_k^0)$	$\Rightarrow$	$E_{w_i^0, w_j^1}(w_k^0)$
1	1	1	$E_{w_i^1, w_j^1}(w_k^1)$	$\Rightarrow$	$E_{w_i^1, w_j^1}(w_k^0)$

with two input wires i and j, and one output wire k. The generator first generates two cryptographic keys for each input wire, denoted as $w_i^0, w_i^1, w_j^0, w_j^1$, where the superscript represents the the encoded input bits (e.g., w_i^0 encodes 0-bit input of wire i, while w_i^1 encodes 1-bit input of wire i). For inputs data $b_i, b_j \in \{0, 1\}$, the *generator* computes the ciphertext as $\mathcal{E}^{\text{symmetric}}.E^{(k)}_{w_i^{b_i}, w_j^{b_j}}(w_k^{g^{\text{AND}}(b_i, b_j)})$, and Table 2.2 presents the gate table in detail. Then, the *evaluator* is able to acquire its input wire associated keys w_j^0, w_j^1 with its input $b_j \in \{0, 1\}$ without revealing that input to the *generator* using the 1-of-2 oblivious transfer (OT) technique. With the input associated key w^{b_j} and the received permuted garbled table, the *evaluator* is able to decrypt the corresponding ciphertext to acquire the output $w_k^{g^{\text{AND}}(b_i, b_j)}$ without learning the input of the *generator*. Finally, these different types of garbled gates can compose any function that can be used in the secure computation protocols.

Even though garbled-circuits based 2PC and MPC problems have been there for several decades, the security community is still working on improving its efficiency and practicality [127, 18, 179, 180, 97]. The garbled-circuits based 2PC or MPC have been recently adopted to address the challenge of secure computation issues in popular machine learning algorithms, even complex DL models [69, 128, 115, 151, 32]. Chameleon [149] combines the best aspects of generic secure function evaluation protocols where linear operations are achieved by additive secret sharing values and nonlinear operations are implemented by garbled-circuit protocols. Based on the efficient mixed 2PC proposed in [55], similar to the Chameleon framework, ABY^3 in [126] proposes a design and implementation of a general framework for PPML and use it to obtain new solutions for different ML algorithms in a three-server model wherein data owners secretly share their data among three servers that train and evaluate models on the joint data using three-party computation. Based on Yao's garbled-circuits, DeepSecure [151] presents a secure DL framework that is built upon automated design, efficient logic synthesis, and optimization methodologies. Recently proposed QUOTIENT [8] is a new method for discretized training of DNNs along with a customized 2PC protocol. EzPC [32] is another type of 2PC framework that generates efficient 2PC protocols from high-level, easy-to-write programs, where the proposed compiler can generate protocols that combine both arithmetic and boolean circuits for better performance.

Even though these garbled-circuits based solutions can provide provably secure and show its promise in training phase rather than just inference phase in deep neural networks without relying on the non-colluding

two or three servers as illustrated in [151, 8], the obvious limitation of the garbled-circuits based solutions is the size of the data transmission. As illustrated in Table 2.2, for a simple computation on two input bits such as $b_i \wedge b_j$, it involves transferring four ciphertexts and additional oblivious transmission for the key, where the size of each ciphertext and keys depends on the adopted symmetric encryption algorithm. Considering the complex computation functions in ML, the size of transmitted data will explode significantly.

Emerging Cryptographic based Approaches: Emerging cryptogahic approaches such as homomorphic encryption and functional encryption that support computation over the ciphertext provide other approaches toward building secure multi-party computation approaches for PPML systems. Emerging cryptography based secure computation also provides strong privacy guarantees like the garbled-circuits based approaches providing confidentiality-level security. However, unlike the garbled-circuits based secure protocols that involve huge amounts of transmitted data, these emerging cryptosystem based approaches only need to transfer the encrypted data instead of the data-encoded garbled-circuits, and related keys via oblivious transfer technique. Here, we briefly introduce *homomorphic encryption* schemes [70, 175, 28, 119, 6] and *functional encryption* schemes [25, 74, 4, 14, 13, 2, 3] that are mainly employed in existing PPML proposals [191, 80, 133, 47, 73, 31, 129, 117, 31].

Homomorphic Encryption is a form of cryptosystem with an additional evaluation capability for computing over ciphertexts without access to the private secret key, in which the result of operations over the ciphertexts, when decrypted, matches the result of operations as if they have been performed on the original plaintext. Some typical types of HE are *partially* homomorphic, *somewhat* homomorphic, *leveled fully* homomorphic, and *fully* homomorphic encryption according to the capability of performing different classes of computations. Unlike a traditional encryption scheme that includes three main algorithms: key generation (Gen), encryption (Enc), and decryption (Dec), an HE scheme also has an extra *evaluation* ($Eval$) algorithm. Formally, an HE scheme $\mathcal{E}_{\text{HE}}$ includes the above four algorithms such that

$$\mathcal{E}_{\text{HE}}.Dec_{sk}(\mathcal{E}_{\text{HE}}.Eval_{pk}(f, \mathcal{E}_{\text{HE}}.Enc_{pk}(m_1), ..., \mathcal{E}_{\text{HE}}.Enc_{pk}(m_n))) = f(m_1, ..., m_n), \tag{2.9}$$

where $\{m_1, ..., m_n\}$ is a set of messages to be protected, pk and sk are the key pairs generated by the key generation algorithm. Based on our observation, we present several commonly adopted HE implementations used in PPML solutions. The Paillier cryptosystem [135] is an additive partially homomorphic encryption system, where given the message m_i and m_j, the Paillier system $\mathcal{E}_{\text{HE}}^{\text{Paillier}}$ satisfies the equation (2.9) such that $\mathcal{E}_{\text{HE}}^{\text{Paillier}}.Enc(m_i) \cdot \mathcal{E}_{\text{HE}}^{\text{Paillier}}.Enc(m_j) = \mathcal{E}_{\text{HE}}^{\text{Paillier}}.Enc(m_i + m_j)$. HElib [79] implemented several typical fully homomorphic encryption schemes such as in [28, 40, 170, 71] with applied optimization techniques like bootstrapping, smart-vercauteren, and approximate number. SEAL [124] is another HE library that allows additions and multiplications to be performed on encrypted integers or real numbers. Other operations, such as encrypted comparison, sorting, or regular expressions, are in most cases not feasible to be done over encrypted data using this technology.

The approach proposed in [80] makes use of homomorphic encryption in constructing a protocol for regression analysis, while the approach proposed in [133] focuses on privacy-preserving ridge regression on

millions of records by combining both homomorphic encryption and garbled-circuits. The approach proposed in [47] provides a computationally secure two-party protocol based on additive homomorphic encryption that substitutes the trusted initializer. Recent work in [41] tries to evaluate the possibility of homomorphic encryption to fully implement its program in ML applications by addressing both the issue of comparison and selection/jump operations.

CryptoNets proposed in [73] tries to apply neural networks to encrypted data by employing a leveled homomorphic encryption scheme to the training data, which allows adding and multiplying encrypted messages; but it requires that one knows in advance the complexity of the arithmetic circuit. Unlike the potential ineffectiveness issue for DNNs in CryptoNets, the proposed approach in [31] combines the original ideas of Cryptonets' solution with the batch normalization principle for a classification problem. Besides, the approach proposed in [129] uses the open-source FHE toolkit HElib for DNN training using a stochastic gradient descent training method. Several recent proposals as in [50, 117, 87] focus on the same problem but with different optimization approaches to increase efficiency as well as model accuracy.

Functional Encryption is another form of cryptosystem that also supports computation over a ciphertext. Typically, an FE $\mathcal{E}_{\text{FE}}$ includes four algorithms: setup, key generation, encryption and decryption algorithms, such that

$$\mathcal{E}_{\text{FE}}.Dec_{sk_f}(\mathcal{E}_{\text{FE}}.Enc_{pk}(m_1), ..., \mathcal{E}_{\text{FE}}.Enc_{pk}(m_n)) = f(m_1, ..., m_n), \tag{2.10}$$

where the setup algorithm creates a public key pk and a master secret key msk, and key generation algorithm uses msk to generate a new functional private key sk_f associated with functionality f. Those two algorithms usually are run by a trusted third-party authority. At this moment, there is a lack of well-known libraries for FE like the HElib and SEAL libraries for the HE. Existing construction of functional encryption schemes for general functionality, such as recently proposed constructions in [74, 24, 182, 68, 30, 108], only focus on the theoretical feasibility or functionality existence. Only a few recently proposed works such as in [4, 5, 10, 20] focus on the simple and applicable FE, but the functionality is limited to the inner-products.

As presented above, the main similarity between *FE* and *HE* is that both support computation over a ciphertext. At a high-level, the main difference between functional encryption and the homomorphic encryption is that given an arbitrary function $f(\cdot)$, the homomorphic encryption allows computing *an encrypted result of* $f(x)$ from an encrypted x, whereas the functional encryption allows to compute *a plaintext result of* $f(x)$ from an encrypted x [12]. Intuitively, a function computation party in a HE scheme (i.e. the evaluation party) can only contribute its computational resources to obtain the encrypted function result, but cannot learn the function result unless it has the secret key, while the function computation party in the FE scheme (i.e., usually, the decryption party) can obtain the function result with the issued functional private key. Besides, except for several most recently proposed decentralized FE schemes [43, 3, 42], the classic FE schemes rely on a trusted third-party authority to provide a key service such as issuing a functional private key associated with a specific functionality.

Unlike HE-based secure computation techniques that have been widely adopted as a candidate solution

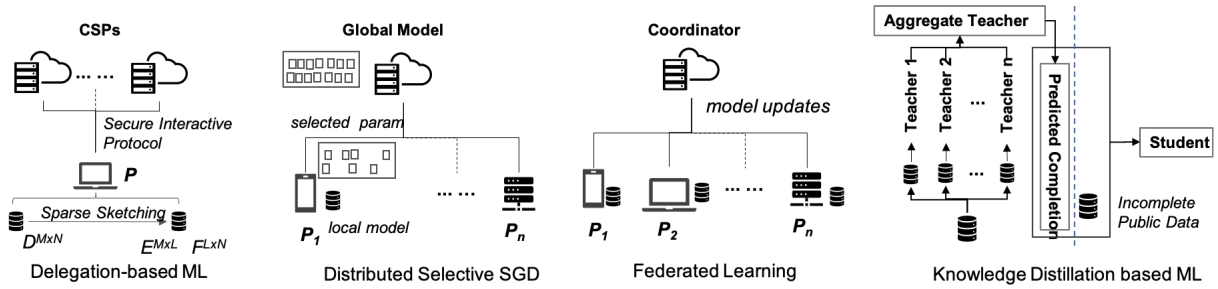


Figure 2.1: Several typical architectures adopted in existing PPML systems.

for secure computation for PPML, recently proposed FE-based PPML solutions such as in [189, 191, 153] also show its promise in terms of its efficiency and practicality. The approach proposed in [153] provides a practical framework for performing partially encrypted and privacy-preserving predictions that combines adversarial training and functional encryption. Our earlier proposed approach such as in [191] presents a CryptoNN framework that supports training a neural network model over encrypted data by using FE to construct a secure computation mechanism. An approach we have proposed as part of this dissertation (presented in Chapter 3.1), [189] focuses on PPFL by utilizing FE to construct a secure aggregation protocol to protect each participant's input in PPFL.

Remark - Impact of Encoding: Unlike in anonymized or differentially private PPML training, where the training data or trained model is in the format of floating-point numbers, the secure computation approaches using crypto-based PPML training data should be in integer format. As a result, there is typically a procedure to convert the training data into integer format before secure computation operations and then recover the computational result back into the floating-point numbers. Due to this procedure, an issue related to crypto-based secure computation is how to decide the encoding degree and what would the impact of the encoding precision be. As partially illustrated in [191, 189], the encoding issue is a trade-off problem; i.e., a higher encoding precision indicates a higher model accuracy, whereas a higher encoding precision denotes more computational time (i.e., more training time).

2.2.3 Architecture Based Approaches

Recently, architecture based approaches have also been shown to be promising for PPML, although there is no generic paradigm for architecture-based approaches. As depicted in Figure 2.1, we present four typical architectural approaches as examples to illustrate how privacy can be guaranteed for ML models.

Delegation based ML Architecture. It is a classic architecture that gives the parties with limited computational resources the capability to create and use ML models. With additional secure techniques or proper trust assumptions, a delegation-based architecture can also provide a privacy-preserving feature for ML. For

instance, CryptoML [125] proposes a practical framework that supports provably secure and efficient delegation for contemporary matrix-based ML, where a delegating client with memory and computational resource constraints is able to assign the storage and computational tasks to the cloud via an interactive delegation protocol based on provably secure Shamir's secret sharing technique. Similarly, the approach proposed in [112] includes a framework for privacy-preserving outsourced classification in a cloud computing scenario, where an evaluator can securely train a classification model over the encrypted multiple data sources with different public keys. Unlike outsourcing the computation to one party, SecureML [128] presents efficient 2PC protocols that fall in two-server model where data owners distribute their private data among two non-colluding servers that train various models on the joint data using 2PC to support PPML for linear regression, logistic regression and neural network training using the SGD method.

Distributed Selective SGD based Architecture. As proposed in [166] distributed selective SGD enables multiple parties to jointly learn an accurate neural network model without sharing their input datasets, where the system includes several participants, each of which has a local private dataset available for training, and one parameter server which is responsible for maintaining the latest values of parameters available to all parties. In particular, the approach assumes two or more participants training independently and concurrently. At each round of local training, the participants acquire the latest values of most-updated parameters and integrate the selected partial parameters into local gradients. After local training, each participant can fully control which gradients to share and how often.

Federated Learning Architecture. FL [120, 100] is also a decentralized ML approach with similar architecture as a distributed selective SGD [166], where each participant maintains a private local dataset of its facilities, and a shared global model is trained by the coordinator (i.e., a central server) using the local model updates generated by those users. Since the training data does not leave each participant's domain, FL can provide the primary privacy guarantee. In particular, a FL architecture can be treated as a general paradigm for a distributed selective SGD architecture. Usually, a FL framework requires that the participant download all gradients in the global model, train the model with the local dataset, and then upload it to the coordinator if the participant does not drop-out in current training epoch, whereas the distributed selective SGD approach gives more control to a participant to select partial parameters in the gradients. Essentially, the distributed selective SGD approach is a sketching version of a FL architecture.

Knowledge Distillation based Architecture. The basic knowledge distillation architecture is a *teacher-student* model. As illustrated in *PATE* framework [136, 138], it mainly focuses on the inference phase of an ML system. In PATE (private aggregation of teacher ensembles) approaches, the knowledge of an ensemble of "teacher" models (i.e., initially trained models) is transferred to a "student" model (i.e., model that will be used), with intuitive privacy provided by training teachers on disjoint data and strong privacy guaranteed by noisy aggregation of teachers' answers. Other typical approaches are *model transformation* and *model compression*. For instance, in MiniONN [115], an existing model is transformed into an oblivious neural network supporting privacy-preserving predictions. Existing DL models in NLP domain with a large

number of model parameters can be compressed to lightweight DL models via knowledge distillation methods [88, 143]. Except for the primary goal of reducing the size of a DL model, it can also bring additional privacy-preserving features [137, 178].

2.2.4 Hybrid Approaches

Due to the higher privacy protection requirements and recently illustrated privacy attacks such as membership inference attacks [167, 19, 90, 110, 130] and model inversion attacks [66, 186, 181, 85], existing single type of privacy-preserving approaches discussed above are not enough. For instance, existing FL frameworks only provide a basic privacy guarantee as each participant can locally hold the training data. However, the global trained model cannot prevent membership inference attacks. To address this issue, one type of hybrid approaches as proposed in [1, 72, 138] tries to integrate FL with a differential privacy mechanism. Similarly, the approach proposed in [116] shows that sketching algorithms have a unique advantage in that they can provide both privacy and performance benefits while maintaining accuracy, where the local model updates do not need to be shared; only sketched updates need to be shared instead. Furthermore, to avoid a *curious coordinator* investigating the participants' input in FL while increasing the global model performance, another type of hybrid approach as proposed in [173, 189] integrates crypto-based secure computation and differential privacy mechanism into a FL framework to provide stronger privacy guarantee. On the other hand, approaches proposed in [83, 22, 39] also utilize secure multi-party computation and FL techniques to collaboratively train an ML model over multiple vertically partitioned datasets. As we introduced in earlier for delegation-based ML system [125, 112, 128], besides the designed delegation based architecture, such a design also relies on secure computation techniques.

In short, traditional anonymization mechanisms and perturbation techniques (i.e., the differential privacy mechanism) can provide privacy protection on final trained model to avoid attacks such as membership inference attack, but cannot completely prevent a *honest-but-curious* central server that may infer private information from the participants' input in the FL framework or delegation based ML systems. Secure computation techniques such as garbled-circuits-based or crypto-based 2PC and MPC are able to protect each participant's input but cannot prevent private information leakage in the final trained model. To pursue a stronger privacy guarantee, well-designed PPML frameworks may need to rely on the integration of de-identification approaches, perturbation techniques, and secure computation within a properly designed architecture.

2.3 Transparent and Trustworthy Infrastructure

The provisioning of *openness* and *accountability*, also referred to as *transparency* in recent literature [104, 152, 102, 122, 58, 33, 64], can additionally help increase users' trust or confidence on service providers,

e.g., cryptography infrastructure, cloud infrastructure, with respect to the protection of their sensitive data.

The concept of transparency in the digital world is used to avoid malicious activities or misbehavior in the critical infrastructures. Certificate transparency proposed in [104, 102] aims to mitigate the certificate based threats caused by fake or forged SSL certificates that are mistakenly or maliciously issued by insiders. Certificate transparency model creates an open framework for monitoring the TLS/SSL certificate system and auditing specific TLS/SSL certificates. Then, several works related to certificate transparency were proposed in [64, 58, 46, 168, 103] to deal with revocation, security and privacy issues.

Most recent and related work is *CONIKS* and *transparency overlay*. CONIKS proposed in [122] deals with key transparency in end-to-end encrypted communications systems where the public keys of end users are a general version of the digital certificate. Transparency overlay propose a formal framework with a specific security proof in [33]. However, such transparency model and framework cannot deal with the trust and other issues we have addressed in this paper, e.g., how to ensure authorities' fulfillment of obligations in key services phases. To address that issue, we have proposed the notion of *authority transparency* in [190] to address similar but more complex issues related to a TPA that is the critical component of many emerging cryptosystems. To extend the notion of *authority transparency* for the entire secure computing infrastructure including the TPA, the third-party IaaS server and coordinator server, we propose the transparent and trustworthy secure computation infrastructure using emerging Ethereum blockchain techniques in this dissertation.

2.4 Summary and Discussion

In this chapter, we have presented existing privacy-preserving approaches that have been adopted in ML considering various factors, such as, various phases of ML systems and the underlying design principles; we also presented corresponding PPML approaches that have been proposed in the literature. Furthermore, we also discussed about transparent and trustworthy infrastructure for PPML approaches.

3.0 Privacy-Preserving Federated Learning using Secure Computing

While traditional ML approaches depend on a centrally managed training data set, privacy considerations have driven interest in decentralized learning frameworks in which multiple participants collaborate to train an ML model without sharing their respective training data sets. Federated learning (FL) [120, 100] has been proposed as a decentralized process that can scale to thousands of participants. Since the training data does not leave a participant's domain, FL is suitable for applications such as health care and financial services, where data sharing raises significant privacy concerns. In primary FL design, as illustrated in Figure 3.1, each participant trains a model locally and exchanges only model parameters with others, instead of sharing privacy-sensitive training data. An entity called *coordinator* merges the model parameters of different participants. Often, a coordinator is a central entity that also redistributes the merged model parameters to all participants; but other topologies have been used as well, e.g., co-locating a coordinator with each participant.

There are two types of FL, *vertical FL and horizontal FL*, that mainly differ on the information available to each participant. In *horizontal FL*, all participants have access to the entire feature set and labels, and thus, they can train their local models based on their own datasets and later share model updates with a coordinator based on their local models. The coordinator then creates a global model by averaging the model weights received from individual participants. In contrast, *vertical FL* refers to collaborative scenarios where the complete set of features and labels are *not* known to a single participant.

The primary *horizontal* FL design aims at protecting data privacy by ensuring each participant would keep its data locally and transmit model parameters [120]. Although at first glance it may provide some level of privacy, attacks reported in literature have demonstrated that it is possible to infer private information through various inference attacks: inference attacks in the learning phase have been proposed in [131]; and deriving private information from a trained model has been demonstrated in [167]. To fully protect the privacy of the training data from inference attacks, the concept of privacy-preserving federated learning (PPFL) has been raised in the recent literature. Furthermore, *vertical* FL is particularly challenging as each participant cannot train a model using its own dataset locally. Participants need to collaborate to find the complete feature vector without exposing their training data, and after aligning their data in a private way. A process to collaboratively train the model needs to take place in a privacy-preserving way without exposing the raw data of each participant.

In this chapter, we propose two PPFL frameworks - *HybridAlpha* and *FedV* [1] - focusing on the *horizontal* PPFL and *vertical* PPFL as presented in Section 3.1 and Section 3.2, respectively.

[1] Note that the *HybridAlpha* and most of work of *FedV* was accomplished while interning at the IBM Research - Almaden, and here I would like to acknowledge the continued collaboration with the team - *AI Security and Privacy Solutions* and the help from co-authors - Dr. Nathalie Baracaldo, Dr. Yi Zhou, Dr. Ali Anwar and Dr. Heiko Ludwig.

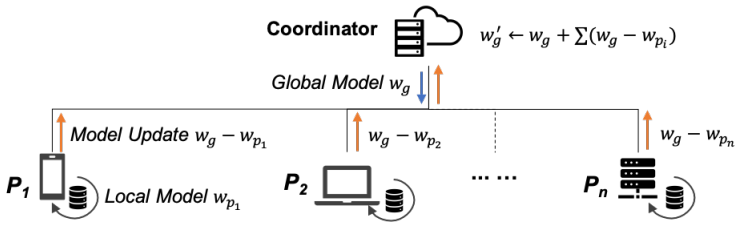


Figure 3.1: General architecture of federated learning. Note that the training process is orchestrated by a coordinator that acts as a third-party semi-trusted (a.k.a, honest-but-curious) entity and interacts with participants (i.e., $p_1, p_2, ..., p_n$). FL training is a process of repeatedly merging the local model updates, where for each iteration the coordinator first sends the global model $\boldsymbol{w}_g$ that could be randomly initialized or a pre-trained to each participant. Once $\boldsymbol{w}_g$ is received, each participant p_i trains the local model $\boldsymbol{w}_{p_i}$ using local data based on $\boldsymbol{w}_g$, and sends back the model updates. The coordinator aggregates all the received model updates to for one round of model training.

3.1 HybridAlpha: An Efficient Secure Computing Approach for Horizontal PPFL

3.1.1 Background and Motivation

Participants in a FL process cooperatively train a model by exchanging model parameters instead of the actual training data, which they might want to keep private. However, parameter interaction and the resulting model still might disclose information about the training data used. For instance, this approach still poses privacy risks such as inference attacks in the learning phase have been proposed by [131] and deriving private information from a trained model has been demonstrated in [167].

To address such privacy leakage, differential privacy [60, 62] has been proposed for a learning framework [1, 138], in which a trusted coordinator controls the privacy exposure to protect the privacy of the model's output. Similarly, [141] proposes to combine differential privacy techniques and secure multiparty computation (SMC) to support privacy-preserving analyses on private data from different data providers, whereas [22] combines secret sharing and authenticated encryption in a failure-robust protocol for secure aggregation of high-dimensional data.

Inspired from the hybrid methodology [141], a recent paper [173] also proposed a hybrid solution that provides strong privacy guarantees while still enabling good model performance. This hybrid approach combines a *noise-reduction* differential privacy approach with protection of SMC protocol, where the underlying security cornerstone is additive homomorphic encryption, i.e., threshold Paillier system [51]. Even though the hybrid approach has good model performance and privacy guarantees, it comes with long training time and high data transmission cost and cannot deal with participants dropping out during the FL process. In Table

Table 3.1: Comparison of privacy-preserving approaches in horizontal FL framework

	Threat Model		Privacy Guarantee		SMC	Features	
Proposals	participant	coordinator	computation	output	type *	communication	dynamic
[166]	honest	honest	✗	✓	–	1 round	✓
[138]	honest	honest	✗	✓	–	1 round	–
[154]	honest	HbC$^\diamond$	✓	✓	HE	2 rounds†	–
[22]	dishonest	HbC$^\diamond$	✓	✓	SS+AE	3 rounds†	dropout
[173]	dishonest	HbC$^\diamond$	✓	✓	TP	3 rounds†	✗
HybridAlpha	dishonest	HbC$^\diamond$	✓	✓	FE	1 round†	dropout/join

* "SS+AE" represents secret sharing techniques with key agreement protocol and authenticated encryption scheme; "HE" is homomorphic encryption scheme; "TP" is Threshold-Paillier system, a partially additive homomorphic encryption scheme; "FE" indicates functional encryption scheme; symbol – indicates non-comparative option.

$^\diamond$ HbC is the abbreviation of *honest-but-curious*.

† The count is based on one epoch at the training phase between the coordinator and the participant. The key distribution communication is not covered here.

3.1, we summarize existing privacy-preserving approaches for horizontal FL from the perspectives of threat model, privacy guarantees, and offered features. We believe a privacy-preserving FL framework should strive for strong privacy guarantees, high communication efficiency, and resilience to changes. As shown by Table 3.1, approaches that offer privacy guarantees incur a large number of communication rounds, substantially increasing the training time for FL systems.

To fully protect the privacy of the training data from inference attacks, it is necessary to provide the *privacy of the computation* and the *output*.

Privacy of Computation. Malicious participants involved in FL training may have an incentive to infer private information of others. Messages exchanged with the coordinator contain model updates that leak private information. For instance, if a bag of words is used as embedding to train a text-based classifier, inspecting gradients can help an adversary identify what words where used (e.g., non-zero gradients constitute words used). SMC protocols can be used to protect inference attacks at training time. These protocols ensure that individual results cannot be exposed while still allowing the computation of aggregated data.

Privacy of Output. ML models can also leak private information about the training data [66, 167, 131]. Here, adversaries can repeatedly query the model to identify if a particular observation was part of the training data. To prevent against these attacks, differential privacy has been proposed. In this case, noise is added to the model to protect individual records in the training dataset.

Limitations in Existing Approaches. Although some of them provide privacy guarantees for the computation and output, they lack relevant features for FL systems. In particular, approaches that increase the number of communication rounds can hinder the applicability of FL, as they augment the training time and amount of data exchanged. For large models such as neural networks, this is a major concern. Another important feature should be provided by FL frameworks is the support for *dynamic participation*. In some scenarios, participants may leave the training process at any time, we refer to these as *dropouts*. As shown in

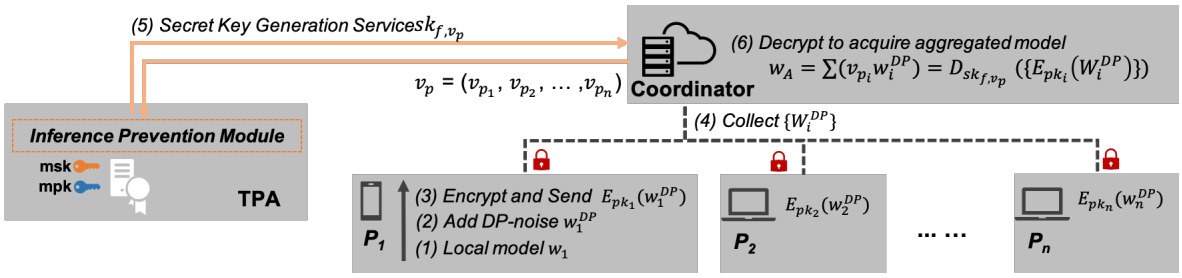


Figure 3.2: Overview of *HybridAlpha* framework. Note that we only present one epoch here. Each *participant* does the local training based on their owned dataset, and then sends out the model parameters using our proposed efficient privacy-preserving approach.

Table 3.1, existing approaches cannot gracefully deal with dropouts and require re-doing an overall training round with new keys. New participants may also join the training process at any time. Existing approaches do not provide support for this dynamic flow and require full-re-keying.

Our proposed *HybridAlpha* reduces significantly the training time by limiting the number of messages exchanged to one by round - substantially less than existing approaches that offer privacy of computation. In what follows, we present in detail some of the basic building blocks that allow us to achieve this result.

3.1.2 *Hybrid-Alpha* Framework

Figure 3.2 presents an overview of *HybridAlpha*. *Participants* want to collaboratively learn a machine learning model without sharing their local data with any other entity in the system. They agree on sharing only model updates with an coordinator. This entity is in charge of receiving model updates from multiple participants to build a common machine learning model.

Participants want to protect their data against any inference attack during the FL process and from the final model. For this purpose, they join a *HybridAlpha*, which has a *Third Party Authority (TPA)*. This entity provides a key management service that initiates the cryptosystem and provides functional encryption keys to all participants. To prevent potential leakage of information, *HybridAlpha* also includes an *Inference Prevention Module* that limits what type of functional encryption keys are provided. This module is designed to ensure that decryption keys cannot be obtained by curious coordinators and to limit potential collusion attacks. We detail this module in the rest of this section.

Threat Model. We consider the following threat model:

- *Honest-but-curious coordinator*: We assume that the coordinator correctly follows the algorithm and protocols, but may try to learn private information inspecting the model updates sent by the participants in the process. This is a common assumption [173, 22].

- *Curious and colluding participants*: We assume that participants may collude to try to acquire private information from other participants by inspecting the messages exchanged with the coordinator or the final model.
- *Trusted TPA*: This entity is an independent agency which is widely trusted by the *participants* and the *coordinator*. In real scenarios, different sectors of the economy already have entities that can take such role. For instance, in the banking industry, central banks often play a fully trusted role, and in other sectors, a third company such as a service or consultant firm can embody the TPA. We also note that assuming such trusted and independent agency is a common assumption in existing cryptosystems that have employed the TPA as the underlying infrastructure [25, 23, 77]. The TPA is in charge of holding the master private and public key. The TPA is also trusted to perform public key distribution and function derived secret key generation. Similarly, *Inference Prevention Module* is fully trusted.

We assume that secure channels are used in all communications, thus, *man-in-the-middle* and *trivial snooping* attacks are prevented. We also assume a secure key-provisioning procedure such as Diffie-Hellman is in place to protect key confidentiality. Finally, attacks that aim to create denial of service attacks or inject malicious model updates are beyond the scope of this dissertation.

Based on the threat model above, our proposed privacy-preserving framework can ensure that (i) the semi-honest coordinator cannot learn additional information except for the expected output by the differential privacy mechanism, and (ii) the malicious colluding participants cannot learn the parameters of other honest participants.

Detailed Operations. We now describe in detail the operations of *HybridAlpha* and begin by introducing the notation used. Let $\mathcal{C}$ be the coordinator and $\mathcal{S}_{\mathcal{P}}$ be a set of n participants, where each participant $\mathcal{P}_i$ holds its own dataset $\mathcal{D}_i$. We denote as $\mathcal{L}_{FL}$ the learning algorithm to be trained. Here, we first introduce the operations of the framework for non-adversarial settings, and then explain how additional features are used to protect against the inference attacks defined in the threat model section.

Non-adversarial Setting: *HybridAlpha*'s operations under non-adversarial settings are indicated in Algorithm 1. As input, *HybridAlpha* takes the set of participants, the algorithm used for training, and the differential privacy parameter ϵ.

HybridAlpha initiates via the TPA setting up keys in the system. In particular, the TPA runs the *Setup* and *PKDistribute* algorithms presented in Appendix A, so that each participant $\mathcal{P}_i$ has its own public key pk_i (see *TPA-initialization* function). We note that *HybridAlpha* allows new participants to join the training process even if it has already started. To achieve this, the TPA provisions a larger number of keys than the initial set of participants (line 3). In this way, when new participants join the training process, they need to acquire the individual public key from the TPA, and then participate in the learning protocol; all this without requiring any changes for other participants.

To begin the learning process, the coordinator $\mathcal{C}$ asynchronously queries each participant $\mathcal{P}_i$ with a query to train the specified learning algorithm $\mathcal{L}_{FL}$ and the number of participant. Then, the coordinator collects

Algorithm 1: HybridAlpha

▷$\mathcal{L}_{FL}$:= Machine learning algorithms to be trained; ϵ := privacy guarantee; $\mathcal{S}_{\mathcal{P}}$:= set of participants, where $\mathcal{P}_i \in \mathcal{S}_{\mathcal{P}}$ holds its own dataset $\mathcal{D}_i$; N := maximum number of expected participants; t := minimum number of aggregated replies;

function *TPA-initialization*$(1^\lambda, N, \mathcal{S}_{\mathcal{P}})$

  $\mathbf{mpk}, \mathbf{msk} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Setup}(1^\lambda, \mathcal{F}_N^1)$ s.t. $N \gg |\mathcal{S}_{\mathcal{P}}|$;

  foreach $\mathcal{P}_i \in \mathcal{S}_{\mathcal{P}}$ **do** $\mathbf{pk}_i \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{PKDistribute}(\mathbf{mpk}, \mathbf{msk}, \mathcal{P}_i)$;

function *aggregate*$(\mathcal{L}_{FL}, \mathcal{S}_{\mathcal{P}}, t)$

  foreach $\mathcal{P}_i \in \mathcal{S}_{\mathcal{P}}$ **do** asynchronously query $\mathcal{P}_i$ with $\text{msg}_{q,i} = (\mathcal{L}_{FL}, |\mathcal{S}_{\mathcal{P}}|)$;

  do $\mathcal{S}_{\text{msg}_{\text{recv}}} \leftarrow$ collect participant response $\text{msg}_{r,i}$ **while** $|\mathcal{S}_{msg_{recv}}| \geq t$ ***and*** *still in max waiting time*;

  if $|\mathcal{S}_{msg_{recv}}| \geq t$ **then**

    specify $v_{\mathcal{P}}$ vector; request the $\mathbf{sk}_{f,v_{\mathcal{P}}}$ from TPA;

    $\mathcal{M} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Decrypt}(\mathbf{sk}_{f,v_{\mathcal{P}}}, w_{\mathcal{P}}, \mathcal{S}_{\text{msg}_{\text{recv}}})$;

  return $\mathcal{M}$

function *participant-train*$(\epsilon, t, msg_{q,i}, \mathcal{D}_i, \boldsymbol{pk}_i)$

  $\mathcal{M}_i \leftarrow \mathcal{L}_{FL}(\mathcal{D}_i)$;

  $\mathcal{M}_i^{DP} \leftarrow \text{DP}(\epsilon, \mathcal{M}_i, t)$;

  $\text{msg}_{r,i} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Encrypt}(\mathcal{M}_i^{DP}, \mathbf{pk}_i)$;

  sends $\text{msg}_{r,i}$ to coordinator;

the responses of each participant $\mathcal{P}_i$ (see *aggregate* function).

When all responses are received, assuming there is quorum, $\mathcal{C}$ needs to request a key from the TPA corresponding to the weighted vector v_p that will be used to compute the inner product. That is, the coordinator requests private key sk_{f,v_p} from the TPA based on v_p. For computation of average cumulative sum of each participant's model, v_p can be set as $v_p = (\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n})$ *s.t.* $|v_p| = n$, where n is the number of received responses. Then, $\mathcal{C}$ updates the global model $\mathcal{M}$ by applying the decryption algorithm of the MIFE cryptosystem on collected ciphertext set $\mathcal{S}_{\text{msg}_{\text{recv}}}$ and sk_{f,v_p}. Note that here we assume the coordinator $\mathcal{C}$ will get all responses from every participant. In the case of dropouts, n can be changed so that it reflects the number of participants that are being aggregated. In the next subsection, we show how *HybridAlpha* provides recommendations to set up t so that the number of allowed dropouts are limited for security reasons.

At the participant side, when a query for training is received by participant $\mathcal{P}_i$, it trains a local model $\mathcal{M}_i$ using its dataset $\mathcal{D}_i$. During the training process[2], the participant adds differential privacy noise to the model parameters according to the procedure presented in Appendix B. Finally, $\mathcal{P}_i$ encrypts the resulting noisy model using the MIFE encryption algorithm and sends it to the coordinator (see *participant-train* function).

Inference Prevention Module: In our threat model, we assume an *honest-but-curious* coordinator that tries to infer private information during the training process. We consider multiple potential attacks where the coordinator manipulates the weighted vector to perform inference.

In particular, suppose that $\mathcal{C}$ wants to infer the model of $\mathcal{P}_i$. $\mathcal{C}$ can try to launch an inference attack to

[2]The differential privacy mechanism depends on the machine learning model being trained. For simplicity, in Algorithm 1 we show the noise added after the training process takes place. However, we note that some DP mechanisms add noise during the training process e.g., to train a neural network with the DP mechanism in [1]

obtain the model updates of participant k by setting the weighted vector as follow:

$$\forall i \in \{1, ..., n\} : \mathbf{v}_p^{'} = \begin{pmatrix} v_{p_i} = 1, \text{if } i = k \\ w_{p_i} = 0, \text{if } i \neq k \end{pmatrix} \tag{3.1}$$

If a malicious coordinator is allowed a key to perform the inner product of this vector with the model updates, the model updates of target user k would become visible; this follows because $v_p^{'}$ zeros-out the model updates of all other participants except for the ones sent by target participant k. If this is not avoided, $\mathcal{C}$ would acquire $w_i + \frac{1}{n}N(0, S^2\sigma^2)$ as the decryption result of the MIFE cryptosystem. Here, the reduced noise $\frac{1}{n}N(0, S^2\sigma^2)$ does not provide the expected privacy guarantee to protect $\mathcal{M}_i$ of $\mathcal{P}_i$ because each honest participant is injecting noise, assuming its model update is aggregated privately with other n participants.

An honest but curious coordinator may also try to create a smaller weighted vector to exclude a subset of participants from the aggregation process. In the worst case, the malicious coordinator would try to shrink the weighted vector to include one single participant to uniquely "aggregate" the model updates of that participant.

Following this same attack vector, a malicious coordinator colluding with dishonest participants may try to build a v_p vector such that: (i) a target participant model update is included in the vectors; (ii) all other honest participants model updates are not aggregated, and (iii) updates of dishonest participants are included in the aggregation process. Since the coordinator is colluding with the dishonest participants included in the aggregation process and only the target participant is included in the aggregation, the model update of the target participant is easily reconstructed (its the single unknown variable in the average equation).

To prevent such inference attacks, we propose an additional component called *Inference Prevention Module* collocated with the TPA. This module intercepts and inspects requests for private keys for given weighted vectors to prevent a curious coordinator from obtaining a key that will allow for an inference-enabling inner product.

To this end, the Inference Prevention Module takes as input a parameter t that defines a threshold on the number of non-colluding participants, where $t \geq \frac{n}{2} + 1$, that is more than half of the participants should not be colluding. By running Algorithm 2 and using parameter t, it is possible to prevent the attacks previously described. In particular, the Inference Module enforces that keys are only provided to weighted vectors that have at least t non-zero elements and that the weight for each included model update is the same.

Threshold t has an impact on the number of dropouts allowed by the system. Mainly, it helps set up the minimum quorum of participants replying to the system. *HybridAlpha* allows a limited number of participants to dropout without requiring any re-keying; only the weighted vector sent by the coordinator needs to be updated by uniquely including the weights of model updates received.

We also note that t has an impact on how much differential privacy noise is added by each participant to achieve a pre-defined ϵ. Concretely, the number of aggregated replies is always at least t, so as explain in Appendix B, the noise can be adapted to always account for t non-colluding participants contributing to the average, e.g., $N(0, \frac{1}{t}S^2\sigma^2)$. For this purpose, t needs to be communicated among all participants and

Algorithm 2: Inference Prevention Module

▷ $\boldsymbol{v}_p$:=A weighted vector to be inspected for inference attacks; t:= threshold of minimum number of dropouts and expected number of non-colluding participants;
function *inference-prevention-filter($\boldsymbol{v}_p$, t)*
  $c_{nz} \leftarrow$ count the non-zero element in $\boldsymbol{v}_p$;
  if $c_{nz} < t$ **then return** "invalid v_p" ;
  foreach *non-zero* $v_{p_i} \in \boldsymbol{v}_p$ **do**
    if $v_{p_i} \neq \frac{1}{c_{nz}}$ **then return** "invalid v_p";
  forward $\boldsymbol{v}_p$ to the TPA;

the coordinator.

Underlying ML Models of HybridAlpha: For simplicity we only use the neural networks as the underlying ML model in our FL framework for illustration and evaluation, however, the our *HybridAlpha* supports various ML algorithms. As functional encryption enables the computation of any inner-product based operation, any model that can be trained through a stochastic gradient descent (SGD)-based algorithm can be trained via our proposed *HybridAlpha*; models in this pool include SVMs, logistic regression, linear regression, Lasso, and neural networks, among others. Other models such as decision trees and random forests which require aggregating counts from each participant can also be trained by considering the counts sent to the coordinator as a vector.

3.1.3 Security and Privacy Analysis

We analyze the security and privacy of our proposed framework from three different perspectives: security offered by MIFE scheme, privacy guarantees of the framework, and prevention for different types of inference attacks.

Security of the Cryptographic Approach: The security of MIFE is critical to *HybridAlpha*, since it is the underlying infrastructure of SMC protocol that supports secure aggregation in *HybridAlpha*. In our adoption of MIFE, we add a *public key distribution* algorithm run by the TPA as a beneficial supplement of the original MIFE scheme proposed in [5] to make it applicable to our FL framework.

Specifically, the additional algorithm is only responsible for distributing each participant's respective unique public key pk_i. Unlike the original design of encryption algorithms where each participant encrypts the data using the master secret key *msk*, our encryption algorithm uses pk_i that is derived from the master keys *mpk* and *msk*. However, the core method in the encryption algorithm remains intact, and our design has no impact on other algorithms, e.g., *SKGenerate, Decrypt*. As a consequence, our adoption of MIFE does not change the security construction in the original MIFE scheme in [5]. It is then as secure as proved in [5]. To avoid redundancy, we do not present the correctness and security proofs to MIFE here, and readers can refer to [5] for more details.

Privacy of the Output Model: We provide ϵ-*differential privacy* guarantee via existing methods presented in previous works, e.g., [173, 1, 141]. These papers have shown via theoretical analysis and experimental

results that such a mechanism can achieve target privacy along with acceptable performance for the final trained model. As a consequence, our proposed framework can also achieve the same privacy guarantee for the output model as demonstrated in [173, 141].

Privacy of Computation: We exploit multi-input functional encryption as the underlying infrastructure for SMC protocol to compute the average of the weights of the participants' local trained models. As stated in Appendix A.3, the MIFE scheme is secure so that any plaintext under its protection cannot be compromised by malicious attackers. The MIFE scheme also guarantees that the decryptor, the coordinator in our FL framework, can only acquire the function results, i.e., the average weight, but not the original data, i.e., weights of the participants' local models.

Inference Attack Prevention: Next, we consider inference attacks for two adversaries: (i) a curious coordinator, and (ii) malicious or colluding participants. In Section 3.1.2, we have shown that a curious coordinator can launch an inference attack targeting a specific participant by manipulating the *weighted vector* w_p and subsequently requesting the function private key. To prevent such inference attacks, we add an additional module in TPA to filter requests for weighted vectors that are maliciously defined to isolate the reply of a single participant. Algorithm 2 verifies that at least t replies are used for aggregation, because there are at least $t > (n/2) + 1$ non-colluding participants; even if the coordinator colludes with dishonest participants he cannot isolate the reply of a target participant.

Even if an adversary manages to collect other participants' encrypted data for a possible brute-force attack, this attack is not successful. In particular, suppose that there exits a malicious participant $\mathcal{P}_i^{'}$ with its own public key $pk_{\mathcal{P}_i^{'}}$, collected encrypted data $c_j = \text{Enc}_{pk_{\mathcal{P}_{j,j\neq i}}}(m_j)$ from $\mathcal{P}_j$, and its own original data set $\mathcal{S}^{'}$. Here m_j is the corresponding plaintext of c_j, and m_j and any $m^{'} \in \mathcal{S}^{'}$ belong to a same integer group. The semantic security of the underlying MIFE scheme in our SMC protocol ensures that the adversary $\mathcal{P}_j^{'}$ does not have a non-negligible advantage to infer the original data m_j compared to the random guess. Furthermore, as we assume the existence of at least t honest participants where each participant does not share the same public key for encryption, the colluding participants cannot infer/identify private information using the output of the coordinator with their local models.

Note that based on the threat model defined in Section 3.1.2, we do not consider the DDoS attack on the coordinator where a malicious coordinator or a outside attacker will interrupt the network or replace a valid update from an honest participant.

3.1.4 Experimental Evaluation

We perform a detailed evaluation of our proposed approach to answer the following questions:

- How does *HybridAlpha* perform theoretically when compared to existing techniques that have *similar* threat models? More specifically, how many crypto-related operations can be reduced by using *HybridAlpha*?

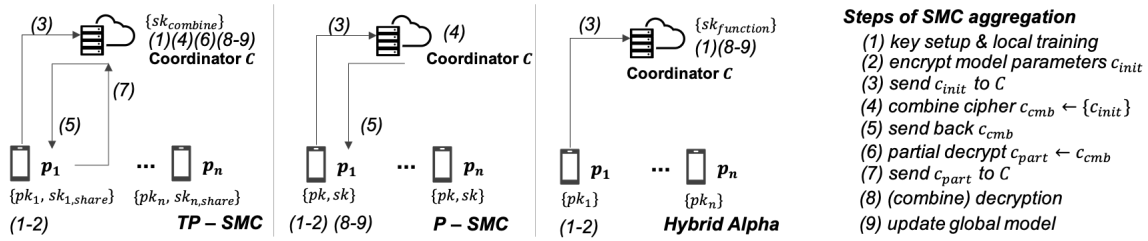


Figure 3.3: Illustration of aggregation via different crypto-based SMC solutions.

- How does our proposed SMC perform under benchmarking? How does precision setting impact computation time compared with existing techniques? What impact do different numbers of participants have?
- How does *HybridAlpha* compare to existing techniques in terms of performance efficiency?

Baselines and Theoretical Analysis. We compare the proposed *HybridAlpha* with two state of the art private-federated learning approaches: [173] and [154], which use different SMC techniques. A graphical overview and comparison of these baselines can be found in Figure 3.3, the steps performed by each approach are defined in this figure. We will use this notation to report our results. Additionally, we provide a brief description of our baselines:

- We refer to the first baseline as *TP-SMC* [173]. This FL approach uses a threshold-based homomorphic cryptosystem that allows for a trusted parameter t that specifies the number of participants that are trusted not to collude.
- We refer as *P-SMC* to our second baseline which is inspired by PySyft [154], an opensource system that uses SPDZ protocol [52, 53]. This construct supports homomorphic addition and multiplication. Because the SGD aggregation only requires addition, we opted for a additive homomorphic approach for the comparison, thus, the results reported for this baseline are representative yet faster than PySyft.

We note that the contrasted approaches follows a *similar threat model* to [173] with a honest-but-curious coordinator, and potentially colluding and malicious participants. However, they differ in the assumption of a TPA. We therefore, show how making use of a TPA, *HybridAlpha* can significantly reduce the training time of machine learning models.

Theoretical Comparison: We now theoretically compare the crypto-related communication steps associated with the contrasted approaches. Suppose that there are n participants and m coordinators in the FL framework, and the threshold for decryption of Threshold-Paillier cryptosystem is t. As shown in Table 3.2, in total, *HybridAlpha* reduces $m(n-1)$ and $m(2t-1)$ operations compared to P-SMC and TP-SMC solutions, respectively. This is achieved because *HybridAlpha* doesn't require sending back encrypted ag-

Table 3.2: The number of crypto-related operations required for each solution.

Communication	TP-SMC	P-SMC	*HybridAlpha*
Step (1)	n	n	$n+m$
Step (3)	$n \times m$	$n \times m$	$n \times m$
Step (5)	$m \times t$	$n \times m$	-
Step (7)	$t \times m$	-	-
TOTAL	$2mt + mn + n$	$2mn + n$	$mn + m + n$

gregated model updates to the participants for decryption. In the following, we also provide the details of experimental results that are consistent with the theoretical analysis.

Experimental Setup. To benchmark the performance of *HybridAlpha*, we train a convolutional neural network (CNN) with the same topology as the one used in [173] to classify the publicly available MNIST dataset of handwritten digits [106]. The CNN has two internal layers of ReLU units, and a softmax layer of ten classes with cross-entropy loss. The first layer contains 60 neurons and the second layer contains 1000 neurons. The total number of parameters of this CNN is 118110. We also use the same hyperparameters reported in previous work: a learning rate of 0.1, a batch rate of 0.01. and for differential privacy we use a norm clipping of 4.0, and an epsilon of 0.5. We used noise-reduction method as in [173] as differential private mechanism. We run experiments for 10 participants, and each participant was randomly assigned 6,000 data points from the MNIST dataset. For model quality, we used the pre-defined MNIST test set. Our implementation uses Keras with a Tensorflow backend.

Cryptosystems Implementation: We implement the contrasted cryptosystems in python based on the opensource integer group of the Charm framework [11]. Charm uses a hybrid design, where the underlying performance-intensive mathematical operations are implemented in native C modules, i.e., the GMP library [3], while cryptosystems themselves can be written in a readable, high-level language. Even though there exists Paillier implementation including its threshold variant using other programming languages, we re-implement them in a unified platform to allow for fair benchmarking and to enable easy integration with python-based machine learning frameworks such as Keras and Tensorflow.

In our implementation, we incorporated the following accelerating techniques. In *HybridAlpha*, as presented in Appendix A, the final step of MIFE decryption is to compute the discrete logarithm of an integer, which is a performance intensive computation. An example would be how to compute f in $h = g^f$, where h, g are big integers, while f is a small integer. To accelerate the decryption, we use a hybrid approach to solve the discrete logarithm problem. Specifically, we setup a hash table $T_{h,g,b}$ to store (h, f) with a specified g and a bound b, where $-b \leq f \leq b$, when the system initializes. When computing discrete logarithms, the algorithm first looks up $T_{h,g,b}$ to find f, where the complexity is $\mathcal{O}(1)$. If there is no result in $T_{h,g,b}$, the algorithm employs the traditional *baby-step giant-step* algorithm [161] to compute f, where the complexity

[3]The GNU Multiple Precision Arithmetic Library (https://gmplib.org/).

is $\mathcal{O}(n^{\frac{1}{2}})$.

The second acceleration method we implemented modifies the encryption and decryption algorithms to allow for a one-shot encryption call of a tensor. Here, each generated random nonce is applied to the whole tensor instead of a single element. We note that a further performance enhancement technique that could be used is parallelizing the encryption/decryption implementation, however, we did not include this enhancement.

Environment Setup: All the experiments are performed on a 2 socket, 44 core (2 hyperthreads/core) Intel Xeon E5-2699 v4 platform with 384 GB of RAM. Note that the FL framework is simulated (not run on the real distributed environment), hence the network latency issues are not considered in our experiment. However, we report a comparison of data transfer by contrasted approaches.

Experimental Results. Here, we first present the benchmark result of three contrasted approaches, and then show the experimental efficiency improvement.

Impact of Floating Point Precision: The parameters of a neural network (weights) are represented as floating point numbers. However, cryptosystems take them as input integers. Hence, the floating point parameters should be represented and encoded into integers. The *precision number* denotes the number of bits used after the decimal point of a floating point number. In Table 3.3 we present the impact of the precision on the computation time of each crypto-based SMC. Based on our experimental results, the precision setting has no significant impact on operation time of each cryptosystem. To be specific, the time cost of encryption, decryption, and other ciphertext computations in each cryptosystem is stable, respectively, of length of the integer.

For encryption, the average time cost of 10 participants on 118110 gradients for *HybridAlpha* is around 4 seconds, while the time cost of P-SMC and TP-SMC under the same setting is about 35 seconds. For decryption, under the same setting, the cost time of HybridAlpha is about 30 seconds, while the time cost of P-SMC and TP-SMC are 31 and 88 seconds, respectively. Note that the decryption time of TP-SMC includes the share decryption by part of participants and the final combination decryption by the coordinator, without considering network latency of transmitting the partial decrypted ciphertext. We can conclude that our proposed approach has significant advantages on both encryption/decryption time cost comparing to P-SMC and TP-SMC solutions.

Finally, the number of decimal points used in the conversion impacts the overall accuracy of the trained model. In the remaining of the experiments, we used 6-digits which allows for good model and training time performance.

Impact of Number of Participants: We also measure the impact of the number of participants on the time cost for each crypto operation. The experimental results are shown in Table 3.4. We see two different trends on the participant and on the coordinator side. At the participant side, the encryption and decryption runtime stays the same for all of the evaluated approaches as the number of participants increases. In contrast, on the coordinator side, the time cost of ciphertext multiplication increases almost

Table 3.3: The impact of precision on computation time of three SMC approaches.

	TP-SMC ▷Time (s)				P-SMC Time (s)			*HybridAlpha* Time (s)	
precision	enc_{avg}	ct_{fuse}	$dec_{share,avg}$	$dec_{combine}$	enc_{avg}	ct_{fuse}	dec	enc_{avg}	dec
2	35.120	2.586	61.080	28.465	35.752	2.269	32.042	4.157	30.075
3	35.675	2.604	61.929	28.202	35.725	2.369	31.574	4.158	30.512
4	35.841	2.571	60.832	28.324	35.821	2.387	31.856	4.110	29.865
5	35.767	2.635	60.369	28.816	35.857	2.493	31.625	4.075	30.149
6	35.724	2.578	60.326	28.286	35.985	2.532	31.587	4.095	30.803

▷ The threshold parameter of Threshold-Paillier encryption system is set to half the number of participants.

Table 3.4: The impact of participant numbers on computation time of three SMC approaches.

	TP-SMC Time (s) ▷				P-SMC Time (s)			*HybridAlpha* Time (s)	
participants	enc_{avg}	ct_{fuse}	$dec_{share,avg}$	$dec_{combine}$	enc_{avg}	ct_{fuse}	dec	enc_{avg}	dec
6	35.968	*1.375*	60.555	*22.184*	35.934	*1.332*	31.616	4.241	*20.246*
8	35.375	*1.843*	60.820	*23.980*	36.039	*1.859*	31.611	4.092	*25.349*
10	35.693	*2.358*	60.988	*28.401*	36.847	*2.611*	32.197	4.077	*31.782*
12	35.685	*2.759*	60.947	*34.684*	36.142	*2.959*	31.588	4.091	*36.884*
14	35.688	*3.215*	60.965	*39.838*	35.932	*3.330*	31.503	4.126	*42.683*
16	35.721	*3.694*	60.917	*46.849*	36.533	*4.481*	32.020	4.059	*47.435*
18	35.683	*4.170*	60.879	*53.441*	36.628	*5.368*	32.996	4.594	*56.519*
20	35.697	*4.764*	60.816	*97.224*	36.743	*5.765*	31.923	4.147	*59.823*

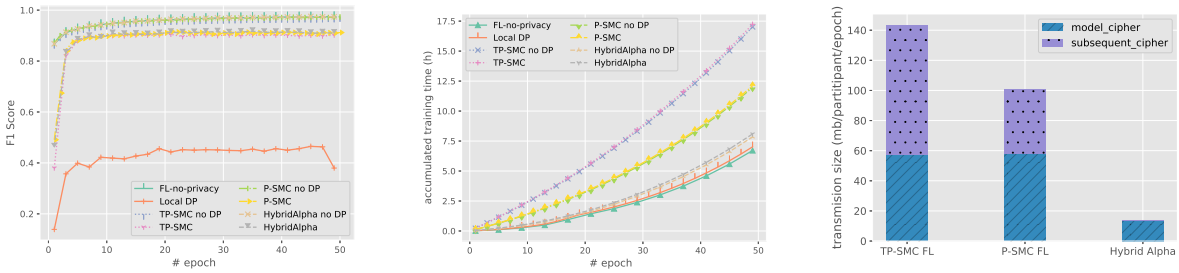
▷ The threshold parameter of TP-SMC is set to half the number of participants.

Table 3.5: Impact of threshold for TP-SMC on computation time.

threshold†	enc_{avg} time (s)	ct_{fuse} time (s)	$dec_{share,avg}$ time (s)	$dec_{combine}$ time (s)
2	35.577	2.602	60.736	12.700
4	35.697	2.592	60.420	23.293
6	35.713	2.625	60.238	34.427
8	36.054	2.623	60.767	46.462
10	35.880	2.626	60.650	58.293

† The total participants number is set to 10 and the precision number is set to 6.

linearly with the increase in the number of participants (shown in italicized numbers in Table 3.4). However, we note a significant difference between *HybridAlpha* and TP-SMC. For *HybridAlpha* the decryption time increases approximately linearly with the increase of participants, while for TP-SMC, the decryption time increases exponentially as the number of participants increases. Focusing on the TP-SMC, we also evaluate the impact of threshold t, which indicates the minimum number of participants who are required to do partial decryption. As shown in Table 3.5, only the final decryption has significant relationship with threshold t. For the same number of participants, the cost time of decryption increases linearly as the threshold number



(a) F1 score (b) training time (c) transmitted ciphertext size

Figure 3.4: Performance comparison in model accuracy, time efficiency and data transmission.

increase.

Model Quality, Training Time and Data Transmission: In this experiment, we evaluate the performance of *HybridAlpha* with respect to multiple techniques to perform FL. In particular, we assess the quality of models produced and the total training time. The contrasted approaches for this experiment include the following additional baselines: (i) "FL-no-privacy", where the neural network is trained without privacy considerations. This method provides a baseline for maximum possible performance in terms of model quality; (ii) "Local DP", where each participant applies differential privacy locally according to [95]; (iii) for "TP-SMC", "P-SMC" and "*HybridAlpha*", we report the results for two cases: adding differential privacy to protect privacy of the output and without adding differential privacy. When no differential privacy is added, we use "TP-SMC no DP", "P-SMC no DP" and "*HybridAlpha* no DP". For privacy-preserving approaches we use an $\epsilon = 0.5$. Finally, our experiments used $t = 5$ for *HybridAlpha* and TP-SMC. This experiment was run with 10 participants.

To measure quality of model performance, we report the F1-score (a measure that combines precision and recall) of the resulting models. The results are presented in Figure 3.4a. We see different trends depending on whether a particular approach protects privacy of the computation and of the output. As expected, approaches that do not protect the privacy of the final model - those that don't inject differential privacy noise- result in a higher F1-score. In contrast, "Local DP" provides the lowest F1-score due to the high amount of noise injected by each participant. For approaches that use SMC to uniquely protect the privacy of the computation, "TP-SMC no DP", "P-SMC no DP" and "*HybridAlpha* no DP", we see higher F1-scores than for those that protect the privacy of the output. This shows the price of protecting against the risk of inference on the model. Finally, we see that approaches that combine differential privacy with SMC are capable of achieving higher F1-scores while protecting the privacy of the input and output.

We now analyze these approaches from the perspective of total training time presented in Figure 3.4b. As it can be seen, our proposed *HybridAlpha* has very similar training time to "FL-no-privacy". In other words, the training time added by ensuring privacy of the input and output is negligent. In contrast, we

see that the slowest approach is TP-SMC even though we set up t to a conservative 50% of the entire number of participants in the system. This result is due to the fact that TP-SMC requires more rounds of communication per global step. The high training time makes TP-SMC suitable for models that require limited number of interactions with the coordinator during training.

Beside the efficiency in training time, we also evaluate the efficiency of network transmission by measuring the volume of encrypted parameters transmitted over the network. In Figure 3.4c, we present the total transmitted ciphertext size under different crypto-based SMC approaches for one epoch. The blue bar represents initial ciphertext size of model parameters, while the spotted purple bar indicates the size of subsequent ciphertext, including multiplied cipher, and partially decrypted ciphers. We can see that *HybridAlpha* provides the lowest transmission rate because it only performs one round of communication on encrypted data without any subsequent ciphertext transmission. Also, our proposed approach has smaller ciphertext size of initial parameters compared to contrasted approaches.

3.2 FedV: PPFL over Vertically Partitioned Data

3.2.1 Background and Motivation

Vertical FL is a powerful approach that can help create ML models for many real-world problems where a single entity does not have access to all the training features or labels. For example, in healthcare, different entities may collect different sets of data about patients that if combined can help achieve significantly improved accuracy of prediction and diagnosis of health conditions of patients. In particular, a sensor company may collect readings of body sensors, including heart rate and sleeping cycle data for a patient. A second participant may be a hospital that records emergency room visits and medical history of the patient. A third participant can be an insurance company that maintains the patient's personal disease history and approvals or denials of insurance requests (labels). As a second example, consider a set of banks and a regulator. These banks may want to collaboratively create an ML model using their datasets to flag individuals who commit money-laundering. Such a collaboration is important as criminals typically use multiple banks to avoid detection. However, if several banks partner together to find a common vector for each client and a regulator, such as the Federal Trade Commission (FTC) provides the labels that show which clients have committed money laundering, such fraud can be identified and/or mitigated. However, note that each bank may not typically want to share its clients' account details and in some cases it is even forbidden to do so. Further, the regulator may also want to maintain labels private.

One of the requirements for privacy-preserving VFL is thus to ensure that the dataset of each participant, including the labels, are private. VFL requires two different processes: *entity resolution* and *vertical training*. Both these processes are orchestrated by a *Coordinator*, which acts as a third semi-trusted participant that interacts with each participant. Before we present the detailed description of each process, we introduce the

notation used throughout the section.

Notation: Let $\mathcal{P} = \{p_i\}_{i\in[n]}$ be the set of n participants in VFL. Let $\mathcal{D}^{[X,Y]}$ be the training dataset, where $X \in \mathbb{R}^d$ represents the feature set and $Y \in \mathbb{R}$ denotes the labels across the set of participants $\mathcal{P}$. Except for the identifier features, there is no overlapping *training* features between any two participants' local datasets, and these datasets can form the "global" dataset $\mathcal{D}$. In VFL, we assume that only one participant has the class labels, namely, the *active participant*, while other participants are *passive participants*. For simplicity, in the rest of the section, let p_1 be the *active participant*. The goal of VFL is to train a ML model $\mathcal{M}$ over the dataset $\mathcal{D}$ from the participant set $\mathcal{P}$ without leaking each participant's data.

Private Entity Resolution (PER): In VFL, unlike in a centralized ML scenario, $\mathcal{D}$ is distributed across multiple participants. Before training takes place, it is necessary to *'align'* the records of each participant without revealing its data. This process is known as entity resolution [45]. After the entity resolution step, records from all participants are linked to form the complete *training samples*.

Ensuring that the entity resolution process does not lead to inference of private data of each participant is paramount in VFL. A curious participant should not be able to infer the presence or absence of a record. Existing approaches, such as [134, 89], use a bloom filter and random oblivious transfer [57, 99] with a shuffle process to perform private set intersection. This helps in finding the matching record set while preserving privacy. We assume there exists shared *record identifiers*, such as names, dates of birth or universal identification numbers, that can be used to perform entity matching. In *FedV*, we employ the anonymous linking code technique called *cryptographic long-term key (CLK)* and matching method called *Dice coefficient* [159] to perform PER, as has been done in [83]. As part of this process, each participant generates a set of CLK based on the identifiers of the local dataset and shares it with a *coordinator* who matches the CLKs received and generate a permutation vector for each participant to *shuffle* its local dataset. The shuffled local datasets are now ready to be used in the private vertical training phase.

Private Vertical Training: After the private entity resolution process takes place, the *coordinator* dictates which samples in a PER-processed training batch of each participant will be used to train a model. In the following, we discuss the challenge of private gradient descent training process in detail.

As the subsets of the feature set are distributed among different participants, gradient descent(GD)-based methods need to be adapted to such vertically partitioned settings. We now explain how and why this process needs to be modified. GD method [132][Section 1.2.3] is a class of optimization algorithms to find the minimum of a target loss function; for example, in machine learning domain, a typical loss function can be defined as follows,

$$E_{\mathcal{D}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{w})) + \lambda R(\boldsymbol{w}), \tag{3.2}$$

where $\mathcal{L}$ is the loss function, $y^{(i)}$ is the corresponding class label of data sample $\boldsymbol{x}^{(i)}$, $\boldsymbol{w}$ denotes the model parameters, and R is regularization term with coefficient λ. GD finds a solution of (3.2) by iteratively

moving in the direction of the locally steepest descent as defined by the negative of the gradient, i.e.,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla E_{\mathcal{D}}(\boldsymbol{w}), \tag{3.3}$$

where α is the learning rate, and $\nabla E_{\mathcal{D}}(\boldsymbol{w})$ is the gradient computed at the current iteration. Due to its simple algorithmic scheme, GD and its variants, like SGD, have become the common approaches to find the optimal parameters (a.k.a. the weights) of a ML model based on $\mathcal{D}$. In a VFL setting, since $\mathcal{D}$ is vertically partitioned among participants, the gradient computation $\nabla E_{\mathcal{D}}(\boldsymbol{w})$ is more computationally involved than in a centralized machine learning setting.

Considering the simplest case where there are only two participants $\{p_A, p_B\}$ in a vertical federated learning system, and the target loss function is MSE (Mean Squared Loss), i.e., $E_{\mathcal{D}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w}))^2$, we have

$$\nabla E_{\mathcal{D}}(\boldsymbol{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w}))\nabla f(\boldsymbol{x}^{(i)}; \boldsymbol{w}). \tag{3.4}$$

To compute the first term of (3.4), $-y^{(i)}\nabla f(\boldsymbol{x}^{(i)}; \boldsymbol{w})$, we need feature information from both p_A and p_B, and labels from p_B. And clearly, $\nabla f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) = [\partial_{\boldsymbol{w}_A} f(\boldsymbol{x}_A^{(i)}; \boldsymbol{w}); \partial_{\boldsymbol{w}_B} f(\boldsymbol{x}_B^{(i)}; \boldsymbol{w})]$ does not always hold for any function f, since f may not be well-separable w.r.t. $\boldsymbol{w}$. Even when it holds for linear functions like $f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) = \boldsymbol{x}^{(i)}\boldsymbol{w} = \boldsymbol{x}_A^{(i)}\boldsymbol{w}_A + \boldsymbol{x}_B^{(i)}\boldsymbol{w}_B$, (3.4) will be reduced to

$$\begin{aligned}
\nabla E_{\mathcal{D}}(\boldsymbol{w}) &= -\frac{2}{n}\sum_{i=1}^{n}(y^{(i)} - \boldsymbol{x}^{(i)}\boldsymbol{w})[\boldsymbol{x}_A^{(i)}; \boldsymbol{x}_B^{(i)}] \\
&= -\frac{2}{n}\sum_{i=1}^{n}\left([y^{(i)}\boldsymbol{x}_A^{(i)}; y^{(i)}\boldsymbol{x}_B^{(i)}] + (\boldsymbol{x}_A^{(i)}\boldsymbol{w}_A + \boldsymbol{x}_B^{(i)}\boldsymbol{w}_B)[\boldsymbol{x}_A^{(i)}; \boldsymbol{x}_B^{(i)}]\right) \\
&= -\frac{2}{n}\sum_{i=1}^{n}\left([(y^{(i)} - \boldsymbol{x}_A^{(i)}\boldsymbol{w}_A - \boldsymbol{x}_B^{(i)}\boldsymbol{w}_B)\boldsymbol{x}_A^{(i)}; (y^{(i)} - \boldsymbol{x}_A^{(i)}\boldsymbol{w}_A - \boldsymbol{x}_B^{(i)}\boldsymbol{w}_B)\boldsymbol{x}_B^{(i)}]\right),
\end{aligned} \tag{3.5}$$

which may lead to training data exposure between two participants due to the computation of some terms (colored in red) in (3.5). Under the VFL setting, the gradient computation at each training epoch relies on (i) the participants' collaboration to exchange their "partial model" with each other, or (ii) exposing their data to the coordinator to compute the final gradient update. Therefore, any naive solutions will lead to a huge risk of privacy leakage, which will be against the initial goal of the federated learning proposals that primitively protects the data privacy.

In summary, existing approaches to train ML models in vertical FL, e.g., [69, 83, 39, 169], are model-specific and rely on the hybrid general (garbled circuit based) secure multi-party computation (SMC) or partially additive homomorphic encryption (HE) (i.e., Paillier cryptosystem [51]). These approaches have several limitations. First, they require the use of Taylor series approximation to train non-linear ML models, such as logistic regression, possibly reducing the model performance. Furthermore, the prediction and inference phases of these vertical FL solutions also rely on the approximation-based secure computation, and hence, they cannot predict as accurately as does a centralized ML model. Secondly, using such cryptosystems

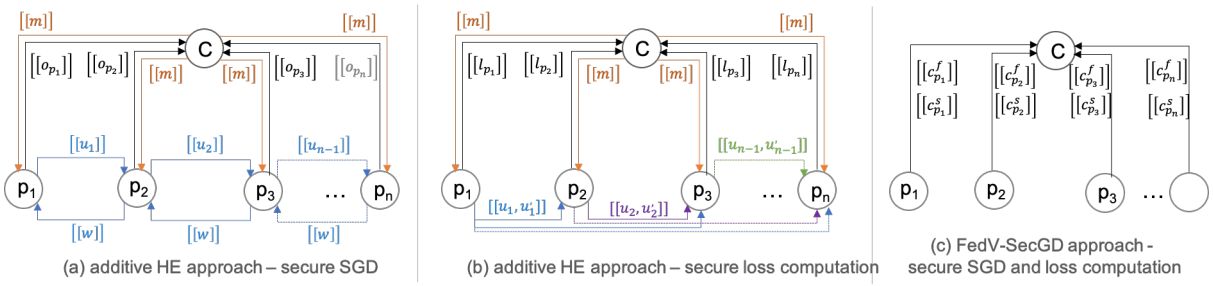


Figure 3.5: Comparison of secure aggregation communications via different approaches.

as part of the training process increases substantially the training time. Thirdly, these protocols require a large number of peer-to-peer communication rounds among participants as illustrated in Figure 3.5 (a) and (b) making it difficult to deploy them in systems that have unreliable connectivity or require limited communication for security reasons; for instance, HIPAA limits the connectivity of data centers to a very few specific entities. Note that figure (a) and (b) show the secure SGD and loss computation proposed in [83] while figure (c) shows our proposed *FedV* approach which can be applied to both secure SGD and secure loss computation. [83] only supports two participants with one coordinator, and the illustration here is our theoretical extension for the multiple participant scenario based on their proposed algorithm. Our approach requires a one-shot messaging and eliminates peer-to-peer communication. Finally, other approaches such as [193] require sharing class distributions which may lead to potential leakage of private information of each participant.

To address these limitations, in this section, we propose *FedV*, a framework that substantially reduces the amount of communication required to train ML models in a vertical FL fashion. *FedV* does not require any peer-to-peer communication among participants and can work with gradient-based training algorithms, such as stochastic gradient descent and its variants, to train a variety of ML models, e.g., logistic regression, support vector machine (SVM), etc. To achieve these benefits, *FedV* cleverly orchestrates multiple functional encryption techniques [4, 5], that are non-interactive in nature, speeding up the training process with respect to the state-of-the-art approaches. Additionally, *FedV* supports more than two participants and allows participants to dynamically join and leave without a need for re-keying. This feature is not provided by garbled-circuit or homomorphic encryption based encryption techniques utilized by state-of-the-art approaches.

3.2.2 *FedV* Framework

FedV enables vertical federated learning without a need for any peer-to-peer communication resulting in a drastic reduction in training time and total data transfer as shown in Figure 3.6. We first overview

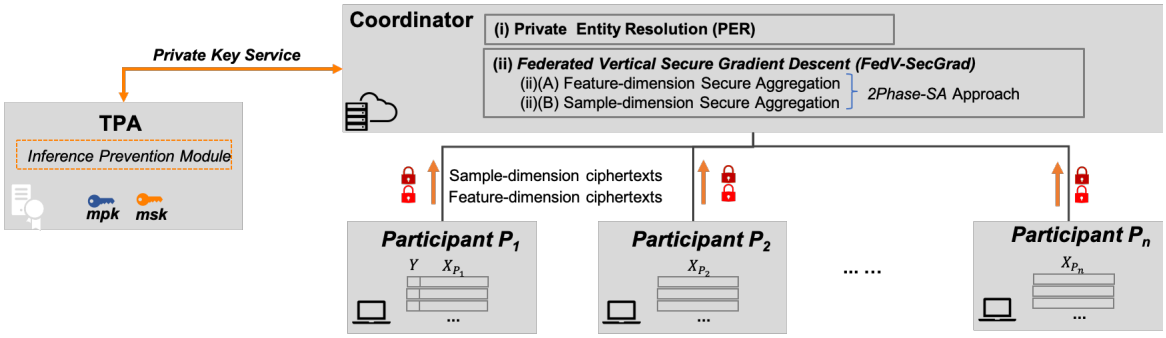


Figure 3.6: Overview of the *FedV*. Note that we assume participant p_1 owns the labels, while all other participants (i.e., $p_2, ..., p_n$) are passive participants.

the entities in the system and explain how they interact under our proposed two-phase secure aggregation technique that makes these results possible.

FedV has three types of entities: a *coordinator*, a set of *participants* and a *third-party authority*.

- Each *participant* owns a training dataset which contains a subset of features and wants to collaboratively train a global model. As in [39], we divide *participants* into two categories: (i) one *active participant* who has training samples with partial features and the class labels, represented by p_1 in Figure 3.6; (ii) multiple *passive participants* who have training samples with only partial features.
- The *coordinator* orchestrates the *private entity resolution* procedure and coordinates the training process among the participants.
- To enable functional encryption, *FedV* includes a *TPA* that is responsible for setting up the underlying cryptosystem, delivering the public key to each *participant* and providing private key service to the *coordinator*. The TPA is in charge of holding the master private and public key. It is also trusted to perform public key distribution and to generate a functionally derived secret key. In real-world scenarios, different sectors of the economy already have entities that can take the role of a TPA. For example, central banks of the banking industry often play a fully trusted role, and some third companies in other sectors such as a service or consultancy firm can embody the TPA. As it will become apparent in our security evaluation, in *FedV the TPA does not have access to neither the dataset nor the ML model.*

To perform the private entity resolution, we make use of the approach proposed in [159] which was described in Section 3.2.1. This approach is then expanded to include a set of random vectors one for each participant to re-shuffle their dataset similarly to [83]. At the end of the entity resolution, the coordinator generates permutation vector π_i for each participant to shuffle its local records. This results in participants having all records aligned before training. Finally, prior to the training process, all participants also agree upon a random seed that will be used to generate a one-time-password sequence [81]. During the training

Algorithm 3: *FedV* Framework

Initialize cryptosystems and deliver public keys and a random seed r for each participant;
Each participant shuffles samples according to the entity resolution vectors $\pi_1, .., \pi_n$;
$\boldsymbol{w} \leftarrow$ random initialization;
repeat
  foreach *mini-batch* $\mathcal{B} \in \mathcal{D}$ **do**
    $\nabla E_{\mathcal{B}}(\boldsymbol{w}) \leftarrow$ execute *FedV-SecGrad* with batch index s_i;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla E_{\mathcal{B}}(\boldsymbol{w})$
  $E_{\mathcal{B}}(\boldsymbol{w}) \leftarrow$ secure loss computation;
  if $E_{\mathcal{B}}(\boldsymbol{w})$ *is stable for a while* **then** break;
until *reach maximum iteration*;
return $\boldsymbol{w}$

process, at a given round, each participant will use the one-time-password associated with the training epoch as the seed to randomly select the samples that are going to be included in a batch for the given round. The random seed could be generated by the TPA and is only shared among participants that do not play the role of coordinator (this is done to prevent inference attacks).

After the private entity resolution, *FedV* starts the *Federated Vertical Secure Gradient Descent* (*FedV-SecGrad*) operation which is the core novelty of this paper. *FedV-SecGrad* is a two-phased secure aggregation operation that enables the computation of gradients by uniquely requiring a single message to be exchanged between participants and the coordinator. For this purpose, participants perform a sample-dimension and feature-dimension encryption. The resulting cyphertexts are then sent to the coordinator, which in turn generates the aggregation vectors to compute the inner products. These aggregation vectors are subsequently sent to the TPA that verifies their validity (we will expand on this later). If the aggregation vectors are concluded to be adequate, the TPA provides the cryptographic key to the coordinator to perform the inner products. As a result of these computations, the coordinator can obtain the gradients.

Threat Model and Assumptions. We consider the following threat model:

- *Honest-but-curious coordinator*: We assume that the coordinator correctly follows the algorithms and protocols, but may try to learn private information from the aggregated model updates. This is a common assumption as claimed in related work [22, 173]. The *coordinator* does not have to be a separate entity in *FedV*. Any *active participant* following an *honest-but-curious* behavior can play the role of *coordinator*. Just like the coordinator, active participants do not get access to random seed used to select data points in each batch.
- *Trusted TPA*: As a critical component in the underlying cryptosystem infrastructure, the TPA is an independent entity which is trusted by *participants* and the *coordinator*. Assuming such a trusted and independent entity is common in existing cryptosystems such as [4, 5].
- *Participants*: We assume a limited number of *dishonest* participants who may try to infer the honest participants' private information. Dishonest participants may collude with each other.

Our proposed *FedV* can guarantee that an *honest-but-curious* coordinator cannot learn additional in-

formation beyond the excepted gradient updates. The coordinator and TPA are assumed not to collude. Additionally, the coordinator and participants do not collude. We note that the TPA does not have access to the training data. A detailed security analysis is presented in Section 3.2.3.

We assume secure channels are in place and hence *man-in-the-middle* and *snooping* attacks are not feasible. Similarly, secure key distribution is assumed to be in place. Finally, denial of service attacks and backdoor attacks where participants try to cause the final model to create a targeted miss classification [36, 15] are outside the scope of this paper.

Vertical Training Process - *FedV-SecGrad*. We now present in detail our *federated vertical secure gradient descent (FedV-SecGrad)* approach for supported ML models. First we provide a high level overview of the secure aggregation operations follow by *FedV-SecGrad* and describe in detail the operations performed by *FedV-SecGrad* in Procedure 4 for linear models. Then, we extend *FedV-SecGrad* to other popular ML models and discuss the Inference Prevention Module (IPM).

We describe *FedV* and its supported ML models in Lemma 1.

Lemma 1. *FedV-SecGrad is a generic approach to perform private vertical federated learning which adopts gradient-based algorithms to train a machine learning model with prediction function that can be written in the form of* $f(\boldsymbol{x};\boldsymbol{w}) := g(\boldsymbol{w}^\intercal \boldsymbol{x})$*, where* $\boldsymbol{x}$ *and* $\boldsymbol{w}$ *denote the feature vector and the model weight vector, respectively. And the loss function is either mean-squared loss or cross-entropy loss depends on the problem context, i.e.,* $\mathcal{L} := \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(\boldsymbol{x}^{(i)};\boldsymbol{w}))^2$ *or* $\mathcal{L} := \frac{1}{n}\sum_{i=1}^{n}[-y^{(i)}\log(f(\boldsymbol{x}^{(i)};\boldsymbol{w})) - (1-y^{(i)})\log(1-f(\boldsymbol{x}^{(i)};\boldsymbol{w}))]$.

Lemma 1 covers ML models besides linear models. Specifically, *FedV-SecGrad* supports non-linear ML models such as logistic regression and SVMs. We demonstrate how this is achieved later. Note that in Lemma 1, we deliberately omitted the regularizer R commonly used in ML, see equation (3.2), because regularizers only depend on model weights $\boldsymbol{w}$ so that it can be computed by the coordinator independent of the dataset of each participant.

For simplicity, we first use linear models, where g is the identity function and the loss is a mean-squared loss, as an example to walk through *FedV-SecGrad*. The target loss function then becomes

$$E_{\mathcal{D}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)})^2. \tag{3.6}$$

We observe that during an SGD training round the gradient computations over vertically partitioned data $\nabla E_{\mathcal{D}}(\boldsymbol{w})$ can be reduced to two types of operations: (i) *feature-dimension aggregation* and (ii) *sample/batch-dimension aggregation*. To perform these two operations, *FedV-SecGrad* follows a *two-phased secure aggregation (2Phased-SA)* process. Specifically, the *feature dimension SA* securely aggregates several batches of training data that belong to different participants in feature-dimension to acquire the value of $y^{(i)} - \boldsymbol{x}^{(i)}\boldsymbol{w}$ for each data sample as illustrated in (3.5), while the *sample dimension SA* can securely aggregate one batch of training data owned by one participant in sample-dimension with the weight of $y^{(i)} - \boldsymbol{x}^{(i)}\boldsymbol{w}$ for each sample, to obtain the batch gradient $\nabla E_{\mathcal{B}}(\boldsymbol{w})$. The communication between the *participants* and the *coordinator* is a one-way interaction requiring a single message.

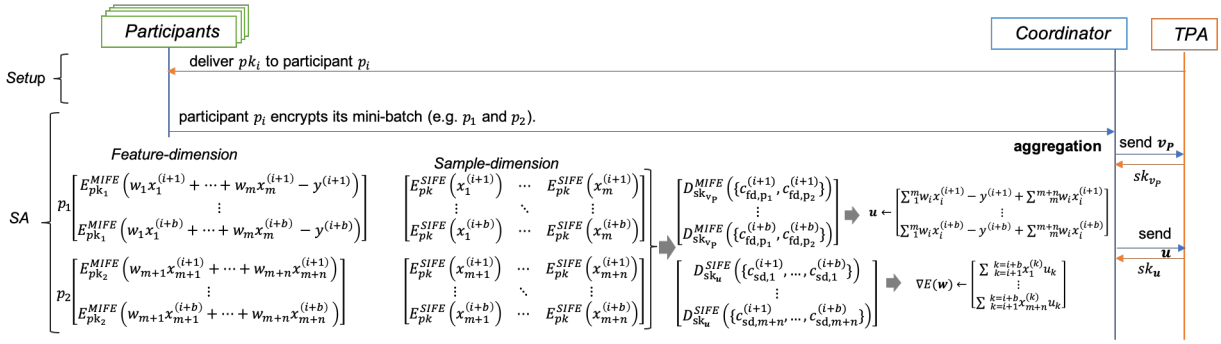


Figure 3.7: Illustration of the *FedV-SecGrad* protocol.

The active participant and all other passive participants perform slightly different pre-processing steps before invoking *FedV*. The active participant p_i appends a vector with labels y to obtain $\boldsymbol{x}_{p_i}^{(i)}\boldsymbol{w}_{p_i} - y$ before encryption. In the case where the active participant does not have any feature data of its own but only labels, it needs to flip the sign of the labels which serves as input to the *FedV-SecGrad* for the feature dimension SA. For each passive participant p_j, they only encrypt their $\boldsymbol{x}_{p_j}^{(i)}\boldsymbol{w}_{p_j}$.

Figure 3.7 illustrates the proposed protocol for a simple case where only two participants with active participant being p_1 and passive participant being p_2. Assume that the training batch size is b and that the current training batch samples for p_1 and p_2 are $\mathcal{B}_{p_1}^{b\times m}$ and $\mathcal{B}_{p_2}^{b\times n}$:

$$\begin{matrix} & & \mathcal{B}_{p_1}^{b\times m} & & & \mathcal{B}_{p_2}^{b\times n} & \\ \begin{bmatrix} y^{(i+1)} \\ \vdots \\ y^{(i+b)} \end{bmatrix} & \begin{bmatrix} x_1^{(i+1)} & \dots & x_m^{(i+1)} \\ \vdots & \ddots & \vdots \\ x_1^{(i+b)} & \dots & x_m^{(i+b)} \end{bmatrix} & & \begin{bmatrix} x_{m+1}^{(i+1)} & \dots & x_{m+n}^{(i+1)} \\ \vdots & \ddots & \vdots \\ x_{m+1}^{(i+b)} & \dots & x_{m+n}^{(i+b)} \end{bmatrix} & & \end{matrix}$$

Feature dimension SA. The goal of *feature dimension SA* is to securely aggregate the sum of a group of inputs from multiple participants without disclosing the inputs to the *coordinator*. Taking the $(i+b)$-th data sample as an example, the coordinator is able to securely aggregate $\sum_{k=1}^{m} w_k x_k^{(i+b)} - y^{(i+b)} + \sum_{k=m+1}^{m+n} w_k x_k^{(i+b)}$. To perform this operation, the coordinator prepares an aggregation functionality vector $\boldsymbol{v}_{\mathcal{P}}$ (e.g., $(1,1)$) and sends it to the TPA to request a function key $\mathrm{sk}_{\boldsymbol{v}_{\mathcal{P}}}^{\mathrm{MIFE}}$.

Each participant p_i encrypts each 'partial model' in the batch sample using the MIFE encryption algorithm with its public key pk_i. Then, each p_i sends the encrypted batch sample to the coordinator. With the received key $\mathrm{sk}_{\boldsymbol{v}_{\mathcal{P}}}^{\mathrm{MIFE}}$, the coordinator can decrypt the collected set of encrypted batch samples. The resulting decryption is the aggregated sum of the elements of $\boldsymbol{w}_{p_1}^{m\times 1} D_{p_1}^{b\times m} - \boldsymbol{y}^{1\times b}$ and $\boldsymbol{w}_{p_2}^{n\times 1} D_{p_2}^{b\times n}$ in the feature dimension.

It is easy to extend the above protocol to a general case with k participants. In this case, the weight vector $\boldsymbol{v}_{\mathcal{P}}$ can be set as an all-one vector $(1, 1, ..., 1)$ with k elements indicating that the coordinator has

received the replies from all participants and wants to weight all replies equally in the feature dimension aggregation. In scenarios where only a subset of participants have replied, it is possible to securely aggregate the results by adapting $\boldsymbol{v}$, accordingly. We will discuss this case in more details later.

Sample dimension SA. The goal of the *sample dimension SA* is to securely aggregate the batch gradient. For example, considering the first feature weight for data sample owned by p_1, the coordinator is able to securely aggregate $\nabla E_{\mathcal{B}}(w_1) = \sum_{k=i+1}^{k=i+b} x_1^{(k)} u_k$ via *sample dimension SA* where $\boldsymbol{u}$ is the aggregation result of *feature dimension SA* discussed above. This SA protocol requires the participant to encrypt its batch samples using the SIFE cryptosystem with its public key pk, as shown in Figure 3.7. Then, the *coordinator* exploits the results of the *feature dimension SA*, i.e., an element-related weight vector $\boldsymbol{u}$ to request a function key $\mathrm{sk}_{\boldsymbol{u}}^{\mathrm{SIFE}}$ from the TPA. With the function key $\mathrm{sk}_{\boldsymbol{u}}^{\mathrm{SIFE}}$, the coordinator is able to decrypt the ciphertext and acquire the batch gradient $E_{\mathcal{B}}(\boldsymbol{w})$.

Detailed execution of the FedV-SecGrad process: We now present in detail our *federated vertical secure gradient descent (FedV-SecGrad)* approach. As shown in Algorithm 3 the general *FedV* which adopts a mini-batch based SGD algorithm to train a ML model in VFL fashion. During the *setup* phase, the TPA securely delivers the corresponding public keys to each *participant* pk_i and a common secret random seed r for batch generation. Before the training phase starts, the coordinator sends each participant p_i with the permutation π_{p_i} generated by the private entity resolution approach described earlier. A permutation vector allows the participant to shuffle its local dataset. At each training epoch, the *FedV-SecGrad* approach specified in Procedure 4 is invoked in Line 6 of Algorithm 3. The coordinator queries the participants with a current batch index s_i and the current model weights $\boldsymbol{w}_{p_i}$ with respect to the receiving participant p_i. The batch index allows the participants to subsample the local dataset using the pre-agreed batch generation function ρ when encrypting the 'partial model'. To reduce the data transfer and protect against inference attacks[4], the coordinator only sends each participant the weights that pertain to its features. We denote these partial model weights as $\boldsymbol{w}_{p_i}$.

For each mini-batch, each participant follows the feature-dimension and sample-dimension encryption process shown in lines 13, 14, and 15 of Procedure 4, respectively. As a result, each participant local 'partial model' is encrypted and the two ciphertexts, $\boldsymbol{c}_{\mathrm{fd}}, \boldsymbol{c}_{\mathrm{sd}}$, are sent back to the coordinator. The coordinator waits for a pre-defined amount of time for participant replies, denoted as two sets of corresponding ciphertexts $\mathcal{S}_{\boldsymbol{c}_{\mathrm{fd}}}, \mathcal{S}_{\boldsymbol{c}_{\mathrm{sd}}}$. Once this time has elapsed, it continues the training process by performing the following secure aggregation steps. First, the feature dimension SA is performed. For this purpose, in line 4 vector $\boldsymbol{v}_{\mathcal{P}}$ is initialized. This vector provides the weights for the inputs of each of the received replies and its norm corresponds to the number of received replies. $\boldsymbol{v}_{\mathcal{P}}$ is sent to the TPA which returns the private keep to perform the decryption; before this key is returned, the TPA verifies the suitability of the vector, more details described later. The feature dimension SA is completed in line 6, where the MIFE based decryption takes place resulting in $\boldsymbol{u} = (u_1, ..., u_b)$ which contains the aggregated weighted feature values of b batch

[4]In this type of attack, a participant may try to find out if its features are more important than those of other participants. This can be easily inferred in linear models.

Procedure 4: FedV-SecGrad.

1 ▷The n participants group $\mathcal{P}$ are indexed as p_i; each participant p_i with rearranged dataset $\mathcal{D}_{p_i}$, model weights $\boldsymbol{w}_{p_i}$, current batch index s_i and pre-agreed batch generate function ρ, has assigned public key pk_i and random secret seed r from TPA; current model weights $\boldsymbol{w} := (\boldsymbol{w}_{p_1}, \boldsymbol{w}_{p_2}, ..., \boldsymbol{w}_{p_m})$;

2 **coordinator:**

3 **foreach** $p_i \in \mathcal{P}$ **do** $\mathcal{S}_{\boldsymbol{c}_{\text{fd}}}, \mathcal{S}_{\boldsymbol{c}_{\text{sd}}} \leftarrow$ *query-party*$(\boldsymbol{w}_{p_i}, s_i)$;

4 specify $\boldsymbol{v}_{\mathcal{P}}$ according to $\mathcal{S}_{\boldsymbol{c}_{\text{fd}}}$;

5 $sk_{\boldsymbol{v}_{\mathcal{P}}} \leftarrow$ *query-key-service*$(\boldsymbol{v}_{\mathcal{P}}, \mathcal{E}_{\text{MIFE}})$;

6 **foreach** $\boldsymbol{c}_{fd} \in \mathcal{S}_{\boldsymbol{c}_{fd}}$ **do** $\boldsymbol{u} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Dec}_{sk_{\boldsymbol{v}_{\mathcal{P}}}}(\boldsymbol{c}_{\text{fd}})$ ▷feature dimension SA ;

7 $sk_{\boldsymbol{u}} \leftarrow$ *query-key-service*$(\boldsymbol{u}, \mathcal{E}_{\text{SIFE}})$;

8 **foreach** $\boldsymbol{c}_{sd} \in \mathcal{S}_{\boldsymbol{c}_{sd}}$ **do** $\nabla E_{\mathcal{B}}^{'}(\boldsymbol{w}) \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{SCIP}}}.\text{Dec}_{sk_{\boldsymbol{u}}}(\boldsymbol{c}_{\text{sd}})$ ▷sample dimension SA ;

9 $\nabla E_{\mathcal{B}}(\boldsymbol{w}) \leftarrow \nabla E_{\mathcal{B}}^{'}(\boldsymbol{w}) + \alpha \nabla R(\boldsymbol{w})$

10 **participant:**

11 **function** *query-party*$(\boldsymbol{w}_{p_i}, s_i)$

12 $\mathcal{B}_{p_i} \leftarrow \rho(r, s_i, \mathcal{D}_{p_i})$ ▷generate the batch in current training epoch;

13 **if** *active* p_i **then** $\boldsymbol{c}_{\text{fd}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Enc}_{pk_{p_i}}(\boldsymbol{w}_{p_i}\mathcal{B}_{p_i} - \boldsymbol{y})$;

14 **else** $\boldsymbol{c}_{\text{fd}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.\text{Enc}_{pk_{p_i}}(\boldsymbol{w}_{p_i}\mathcal{B}_{p_i})$;

15 $\boldsymbol{c}_{\text{sd}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{SCIP}}}.\text{Enc}_{pk}(\mathcal{B}_{p_i})$ in sample dimension;

16 **return** $(\boldsymbol{c}_{\text{fd}}, \boldsymbol{c}_{\text{sd}})$ to the coordinator;

17 **TPA:**

18 **function** *query-key-service*$(\boldsymbol{v}, \mathcal{E})$

19 **return** $sk_{\boldsymbol{v}} \leftarrow \mathcal{E}_{\text{FE}}.SKGen(\boldsymbol{v})$;

samples. After that, the sample dimension SA takes place. The coordinator uses $\boldsymbol{u}$ as an aggregation vector that is sent to the TPA to obtain a functional key $\text{sk}_{f,\boldsymbol{u}}$. The TPA verifies the validity of vector $\boldsymbol{u}$ and returns the key if appropriate. Finally, the aggregated model update $\nabla E_{\mathcal{B}}(\boldsymbol{w})$ is found in line 8 by performing a SIFE decryption using $\text{sk}_{f,\boldsymbol{u}}$.

Extending FedV to other machine learning models: We now briefly analyze how to apply *FedV-SecGrad* approach on three classic machine learning models, and defer detailed analysis in Appendix C.

Logistic Regression. The minor modification of directly applying *FedV-SecGrad* in logistic model is either sharing the labels with the coordinator or making the active participant play the role of the coordinator. Suppose that the prediction function $f(\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{w}^\intercal \boldsymbol{x}}}$ can be written as $g(\boldsymbol{w}^\intercal \boldsymbol{x})$, where $g(\cdot)$ is the sigmoid function, i.e., $g(z) = \frac{1}{1+e^{-z}}$. The gradient computation over a mini-batch $\mathcal{B}$ of size n can be described as $\nabla E_{\mathcal{B}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i\in\mathcal{B}}(g(\boldsymbol{w}^{(i)\intercal}\boldsymbol{x}^{(i)})) - y^{(i)})\boldsymbol{x}^{(i)}$. In our proposed *FedV-SecGrad*, the *coordinator* is able to acquire $\boldsymbol{w}^{(i)\intercal}\boldsymbol{x}^{(i)}$ following the *feature dimension SA* process. With the provided labels, it can then compute $u^{(i)} = g(\boldsymbol{w}^{(i)\intercal}\boldsymbol{x}^{(i)}) - y^{(i)}$. Finally, *sample dimension SA* is applied to compute $\nabla E_{\mathcal{B}}(\boldsymbol{w}) = \sum_{i\in\mathcal{B}} u^{(i)}\boldsymbol{x}^{(i)}$. *FedV-SecGrad* also provide an alternative approach for the case of restricting label sharing, where the logistic computation is transferred to linear computation via Taylor approximation as used existing VFL solutions [83].

SVMs with Kernels. The prediction function can be presented as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i=1}^{n} w_i y_i k(\boldsymbol{x}_i, \boldsymbol{x})$, where $k(\cdot)$ is the corresponding kernel function. As kernel functions, such as, linear kernel $\boldsymbol{x}_i^\intercal \boldsymbol{x}_j$, polynomial kernel $(\boldsymbol{x}_i^\intercal \boldsymbol{x}_j)^d$, sigmoid kernel $tanh(\beta \boldsymbol{x}_i^\intercal \boldsymbol{x}_j + \theta)$ (β and θ are kernel coefficients), are based on inner-product computation

Procedure 5: Inference Prevention Module

```
function IPM(v)
    if exploited-vector-filter(v) then forward v to the TPA ;
    else return "exploited vector" message ;
function exploited-vector-filter
    c_nz ← count the non-zero element in v;
    if c_nz < t then true;
    else false;
```

which is supported by our *feature dimension SA* and *sample dimension SA* protocols, these kernel matrices can be computed before the training process. Therefore *FedV-SecGrad* can securely compute the kernel matrix first and then use it in the training process. The gradient computation process for all SVMs equipped with the aforementioned kernel functions will be reduced to a gradient computation of a standard linear SVM, which can clearly be supported by *FedV-SecGrad*.

Inference Prevention Module: The original inner-product based functional encryption schemes proposed in [4, 5] do not consider the case of a *curious* decryption party who manipulates a specially constructed vector to request the functional private key from the TPA, and hence it is possible to reveal one element in the encrypted vector. Our *FedV-SecGrad* approach in *FedV* is much more complex than the simple adoption of functional encryption cryptosystems. For instance, an *honest-but-curious* coordinator can send a manipulated vector such as $\boldsymbol{v}_{\text{exploited}} = (0, ..., 1, 0)$ to request function key to infer the second last element in the input vector $\boldsymbol{x}$ because the inner-product $\langle \boldsymbol{x}, \boldsymbol{v}_{\text{exploited}} \rangle$ is known to the coordinator. Another potential attack to infer the second last element in $\boldsymbol{x}$ is to utilize two manipulated vectors such as $\boldsymbol{w}_{\text{e},1} = (1, 1, 1)$ and $\boldsymbol{w}_{\text{e},2} = (1, 0, 1)$ in two training epochs, respectively. We add a IPM in the TPA to prevent the potential partial private information leakage caused by inference attacks.

As presented in Procedure 5, *IPM* includes one filter, namely *exploited vector filter*, that checks the validity of the querying vector used for generating the function derived key. Throughout the training phase, the *exploited vector filter* will ensure that the non-zero element in the request is less than a threshold t. We present the detailed analysis in Section 3.2.3.

Enabling Dynamic Participation in *FedV*. An important feature in FL systems is the ability to allow participants dynamically to join in and drop out during the training phase. In VFL, a *participant* or the *coordinator* may be absent in several application scenarios. Here, we discuss these variations in our proposed *FedV* framework.

Our *FedV* supports the absence of the *coordinator*. In such a case, the *active participant* is able to play the role of the coordinator. As the *active participant* could be *curious*, but assumed to not collude with other malicious participants in the threat model described above, the privacy requirement of the *active participant* is the same as that of the *coordinator*. A detailed security and privacy analysis will be presented in Section 3.2.3. In addition, the proposed *FedV* framework is also applicable to the dynamic group of participants. In particular, it allows limited number of participants to join in and drop out during the FL training phase.

Existing VFL frameworks rely on the peer-to-peer communications among the participants, which prevents the drop-out behaviors of participants. In *FedV*, communication only occurs between the *coordinator* and the *participants*, which makes the join-in and drop-out behaviors easy to manage, without affecting the existing participants.

Generally, *FedV* can employ the coordinate descent (CD) algorithms (and its variations, block coordinate descent (BCD) algorithms) [185] instead of the SGD to train the ML model. In particular, BCD algorithms allow optimization of the objective function only over one segment (i.e., group of features) at each sub-iteration, while keeping all the other feature segments fixed. Specifically, to train a target ML model, suppose that it requires features $\mathcal{F}^d = (f_1, ..., f_d)$ that are owned by different participants. For each sub-iteration, the *FedV* only needs one participant's data including one feature or partial features in the CD/BCD training. That means the rest of the participants are allowed to drop-out during the training process, and can re-join to continue training when they are available. Unlike the VFL proposal in [83], such CD/BCD are achievable in our *FedV* as our design does not rely on the peer-to-peer communications among the participants.

3.2.3 Security and Privacy Analysis

Security Analysis. The *SA* protocols are the critical components to build the *FedV-SecGrad* approach in the *FedV* framework that supports the basis for the security and privacy guarantees. As illustrated in Appendix A, we employ two types of functional encryption schemes, namely, SIFE and MIFE to construct the *SA* protocols. In our implementation of *SIFE* [4] and *MIFE* [5], we add a public key distribution method run by fully trusted TPA as a supplement of existing functional encryption schemes. Such additional method is only responsible for distributing the public key for each participant. This, however, does not affect the ordinal encryption and decryption constructions as compared to the originally schemes. Thus, for the formal proof of security of adopted FE schemes we refer the readers to [4, 5].

Our *FedV* allows a limited number of colluding participants, but still provides the privacy guarantee for the rest most honest participants. Here, we analyze the possible security concern related to a *brute-force* attack where a limited number of *colluding* participants monitor/inspect the encrypted partial model from other participants. With regards to the public-key setting in *FedV*, each participant has its respective public key pk_{MIFE} and they all have a common public key pk_{SIFE}. Intuitively, such settings could enable the *colluding* participants to infer the target encrypted data by iteratively encrypting its candidate partial model and then checking the ciphertext with target encrypted model as all participants share a common public key pk_{SIFE}. The *SIFE* [4] has proved to be IND-CPA secure that indicates the encrypted models are indistinguishable. For instance, for input data x, with the same public key pk_{SIFE}, the encrypted ciphertexts $c_1 = E_{pk_{\text{SIFE}}}(x), c_2 = E_{pk_{\text{SIFE}}}(x), ..., c_n = E_{pk_{\text{SIFE}}}(x)$ are indistinguishable even for the same input. Thus, there is still a non-negligible advantage for the attackers by increasing the number of colluding participants to brute-force the encrypted data from the honest participants. In summary, our proposed *FedV* is resist to the inference attacks from the colluding participants.

Privacy Analysis. Beside the security guarantee, we analyze extra privacy guarantee in *FedV* under the threat model presented in Section 3.2.2. Briefly, in our threat model, the *TPA* is fully trusted, while the *coordinator* could be *honest-but-curious*.

Here, we analyze the possible inference attacks. As *honest-but-curious* coordinator can launch two types of inference attacks in our designed *FedV* framework: (i) sending exploited weight-related vector such as $\boldsymbol{w}_{\text{exploited}} = (0, ..., 1, 0)$ to request function key to infer the second last element in the input vector $\boldsymbol{x}$ as the inner-product $\langle \boldsymbol{x}, \boldsymbol{w}_{\text{exploited}} \rangle$ is known to the coordinator; (ii) storing the intermediate inner-product results of each iteration to construct degree-1 multivariate polynomial equations, namely, solving $\boldsymbol{W}\boldsymbol{x}_{\text{target}} = \boldsymbol{z}$ to infer target sample $\boldsymbol{x}_{\text{target}}$, where $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_{n_{\text{epoch}}}]$ is constructed from the model weight of each epoch, and $\boldsymbol{z}$ is collected from the inner-product of each epoch.

To prevent the inference attack (i), our proposed *Inference Prevention Module* is able to filter the $\boldsymbol{w}_{\text{exploited}}$. Specifically, the filter module will check the vector $\boldsymbol{w}_{\text{exploited}}$ to ensure that the number of non-zero elements is greater than a threshold τ, and hence the *curious server* cannot distinguish a specific element of $\boldsymbol{x}$ among the weighted-sum of at least τ $\boldsymbol{w}$. The inference attack (ii) is able to occur at both *F-SA* and *S-SA* phase. To solve a system of linear multivariate equations, it requires the *curious* coordinator is able to collect enough number of equations for a target training sample $\boldsymbol{x}_{\text{target}}$ during the secure aggregation phase. However, the coordinator has non-negligible advantage to distinguish the ciphertext of $\boldsymbol{x}_{\text{target}}$ among all encrypted training samples as illustrated as indistinguishability in [4, 5]. Different from existing solutions where the coordinator is responsible for generating the permutations and encrypted mask in the phase of privacy-preserving entity resolution, our proposed solution does not require the encrypted mask. In our *FedV*, each training batch is generated by an agreed batch generation function ρ that relies on the input of a secret random seed generated by a trusted independent entity such as TPA as illustrated in our framework. Hence, the coordinator cannot learn the position of target $\boldsymbol{x}_{\text{target}}$ in *FedV* for different training epochs due to periodical shuffle operations. Furthermore, in the dynamic groups case, one training record only appears in limited number of training epochs. As a result, the *curious* coordinator cannot collect enough number of equations for a specific training record.

3.2.4 Experimental Evaluation

To evaluate the performance of our proposed *FedV* framework, we compare with the following baselines: (i) *Contrasted baseline*: we refer the *contrasted baseline* as the proposed VFL in [83]. Here, the underlying ML need to be trained is the logistic regression model and the secure protocols are built on the additive homomorphic encryption (HE). In addition, like most additive HE based privacy-preserving ML solutions, the SGD and loss computation in [83] also relies on the Taylor series expansion to approximately compute logistic function. (ii) *Centralized baselines*: we refer the *centralized baselines* as the training of different ML models in centralized manner. The contrasted ML algorithms include: the logistic regression model, the logistic regression model with Taylor approximation, the basic linear regression model in ordinary least

Table 3.6: The number of required crypto-related communication for each iteration in the VFL.

Types	Communication Entities	[83]	*FedV*
Secure Stochastic Gradient Descent	coordinator $\leftrightarrow$ participants	$2n$	n
	participants $\leftrightarrow$ participants	$2(n-1)$	0
	TOTAL	$2(2n-1)$	n
Secure Loss Computation	coordinator $\leftrightarrow$ participants	$2n$	n
	participants $\leftrightarrow$ participants	$n(n-1)/2$	0
	TOTAL	$(n^2+3n)/2$	n

squares (OLS), the Support Vector Machine (SVM). Note that as the supported lasso regression and ridge regression models are based on the OLS-based linear regression by adding the L1 and L2 regularization in the cost function respectively. Such regularization module has no affect on the collaboration of participants in the VFL, and hence, we do not report such results in the centralized baselines.

Before the experimental evaluation, we first theoretically evaluate the proposed *FedV* with *contrasted baseline* that use additive HE as the underlying cryptosystem to construct the secure aggregation approach to support secure SGD over the vertically partitioned dataset. A graphical overview and comparison of our proposed *FedV-SecGrad* approach and existing *additive HE* approach is presented in Figure 3.5. Here, we theoretically compare the number of *crypto-related communication* in our proposed *FedV* framework to the contrasted approaches. Suppose that there are n participants and one coordinator in the VFL framework. Note that here we did not consider the dynamic groups case in the *FedV* as the existing *VFL* does not support such dynamic drop-out and join-in cases. As shown in Table 3.6, in total, *FedV* has reduced the number of interactions in SGD phase from $4n-2$ to n, while reducing the number of interactions in loss computation by an order of magnitude, from $(n^2-3n)/2$ to n. The number of interactions during the secure aggregation at the SGD phase of loss computation phase is linear to the number of participants in our *FedV*.

Experimental Setup. To evaluate the performance of the *FedV*, we train several popular ML models such as linear regression, logistic regression, Taylor approximation based logistic regression, and SVM to classify several publicly available datasets from *UCI Machine Learning Repository*. As depicted in Table 3.7, the evaluated datasets include website *phishing*, *ionosphere*, landsat satellite (*statlog*), and optical recognition of handwritten digits (*optdigits*). The number of attributes of those dataset cover from 10 to 64, while the total number of sample instances is from 351 to 6435. The division of training set and test set is also presented in Table 3.7. Please note that if the original problem of the studied dataset is multi-class then we convert it into binary classification problem.

Implementation. We implement the contrasted VFL, our proposed *FedV* and several centralized baseline ML models in Python. To achieve the integer group computation that is required by both the additive homomorphic encryption and the functional encryption, we employ the *gmpy2* library that is a C-coded Python extension module that supports multiple-precision arithmetic, where the underlying performance-

Table 3.7: Dataset description in the experimental evaluation.

Dataset	Attributes	Total Samples	Training	Test
Phishing	10	1353	1120	233
Ionosphere	34	351	288	63
Statlog	36	6435	4432	2003
OptDigits	64	5620	3808	1812

intensive arithmetic operations are implemented in native C modules such as the GMP library. We implement the Paillier cryptosystem as the construction of additive HE scheme that is same to the scheme used in [83], and the constructions of MIFE and SIFE are from [4, 5], respectively.

As those constructions do not provide the solution to address the discrete logarithm problem in the decryption phases, which is a performance intensive computation, we use the same approach as mentioned in Section 3.1, where a hybrid approach is employed. Specifically, to compute the f in $h = g^f$, we setup a hash table $T_{h,g,b}$ to store (h, f) with a specified g and a bound b, where $-b \leq f \leq b$, when the system initializes. When computing discrete logarithms, the algorithm first looks up $T_{h,g,b}$ to find f, where the complexity is $\mathcal{O}(1)$. If there is no result in $T_{h,g,b}$, the algorithm employs the traditional *baby-step giant-step* algorithm [161] to compute f, where the complexity is $\mathcal{O}(n^{\frac{1}{2}})$.

Experimental Environment. All the experiments are performed on a 2.3 GHz 8-Core Intel Core i9 platform with 32 GB of RAM. Note that both the contrasted VFL and our *FedV* frameworks are distributed by multiple processes, where each process represents a participant. The participants and the coordinator communicate over local socket (not run on the real distributed environment for real network), hence the network latency is not measured in our experiment.

Experimental Results. As the contrasted baseline [83] only supports two participants to train a logistic regression model, we first present the comparison results in the setting of two participants, and then we explore the performance of *FedV* using different machine learning models. In last, we study the impact of varying number of participants in *FedV*.

Comparison to Contrasted Approach. We first compare the performance of *FedV* to existing contrasted baseline [83] and centralized baselines, i.e., the non-FL logistic regression model and logistic regression with Taylor approximation over four selected datasets.

As the underlying *logistic regression* is a non-linear model, we implement two types of *FedV*: *normal FedV* and *approximation FedV* according to our proposed approaches, where *approximation FedV* is the implementation applying our *FedV-SecGrad* to address the approximated logistical regression that is same to the contrasted baseline but use different secure aggregation approaches. The *normal FedV* is the implementation applying our *FedV-SecGrad* to address the secure aggregation directly. Accordingly, we also implement a non-FL centralized logistic regression in approximation setting.

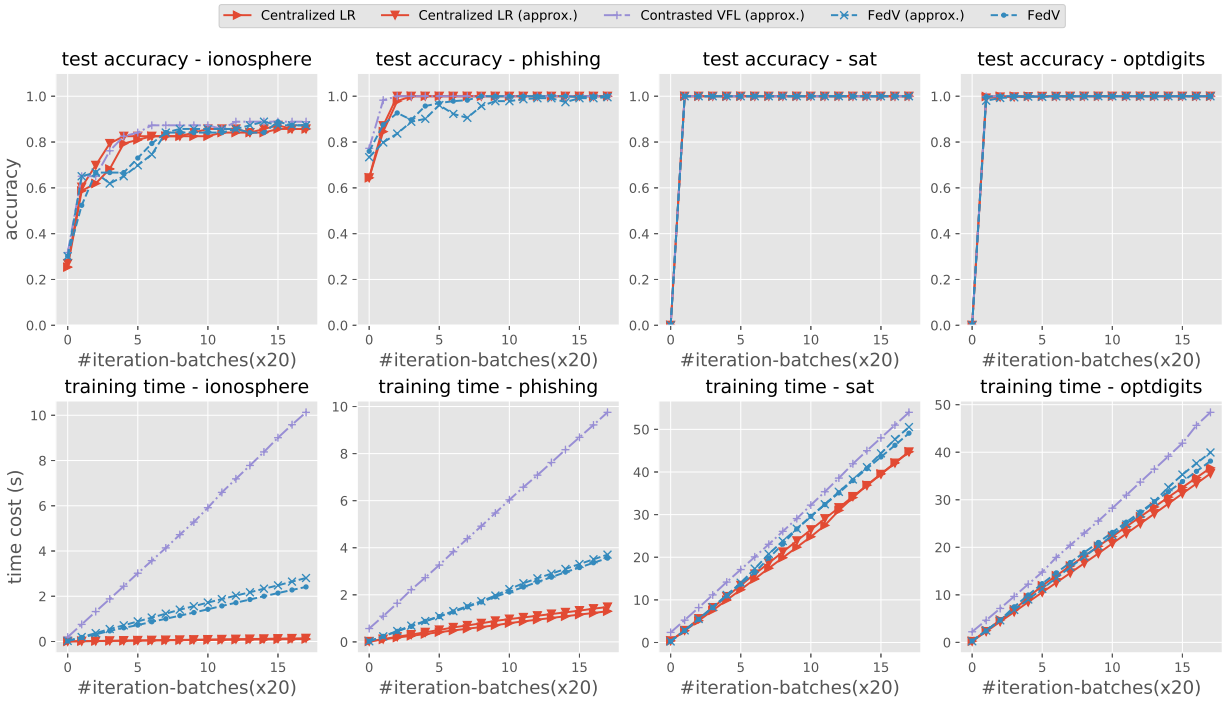


Figure 3.8: Comparison of model accuracy and training time in LR model in two-participant setting.

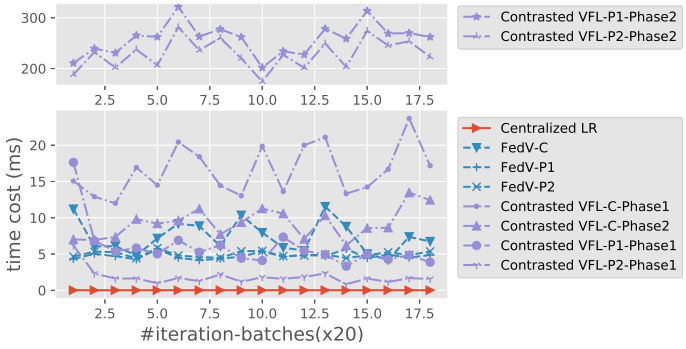


Figure 3.9: Decomposition of training time in *FedV* and contrasted VFL baseline.

In Figure 3.8 we present the test accuracy and training time of each contrasted approach for different datasets. Results show that both of our *approximation FedV* and *normal FedV* can achieve the comparable test accuracy to the *contrasted baseline* [83] and the *centralized non-FL baselines* for all four datasets. Regarding training time, our implemented *approximation FedV* and *normal FedV* efficiently reduce the training time by 10% to 70% for the studied datasets when training for 360 iterations. For instance, as

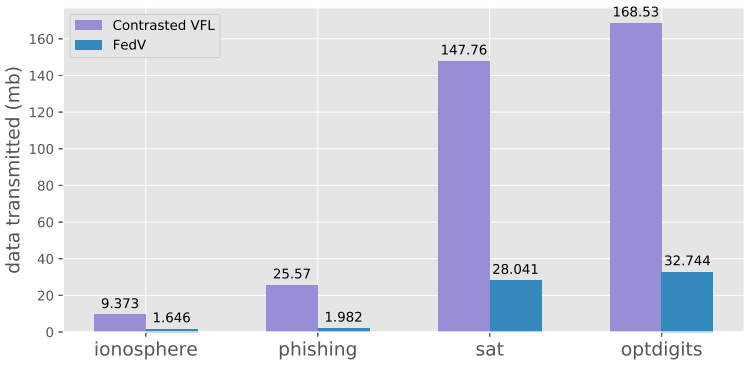


Figure 3.10: Comparison of the size of total data transmitted of training LR model over various datasets.

depicted in Figure 3.8, our *FedV* can reduce around 70% training time for *ionosphere* dataset while just reducing around 10% training time for *sat* dataset. The variation in training time reduction among different datasets is caused by the dataset distribution and model convergence speed.

Furthermore, we decompose the training time of LR model over *ionosphere* dataset in *FedV* and the contrasted VFL baseline to show the exact reason of the training time reduction. As shown in Figure 3.9, compared to the contrasted VFL baseline, *FedV* does not rely on two phase communications for secure gradient computation. Note that in the legend, "C" represents the coordinator, while "P1" and "P2" denotes the active participant and the passive participant, respectively. Besides, the process time of phase 1 of coordinator and phase 2 of each participant is significantly higher than our *FedV*.

We also compare and decompose the size of total data transmitted of LR model over various datasets in *FedV* and the contrasted VFL baseline. As shown in Figure 3.10, compared to the contrasted VFL baseline, *FedV* can reduce the size of total data transmitted by 80% to 90%, because *FedV* only relies on designed non-interactive secure aggregation protocols and does not need the frequent communications used in contrasted VFL baseline.

Performance of FedV over Different ML Models and Impact of Number of Participants. We explore the performance of *FedV* using different popular machine learning models, i.e., the OLS-based linear regression, support vector machine, logistic regression, and also compare with the contrasted centralized baselines - non-FL machine learning models. The first row of Figure 3.11 reports the test accuracy while second row of Figure 3.11 shows the training time for 360 training iterations. In general, our proposed *FedV* can achieve the comparable test accuracy for all machine learning models for all the studied datasets, even though the rates of convergence are different for different datasets. Our *FedV-SecGrad* is based on cryptosystems that computes over integers instead of floating-point number, so *FedV* will lose portion of fractional part of a floating-point number. As shown in Figure 3.11, our *FedV* is able to achieve comparable training time to

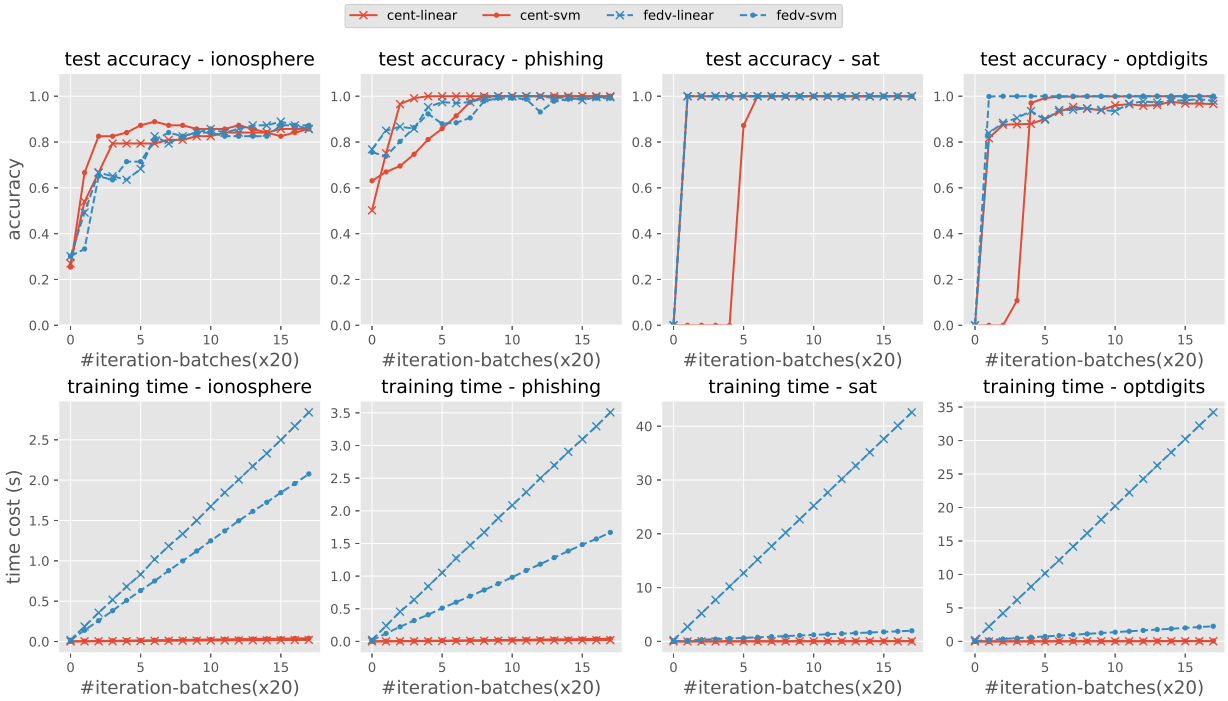


Figure 3.11: Performance of *FedV* over various ML models and the comparison.

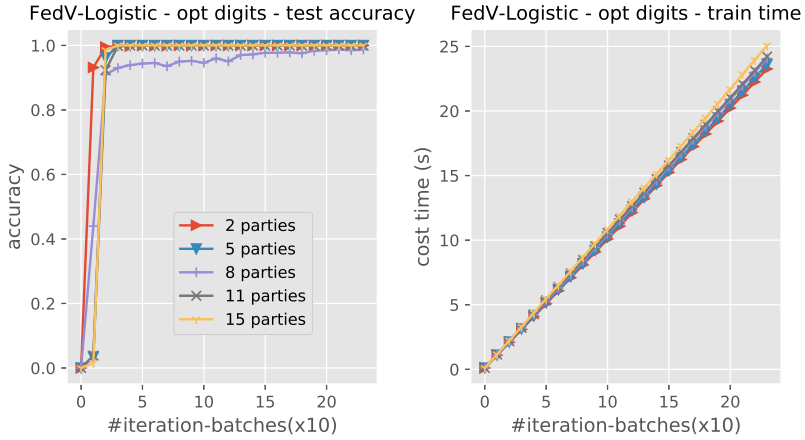


Figure 3.12: Impact of number of participants on the performance of the *FedV* framework.

the corresponding non-FL machine learning models. We also explore the impact of number of participants in *FedV*. Figure 3.12 reports the experimental results of *FedV* under multiple participants setting (i.e., from 2 participants to 15 participants) over the *OptDigits* dataset. Note that the contrasted baseline [83] does

not support more than two participants, and hence we cannot report its performance here. Each of the test setting has a random initialization model weight. As reported in Figure 3.12, the number of participants does not impact the model accuracy, finally, all test cases reach the 100% accuracy. Note that the result is based on the evaluation on *OptDigits* dataset with *FedV Logistic Regression* model. Besides, the training time is linear to the number of participants, and is still very efficient. As reported in Figure 3.8, the training time of *FedV* in logistic reugression model is very close to the normal non-FL logistic regression. For instance, for 100 iterations, the training time of *FedV* with 14 participants is around 10 seconds, while the normal non-FL logistic regression training is about 9.5 seconds.

3.3 Summary and Discussion

FL promises to address privacy concerns and regulatory compliance. However, extant approaches addressing privacy concerns in FL provide strong privacy guarantees and good model performance at the cost of higher training time and high network transmission. We propose *HybridAlpha*, a novel federated learning framework to address these issues. Our theoretical and experimental analysis shows that, compared to existing techniques, on average, *HybridAlpha* can reduce the training time by 68% and data transfer volume by 92% without sacrificing privacy guarantees or model performance. Using *HybridAlpha*, FL can be applied to use cases sensitive to training time and communication overhead, in particular for models with a large number of parameters. Most of existing PPFL frameworks only focus on the case of horizontally partitioned data, while few vertical FL solutions just work on a specific machine learning model and suffer from inefficient secure computation and training time. To address the challenges, we proposed *FedV* an efficient and vertical PPFL framework built based on a two-phase secure aggregation approach that makes use of functional encryption schemes. *FedV* can be used to train a variety of ML models without any approximation including linear models, logistic regression, SVMs among others. Importantly, *FedV* removes the need of peer-to-peer communications among participants reducing substantially the training time and making it applicable for applications where participants cannot connect with each other. Our experiments show reduction of 10%-70% of training time and 80%-90% transmitted data size with respect to the state-of-the art approaches. *FedV* is first VFL framework to support dynamic participants that may be unable to sustain connection throughout the training process, a challenge characteristic of FL. It allows a subset of participants to drop off and rejoin the training process without requiring expensive re-keying operations or affecting the training process.

4.0 Privacy-Preserving Deep Neural Networks using Secure Computing

Deep neural networks (DNN), also known as deep learning (DL), have been increasingly used in many fields such as computer vision, natural language processing, speech/audio recognition, etc. With increasing availability of huge amounts of training data and devices with increased computing power, they are becoming more practical for use in ML based applications. Such DNNs usually consist of two phases: the *training* phase and the *inference* phase. In the *training* phase, a well-designed DNN is provided as input a training dataset and an appropriate optimization algorithm to generate optimal parameters; then, in the *inference* phase, the generated model (i.e., optimal parameters) is used for inference tasks, namely, predicting a label for a specific input sample.

In DNN-enabled applications, one of the critically needed components is a powerful computing infrastructure with powerful CPU and GPU, larger memory storage, etc. Existing commercial AI service providers such as Google, Microsoft, and IBM have devoted significant efforts toward building *infrastructure as a service* (IaaS) platforms for clients that do not have such powerful computing devices. The clients can employ these AI related IaaS to manage and train their models and provide prediction services to their customers.

The size of the available training dataset is another critical issue. In particular, the performance of a DNN system heavily relies on the availability of a large volume of training data. However, in many scenarios, training data used by a DNN may contain privacy-sensitive information. Recent data breach incidents have increased the privacy concerns related to large-scale collection and use of personal data [150, 176]. Moreover, recent regulations such as European General Data Protection Regulation (GDPR), California Consumer Privacy Act, Cybersecurity Law of China, etc., restrict the availability and use of privacy sensitive data. Such privacy concerns of users and requirements imposed by regulations pose a significant challenge for the deployment of DNN solutions.

To balance the privacy concerns and regulatory issues mentioned above, with that of the need of large volumes of training data, several approaches have been proposed in the literature for building privacy-preserving ML (PPML) solutions. These approaches include: (i) applying privacy-preserving mechanisms such as *differential privacy* to limit the disclosure of private information before outsourcing a dataset to a third party to train a DNN model [1]; (ii) employing new DNN architectures such as *federated learning* (FL) where each participant trains a model locally and exchanges only the model parameters with the coordinating server [166]; and (iii) utilizing existing general *secure multi-party computation* (SMC) techniques or other encryption schemes (e.g., homomorphic encryption) to build appropriate security protocols to protect the input training data while training a DNN model [128, 151, 73, 129, 191].

In Table 4.1, we summarize existing representative privacy-preserving approaches used by DNN models. Existing solutions such as a FL approach and those based on *differential privacy* cannot provide strong privacy guarantees because of *inference attacks*, as demonstrated in the literature [66, 167, 131]. The general

Table 4.1: Comparison of representative privacy-preserving approaches in deep learning

Proposal	Training	Prediction	Privacy ▷	Multi-Source †	Technique
[166]	✓	✗	○	-	Federated Setting *
[1]	✓	✗	○	-	Differential Privacy
[128]	-	✓	◑	✗	Customized SMC (GC)‡
[151]	-	✓	◑	✗	General SMC (GC)
[73, 86, 31, 92]	✗	✓	●	✗	Homomorphic Encryption
[129]	✓	✓	●	✗	Homomorphic Encryption
CryptoNN (our work) [191]	✓	✓	●	-	Functional Encryption
NN-EMD (our work)	✓	✓	●	✓	Hybrid Functional Encryption

▷ The column denotes privacy guarantee degree such as mild approach ○ (e.g. differential privacy) and strong guarantee ● (e.g. confidentiality level privacy guarantee).
† The minus symbol indicates the approach supports multiple data sources to some extent.
‡ The data owner trains the model by itself and outsources partial computation in a privacy-preserving setting.
* The model is trained in a federated manner where each data owner trains a partial model on their private data.

SMC(garbled circuits based) approaches, such as those proposed in [128, 151], have a limitation with regards to large volumes of encrypted data that need to be transferred during the execution of the associated secure protocols. Except for the recently proposed solutions in [191, 129], most of these SMC approaches only address privacy issues in the *inference* phase rather than in the *training* phase; this is mainly due to the efficiency challenges related to both computation and communication.

Furthermore, none of the existing solutions for privacy-preserving training of a DNN consider the fact that training data may be coming from multiple data sources partitioned horizontally or vertically. The training dataset may have different composition cases; it may include data from multiple data sources, where:

(i) Each data source provides a dataset that includes all the features;

(ii) Each source provides a dataset that has only a subset of the required features; but, collectively these datasets cover the complete set of features;

(iii) It is a hybrid of (i) and (ii).

Even though existing privacy-preserving solutions such as *CryptoNN* [191] support case (i), cases (ii) and (iii) still pose a huge challenge.

In this chapter, we propose a framework to train a *Neural Network over Encrypted Multi-sourced Datasets* (NN-EMD). That is, *NN-EMD* trains a DNN using a dataset that is composed of independently encrypted datasets from many different sources. Each data source may provide its encrypted data that may include a *complete set of features* or only a *subset of features*.

The goal here is to provide a strong privacy guarantee, while training a DNN model in a more efficient way as compared to the most recently proposed solutions, namely, those in [191, 129]. To the best of our knowledge, this is the first efficient and more practical approach for training a DNN over a set of encrypted/private datasets.

4.1 NN-EMD: Training Neural Networks using Encrypted Multi-Sourced Datasets

4.1.1 Background and Motivation

As mentioned earlier, training DNNs requires a large volume data that may have privacy sensitive data and significant computational resources. An entity that aims to employ a DNN-based solution may not have both. For instance, DNN for breast cancer screening can provide much more effective, efficient, and patient-centric breast cancer screening support than ever before [44]. However, some small clinics may not be able to train a breast cancer screening AI model based on their collected patients' healthcare records because of lack of adequate computing power and ML expertise. So, as an alternative, they can use a commercial cloud service that can provide the required computational infrastructure or DNN models; but such outsourced computation is not desirable without employing appropriate privacy-preserving techniques that can guarantee users' or regulatory privacy protection requirements.

In this section, we focus on a privacy-preserving DNN (PPDNN) approach that uses a client-server architecture with two parties: (i) the *cloud service provider* (server) with powerful computational infrastructure that can be employed for training a DNN model; and (ii) a *client pool* (data sources) that have privacy-sensitive datasets and need to build a DNN model based on these training datasets without leaking private information.

Such a PPDNN approach needs novel secure computation protocols to support efficient computation and interactions between a pool of clients and a server, while offering strong privacy guarantees. Existing general SMC solutions (i.e., garbled circuits) have limitations because they need to perform several rounds of communication involving transmission of large volumes of intermediate data. Using these techniques for DNN is cost prohibitive because of huge volumes of training data needed. Cryptography-based solutions (e.g., homomorphic encryption-based SMC) also have computational efficiency problems.

To the best of our knowledge, the approach proposed by Bost et al. in [27] is the initial work that supports both training and predictive analysis over encrypted data. Their approach achieves this by integrating several crypto schemes (i.e., Quadratic Residuosity cryptosystem, Paillier cryptosystem, and homomorphic encryption) with secure protocols designed for them. However, it only supports limited types of basic ML models such as naïve bayes, decision trees and support vector machines, but not DNN. Most recently, approaches proposed by Nandakumar et al. in [129], and Xu et al. in [191] are the only ones that support training DNN over encrypted data; their approaches use homomorphic and functional encryption, respectively. Insight from these two approaches indicate that the crypto-based secure computing techniques are promising for the training phase of a DNN model. However, there are two key challenges toward achieving effective and efficient training of DNNs over encrypted datasets that we address in this paper:

- *Efficiency of Training Process.* The existing secure computing protocols are not efficient, as mentioned above. For instance, with optimized approaches (e.g., multiple threads, training data distillation) in [129],

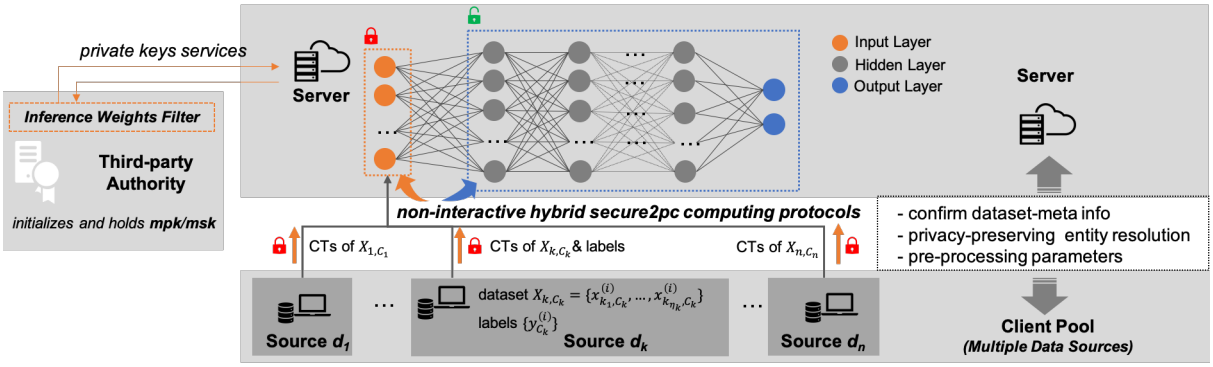


Figure 4.1: Overview of the proposed *NN-EMD* framework.

training time for one mini-batch, with 60 samples, is around 40 minutes. This indicates that training time in the case of a larger volume of training data will be significantly higher.

- *Multiple Data Sources.* There is a lack of consideration of real complex training data composed of horizontally and vertically partitioned datasets coming from multiple data sources. Meanwhile, the training techniques also provide strong confidentiality-level privacy guarantees.

4.1.2 Overview of *NN-EMD*

In *NN-EMD*, we have the following three roles/entities: a *client pool*, a *server*, and a *trusted third-party authority (TPA)*:

- A *client pool* of multiple *data sources* that collaboratively contribute to the final training dataset composed of horizontally and vertically partitioned data, or a hybrid mix of the two. Each data source still keeps its data confidential from the rest.
- A *server* responsible for training a DNN over a training dataset composed of multiple private datasets.
- A trusted *TPA* that initializes the underlying cryptosystems by setting secret credentials. Then, it distributes the associated public keys to data sources in the *client pool* and the *server*, and provides private key service to the *server* during the training phase. *Note that it is not necessary for a TPA to acquire/access any (encrypted) training data.*

Figure 4.1 illustrates the *NN-EMD* framework. Before training a model , the *server* collects the meta information about the training datasets from the *client pool* and then *client pool* coordinates the privacy-preserving entity resolution task with each data source if the final training dataset is a vertical composition of datasets from sources in the client pool. We assume that for each data sample, there exists at least one data source having the label and only one data source's labels are enrolled in the training phase. Meanwhile, all the clients in the *client pool* and the *server* acquire associated cryptographic keys from the *TPA*. Then,

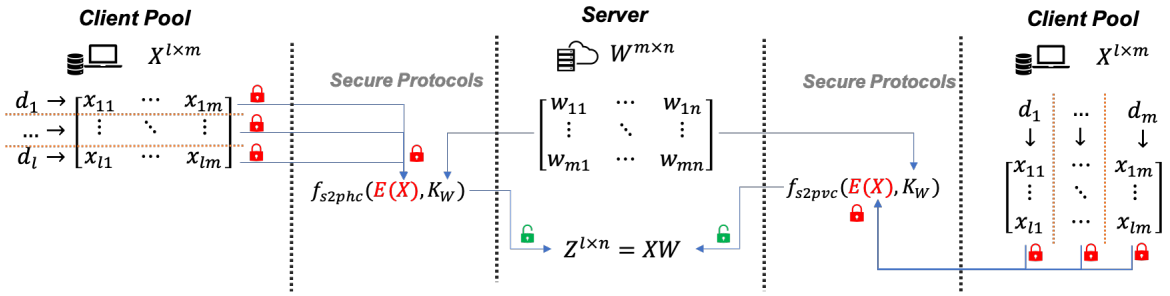


Figure 4.2: Illustration of secure two-party computation approaches between the client pool and the server.

each data source in the *client pool* pre-processes its data as required by the framework and outsources the encrypted data to the *server*. The *server* starts to train the model by setting up the proper training hyperparameters, e.g., learning rate, number of iterations, and the total number of data sources, etc.

4.1.3 Threat Model and Assumptions

We assume that there exists a *trusted* TPA. This *TPA* is an independent third-party that is trusted by all the *data sources* in a *client pool* and the *server*. Note that it is also a common assumption in cryptosystems such as those proposed in [25, 23, 77]. The role of a trusted *TPA* is similar to the role of a trusted *certificate authority* in existing public key infrastructures. In this paper, we consider the following threat model:

(i) *Honest-but-curious Server*; which is a common assumption in most of the existing approaches ([22, 191, 129]). Here, the *server* follows the instructions of a protocol or algorithm, but may try to learn private information by inspecting the collected encrypted dataset and decrypted functional results during the training phase.

(ii) *Curious and Colluding Data Sources*: In a *client pool*, some of curious data sources may try to collude to infer private information of other non-colluding data sources by inspecting their outsourced encrypted data.

4.1.4 Secure Computation Approaches

Here, we present our proposed privacy-preserving secure computation approach between a *server* and a *client pool* (data sources). To be specific, we propose two secure computation protocols, namely, *secure two-party horizontally partitioned* computation protocol (*S2PHC*, see Figure 4.3) and *secure two-party vertically partitioned* computation protocol (*S2PVC*, see Figure 4.4).

Note that arrows indicate assignment operation, while the equal sign is a comparison operation. Both the protocols are non-interactive secure two-party computation protocols, where there are no interactions between the *server* and the *client pool*; i.e., they have only one-way communication between them.

Secure Two-party Horizontally Partitioned Computation Protocol

Initialization and Key Services

$\Rightarrow$ *TPA initializes the system as follows:*

- initializes the single-input FEIP cryptosystem by generating a common public key and master private key, $pk_{\text{SCIP,com}}, msk_{\text{SCIP}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{SCIP}}.Setup(1^{\lambda}, 1^{\eta})$ by giving parameters λ and η, where λ is the security parameter indicating the bit length of security credentials, while η denotes the maximum length of all possible input vectors of the inner-product function f_{SIIP} during the execution phase of the protocol.
- initializes a private authenticated channel with the *server* and the *client pool*, respectively.
- delivers the public key $pk_{\text{SCIP,com}}$ and the parameter η to both parties, namely, *client pool* and *server*.

$\Rightarrow$ *TPA provides key services as follows:*

- receives a functional private key request, and $\boldsymbol{w}$ from the *server*.
- checks the $\boldsymbol{w}$ to prevent potential inference attack by making sure $|\boldsymbol{w}| \leq \eta$ and non-zero elements of $\boldsymbol{w}$ is less than the threshold τ using the weights filter module .
- executes private key generation algorithm to generate private key $sk_{\text{SCIP},\boldsymbol{w}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{SCIP}}.KeyGen(msk, \boldsymbol{w})$, and sends back the key via the private authenticated channel.

Party: Client Pool $\Rightarrow$ *all data sources in the client pool agree on an encoding precision ϵ_{client}. For each data source $d_k \in \{d_1, ..., d_l\}$ in the client pool, each client in the pool executes the following steps:*

- receives the public key pk_{com} and η from the *TPA* and verifies the validity of pk_{com}.
- encodes elements in data from floating-point format $\boldsymbol{X}_{\text{fp}}$ into integer format $\boldsymbol{X}_{\text{int}}$ with encoding precision ϵ_{client}.
- counts the shape of the length of $\boldsymbol{X}_{\text{int}} \rightarrow (s_{d_k}.r, s_{d_k}.c)$, and checks $s_{d_k}.c \leq \eta$.
- for each row $\boldsymbol{x}_i$ of $\boldsymbol{X}_{\text{int}}$, calls SCIP encryption algorithm $ct_{\boldsymbol{x}_i,d_k} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{SCIP}}.Enc(pk_{\text{com}}, \boldsymbol{x}_i)$.
- if any above operations (assertion, verification, encoding, encryption) fails, abort.
- sends all ciphertexts $\{ct_{\boldsymbol{x}_1,d_k}, ..., ct_{\boldsymbol{x}_l,d_k}\}$ and parameters $\epsilon_{\text{client}}, (s_{d_k}.r, s_{d_k}.c)$ to the *server*.

Party: Server $\Rightarrow$ *the server executes the following steps:*

- receives the public key $pk_{\text{SCIP,com}}$ and η from the *TPA* and verifies the validity of $pk_{\text{SCIP,com}}$.
- collects ciphertexts $\mathbf{ct} \leftarrow \{ct_{\boldsymbol{x}_1,d_k}, ..., ct_{\boldsymbol{x}_l,d_k}\}$ and parameters ϵ_{client}, $\{(s_{d_k}.r, s_{d_k}.c)\}$ from the *client* party.
- sets up the encoding precision ϵ_{server}, and encodes each element in input weights from floating-point format $\boldsymbol{W}_{\text{fp}}$ into integer format $\boldsymbol{W}_{\text{int}}$.
- counts the shape of $\boldsymbol{W}_{\text{int}} \rightarrow (s_{\text{server}}.r, s_{\text{server}}.c)$, and checks $\forall i, j, s_{\text{client}}.c \leftarrow s_{d_i}.c = s_{d_j}.c$ and $s_{\text{server}}.r = s_{\text{client}}.c \wedge s_{\text{server}}.r \leq \eta$.
- for each column $\boldsymbol{w}_i$ of $\boldsymbol{W}_{\text{int}}$, sends a function private key request to the *TPA*, and collects the received private keys $\boldsymbol{sk} \leftarrow \{sk_{f_{\text{SIIP}},\boldsymbol{w}_1}, ..., sk_{f_{\text{SIIP}},\boldsymbol{w}_m}\}$ with verification.
- if all above operations (assertion, verification, encoding, encryption) fails, abort.
- initializes a matrix $\boldsymbol{Z}$ with shape $(|\boldsymbol{ct}|, |\boldsymbol{sk}|)$, and for each $i \in \{1, ..., |\boldsymbol{ct}|\}$ and $j \in \{1, ..., |\boldsymbol{sk}|\}$, calls decryption algorithm $w_{i,j} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{SCIP}}.Dec(pk_{\text{SCIP,com}}, \boldsymbol{ct}[i], \boldsymbol{sk}[j], \boldsymbol{w}_j)$.
- decodes each element in $\boldsymbol{Z}$ from integer format into floating-point format using ϵ_{server} and ϵ_{client}.

Figure 4.3: Description of secure two-party horizontally partitioned computation protocol.

The difference between the two secure computation protocols is mainly with regards to how the input datasets from *client pool* are composed. Suppose that there is a secure computation task such as a matrix multiplication $\boldsymbol{X}^{l\times m}\boldsymbol{W}^{m\times l}$ between the *client pool* and the *server*, where the *client pool* has the matrix $\boldsymbol{X}^{l\times m}$ that is composed of data from different data sources $\{d_k\}$ and the *server* has $\boldsymbol{W}^{m\times l}$. As shown in Figure 4.2, $\boldsymbol{X}^{l\times m}$ represents horizontal or vertical composition of data from multiple sources.

Secure Two-party Horizontally Partitioned Computation Protocol. We present the detailed description of the *S2PHC* protocol in Figure 4.3. The protocol is built from the single-input functional encryption scheme. Here, we suppose that each data source in the *client pool* has the same column length related to $\boldsymbol{X}$ as illustrated in Figure 4.2. Note that the *S2PHC* protocol can be considered as an improvement of the secure matrix computation approach proposed in [191] where the possibility of multiple horizontal data sources had been mentioned, but no theoretical analysis and practical implementation were presented. Unlike in [191], we present specific practical construction in our protocol with the experimental evaluation

Secure Two-party Vertically Partitioned Computation Protocol

Initialization and Key Services

⇒ *TPA initializes the system as follows:*

- initializes the multi-input FEIP crypto schemes by generating a common public key, master public key and private key $pk_{\text{MCIP,com}}, mpk_{\text{MCIP}}, msk_{\text{MCIP}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.Setup(1^{\lambda}, \vec{\boldsymbol{\eta}}, n)$ by giving parameters λ and $(\vec{\boldsymbol{\eta}}, n)$, where $(\vec{\boldsymbol{\eta}}, n)$ indicates the allowed n maximum number of data sources where each data source has maximum input length represented as $\vec{\eta}$, during the computation execution of f_{MCIP}.
- assigns a identity id_{d_k} for each registered data source d_k in the *client pool*.
- initializes a private authenticated channel with the *server* and the *data sources*, respectively.
- delivers d_k-associated $pk_{\text{MCIP},id_{d_k}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.PK(mpk_{\text{MCIP}}, msk_{\text{MCIP}}, id_{d_k})$, $\eta_{id_{d_k}} \leftarrow \vec{\eta}$ and the common public key $pk_{\text{MCIP,com}}$ to each data source id_{d_k}, respectively.
- delivers the common public key $pk_{\text{MCIP,com}}, \vec{\boldsymbol{\eta}}, n$ to the *server*.

⇒ *TPA provides key services:*

- receives the request $\boldsymbol{w}$ from the *server*.
- checks $\boldsymbol{w}$ to prevent potential inference attack by checking that non-zero elements of $\boldsymbol{w}$ is less than the threshold τ using weights filter module.
- generates private key $sk_{\text{MCIP},\boldsymbol{w}} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.SK(mpk_{\text{MCIP}}, msk_{\text{MCIP}}, \boldsymbol{w})$, and sends back the key via the private authenticated channel.

Party: Client Pool ⇒ *all data sources agree on an encoding precision ϵ_{client}. For each data source $d_k \in \{d_1, ..., d_m\}$ in the client pool, each client executes the following steps:*

- receives the public key $pk_{\text{MCIP,com}}$, $pk_{\text{MCIP},id_{u_k}}$ and $\eta_{id_{u_k}}$ from the *TPA* and verifies the validity of $pk_{\text{MCIP,com}}$ and $pk_{\text{MCIP},id_{u_k}}$.
- encodes elements in data from floating-point format $\boldsymbol{X}_{\text{fp}}$ into integer format $\boldsymbol{X}_{\text{int}}$ with encoding precision ϵ_{client}.
- counts the shape of the length of $\boldsymbol{X}_{\text{int}} \rightarrow (s_{d_k}.r, s_{d_k}.c)$, and checks $s_{d_k}.c \leq \eta_{id_{u_k}}$.
- for each row $\boldsymbol{x}_i$ of $\boldsymbol{X}_{\text{int}}$, calls MCIP encryption $ct_{\boldsymbol{x}_i,d_k} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.E(pk_{\text{com}}, \boldsymbol{x}_i)$.
- if any above operations (assertion, verification, encoding, encryption) fails, abort.
- sends all ciphertexts $\{ct_{\boldsymbol{x}_1,d_k}, ..., ct_{\boldsymbol{x}_l,d_k}\}$ and parameters $\epsilon_{\text{client}}, (s_{d_k}.r, s_{d_k}.c)$ to the *server*.

Party: Server ⇒ *the server executes the following steps:*

- receives the public key $pk_{\text{MCIP,com}}, \vec{\boldsymbol{\eta}}, n$ from the *TPA* and verifies the validity of $pk_{\text{MCIP,com}}$.
- collects the ciphertexts $\mathbf{ct} \leftarrow \{\{ct_{\boldsymbol{x}_i,d_1}\}, ..., \{ct_{\boldsymbol{x}_i,d_m}\}\}$ and parameters $\epsilon_{\text{client}}, \{(s_{d_k}.r, s_{d_k}.c)\}$ from the *client pool*.
- sets up the encoding precision ϵ_{server} and encodes each element in input weights from floating-point number $\boldsymbol{W}_{\text{fp}}$ into integer number $\boldsymbol{W}_{\text{int}}$.
- counts the shape of $\boldsymbol{W}_{\text{int}} \rightarrow (s_{\text{server}}.r, s_{\text{server}}.c)$, and checks $\forall i, j, s_{d_i}.r = s_{d_j}.r$ and $s_{\text{server}}.r = \sum s_{d_i}.c \wedge s_{\text{server}}.r \leq \sum \vec{\boldsymbol{\eta}} \wedge |\boldsymbol{ct}| < n$.
- for each column $\boldsymbol{w}_i$ of $\boldsymbol{W}_{\text{int}}$, sends a function private key request to the *TPA*, and collects the received keys $\boldsymbol{sk} \leftarrow \{sk_{f_{\text{MCIP}},\boldsymbol{w}_1}, ..., sk_{f_{\text{MCIP}},\boldsymbol{w}_m}\}$ with verification.
- if all above operations (assertion, verification, encoding, encryption) fails, abort.
- re-organizes $\boldsymbol{ct} \rightarrow \boldsymbol{ct}'$ by aggregating by $\boldsymbol{ct}$ index.
- initializes a matrix $\boldsymbol{Z}$ with $|\boldsymbol{ct}'|$ rows and $|\boldsymbol{sk}|$ columns, and for each $i \in \{1, ..., |\boldsymbol{ct}'|\}$ and $j \in \{1, ..., |\boldsymbol{sk}|\}$, and calls decryption algorithm $w_{i,j} \leftarrow \mathcal{E}_{\text{FE}}^{\mathcal{F}_{\text{MCIP}}}.D(pk_{\text{com}}, \boldsymbol{ct}[i], \boldsymbol{sk}[j], \boldsymbol{w}_j)$.
- decodes each element in $\boldsymbol{Z}$ from integer format into float point format using ϵ_{server} and ϵ_{client}.

Figure 4.4: Description of secure two-party vertically partitioned computation protocol.

in Section 4.3.

Secure Two-party Vertically Partitioned Computation Protocol. Figure 4.4 shows the details of the S2PVC protocol. As for the *S2PHC* protocol, here, we assume that each data source from the *client pool* has the same row length with regards to $\boldsymbol{X}$. The *S2PVC* protocol is constructed using the multi-input functional encryption scheme as the key underlying scheme.

4.1.5 *NN-EMD* Training

Here, we present the details of our proposed *NN-EMD* framework. As mentioned above, *NN-EMD* mainly includes two parties: the *server* and the *client pool*, and they use *S2PHC* and *S2PVC* protocols.

Algorithm 6: *NN-EMD* Training Algorithm

Input: secure parameter 1^λ, functionality parameters $(\eta, \vec{\boldsymbol{\eta}}, n)$, data sources $S_d = \{d_k\}$, each data source d_k has dataset $\boldsymbol{X}_{d_k}$.
Output: trained model $\boldsymbol{W}$

1 initialize *S2PHC* protocol by setting $(1^\lambda, \eta)$ and *S2PVC* protocol by setting $(1^\lambda, \vec{\boldsymbol{\eta}}, n)$;
2 $p_{\text{batch}}, T_{d_k} \leftarrow$exchange meta-information of $\{\boldsymbol{X}_{d_k}\}$;
3 **party client** *pre-process*$(\{\boldsymbol{X}_{d_k}, p_{batch}, p_{T_{\boldsymbol{X}_{d_k}}}\})$
4 **foreach** $d_k \in S_{d_k}$ **do**
5 **if** T_{d_k}=T_f **then**
6 **foreach** *mini batch* $\boldsymbol{X}_{d_k,batch} \in \boldsymbol{X}_{d_k}$ **do**
7 $S_{\boldsymbol{ct}_{\text{ff}}} \leftarrow \text{S2PHC}(d_k, \boldsymbol{X}_{d_k,\text{batch}})$, $S_{\boldsymbol{ct}_{\text{bp}}} \leftarrow \text{S2PHC}(d_k, \boldsymbol{X}^\intercal_{d_k,\text{batch}})$;
8 **else**
9 start entity resolution with shuffle;
10 **foreach** *mini batch* $\boldsymbol{X}_{d_k,batch} \in \boldsymbol{X}_{d_k}$ **do**
11 $S_{\boldsymbol{ct}_{\text{ff}}} \leftarrow \text{S2PVC}(d_k, \boldsymbol{X}_{d_k,\text{batch}})$ $S_{\boldsymbol{ct}_{\text{bp}}} \leftarrow \text{S2PHC}(d_k, \boldsymbol{X}^\intercal_{d_k,\text{batch}})$;
12 sends $S_{\boldsymbol{ct}_{\text{ff}}}, S_{\boldsymbol{ct}_{\text{bp}}}, T_{d_k}$ and Y if d_k has the label;
13 **party server** *training*$(\{S_{\boldsymbol{ct}_{ff}}, S_{\boldsymbol{ct}_{bp}}, T_{d_k}, \boldsymbol{Y}\})$
14 $\boldsymbol{W} \leftarrow$ initialize model weights;
15 **foreach** *iteration* **do**
16 **foreach** *mini batch* $\boldsymbol{ct}_{ff} \in S_{\boldsymbol{ct}_{ff}}, \boldsymbol{ct}_{bp} \in S_{\boldsymbol{ct}_{bp}}$ **do**
17 **if** T_{d_k}=T_p **then** $\boldsymbol{A}_1 \leftarrow \text{S2PVC}(\text{server}, \boldsymbol{ct}_{\text{ff}})$;
18 **else** $\boldsymbol{A}_1 \leftarrow \text{S2PHC}(\text{server}, \boldsymbol{ct}_{\text{ff}}, \boldsymbol{W}_1)$;
19 $\boldsymbol{A} \leftarrow$ feed-forward$(\boldsymbol{A}_1, \boldsymbol{W})$;
20 $\boldsymbol{\nabla}_{2,...,l}, \boldsymbol{\sigma} \leftarrow$ gradient compute$(\boldsymbol{Y}, \boldsymbol{W}, \boldsymbol{A})$;
21 $\boldsymbol{\nabla}_1 \leftarrow \text{S2PHC}(\text{server}, \boldsymbol{ct}_{\text{bp}}, \boldsymbol{\sigma})$;
22 $\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\nabla$;
23 **return** $\boldsymbol{W}$

The *NN-EMD* training includes three types: (i) horizontal partitioning based training (HPT), (ii) vertical partitioning based training (VPT), and (iii) hybrid partitioning based training (HybridPT).

Suppose that there exists data sources S_d=$\{d_1, ..., d_m\}$, where each data source $d_k \in S_d$ has dataset $\boldsymbol{X}_{d_k}$. The goal of the *NN-EMD* framework is to train a DNN model based on the dataset $\boldsymbol{X}$ that is composed of $\{\boldsymbol{X}_{d_1}, ..., \boldsymbol{X}_{d_m}\}$ without leaking $\boldsymbol{X}$ to the *server*, and without disclosing $\boldsymbol{X}_{d_i}$ to d_j where $d_i, d_j \in S_d \wedge d_i \neq d_j$. Such an assumption is common in existing vertical ML related literature, and also indicates there is no overlapping features among those data sources except for the identity feature used for the privacy-preserving entity resolution.

Algorithm 6 illustrates how our proposed *S2PHC* and *S2PVC* protocols are integrated in the training process of a DNN model. First, we initialize *S2PHC* and *S2PVC* protocols with proper security parameter 1^λ and function parameters $(\eta, \vec{\boldsymbol{\eta}}, n)$ as defined in Section 4.1.4. Then, the *server* acquires the basic meta-information of the training dataset from each source from the *client pool*, and decides several training hyperparameters such as proper mini-batch size p_{batch} and dataset type T_{d_k} shared with each data source (lines 1-3). Here, we define dataset types: T_f and T_p to indicate a dataset with full or partial set of features, respectively.

According to different compositions of final training data $\boldsymbol{X}$, we propose three different training approaches: *training over horizontally partitioned data*, *training over vertically partitioned data*, and *training over hybrid partitioned data*.

HPT. This approach deals with the case where each data source's dataset has full set of features needed

in the training. That is, $\boldsymbol{X}$ is horizontally composed of $\{\boldsymbol{X}_{d_1}, ..., \boldsymbol{X}_{d_m}\}$. In this case, each data source first divides its local dataset into several mini-batches according to the received batch parameter. Then, for each mini-batch, the data source executes *S2PHC* protocol twice with input mini-batch $\boldsymbol{X}_{d_k,batch}$ and its transpose $\boldsymbol{X}^{\intercal}_{d_k,batch}$, respectively. The generated ciphertexts $S_{\boldsymbol{ct}_{\text{ff}}}$ and $S_{\boldsymbol{ct}_{\text{bp}}}$ are used in feed-forward computation and back-propagation computation in the training phase, respectively.

On the server side, weights are randomly initialized for the model. For each mini-batch iteration, *S2PHC* protocol is executed with $S_{\boldsymbol{ct}_{\text{ff}}}$ to support secure computation that occurs between the input layer and the first hidden layer (line 25). As the output is in plaintext, the normal feed-forward operations can be continued as in a normal DNN training phase. In the back-propagation phase, the normal gradient computation can be done first from the last layer. When it comes to the first layer, the *server* executes the *S2PHC* protocol with different ciphertext, namely, $S_{\boldsymbol{ct}_{\text{bp}}}$. Finally, the weights are updated using the learning rate and current gradients.

VPT. This approach is for the case where each data source's dataset has a subset of features, however, these partial features collected from all the sources form the complete set of features; i.e., $\boldsymbol{X}$ is vertically composed of $\{\boldsymbol{X}_{d_1}, ..., \boldsymbol{X}_{d_m}\}$. Note that we assume that each $\boldsymbol{X}_{d_k}$ has an identity column so that the privacy-preserving entity resolution mechanism can be executed; there is no overlapping features that will be used in the training. In this case, each data source starts with a privacy-preserving entity resolution mechanism with the *server* that plays the role of a coordinator, similar to those in other approaches such as in [159, 89]. Here, each data source sends the encoded identical features to the *server* for entity matching. Then, the *server* generates a proper permutation for each data source to re-order its local data. As a result, a data source does not know which entity in its dataset has been enrolled in the training; and the *server* still cannot learn the training dataset. As entity resolution is not the core contribution in our framework, we refer the reader to [89] for more details on that.

Here, each data source generates $S_{\boldsymbol{ct}_{\text{ff}}}$ by executing the *S2PVC* with input $\boldsymbol{X}_{d_k,batch}$, while generating $S_{\boldsymbol{ct}_{\text{bp}}}$ by executing *S2PHC* with input $\boldsymbol{X}^{\intercal}_{d_k,batch}$. The *server* acquires the output of the first hidden layer by executing the *S2PVC* protocol with corresponding $S_{\boldsymbol{ct}_{\text{ff}}}$.

HybridPT. Our *NN-EMD* framework can also be applied to the hybrid case where $\boldsymbol{X}$ is composed of the data from multiple data sources using a mix of horizontal and vertical compositions. Algorithm 6 is for processing the hybrid training case by integrating a HPT approach with a VPT approach.

Comparison with Existing Solutions. Here we briefly compare our *NN-EMD* framework with *CryptoNN* [191] and the one in [129]. *CryptoNN* is actually a special instance of our *NN-EMD* framework in the HPT setting. Unlike those in [191] [129], *NN-EMD* does not protect the label information in the training dataset. Actually, the encrypted label information in *CryptoNN* framework can be easily inferred, while the design of encrypting label in [129] is required by the adoption of underlying homomorphic encryption. We argue that *NN-EMD* satisfies the privacy requirements even though the label is exposed to the *server*; we analyze this in Section 4.2. In [129], all the outputs of each layer are still in ciphertext form. The output of the first

hidden layer in *NN-EMD* is in plaintext; because of which the training time does not increase as in [129].

Note that we do not present the *inference* phase of the DNN model since the *inference* can be viewed as one iteration of feed-forward computation in the training phase, as shown in Algorithm 6.

4.2 Security and Privacy Analysis

4.2.1 Security of Underlying Cryptosystems

S2PHC and *S2PVC* protocols are critical components of the *NN-EMD* framework that provides the basis for privacy guarantees. As presented in Appendix A, we add protocols to deliver the public keys and private keys generated by the *TPA* on the originally proposed constructions of single-input and multi-input functional encryption schemes that we adopt for our proposed scheme.

For the formal proof of security of adopted functional encryption schemes we refer the readers to [4, 5]. In our adoption of these schemes, the added public key distribution and private key delivery methods are managed by the *TPA*. This, however, does not affect the ordinal encryption and decryption constructions as compared to the originally proposed schemes.

Colluding Data Sources. With regards to the public-key setting in our framework with multiple data sources, each data source has its respective public key $pk_{\text{MI-FEIP}}$ and they all have a common public key $pk_{\text{SI-FEIP}}$. Here, we analyze the possible security concern related to a *brute-force* attack where a *colluding* data source monitors or inspects the encrypted outsourced datasets from other data sources/clients. Intuitively, such settings could enable the *colluding* data source in the client pool to infer the target encrypted data by iteratively encrypting its candidate data and then checking the ciphertext with target encrypted data as all sources share a common public key $pk_{\text{SI-FEIP}}$. However, the underlying SI-FEIP scheme can prevent such an attack by introducing a random initialization factor in the encryption algorithm. The encryption algorithm will generate different ciphertext even for the same input when invoked by the secure computation protocols. For instance, for input data x, with the same public key $pk_{\text{SI-FEIP}}$, the encrypted ciphertexts $c_1 = E_{pk_{\text{SI-FEIP}}}(x), c_2 = E_{pk_{\text{SI-FEIP}}}(x), ..., c_n = E_{pk_{\text{SI-FEIP}}}(x)$ are indistinguishable. That ciphertext indistinguishability is guaranteed by the IND-CPA security of SI-FEIP [4]. Thus, there is still a non-negligible advantage for the attackers by increasing the number of colluding data sources to brute-force the encrypted data from the non-colluding data source [4]. As a result, our framework can resist such a *brute-force* attack by the *colluding* data sources.

As mentioned earlier, the labels in our framework is not protected. We argue that such a design does not disclose the private information of the training data. Essentially, in the binary classification task, the label is encoded into meaningless value such as using {1,-1} to represent positive and negative labels rather than using a meaningful/concrete label such as "this x-ray image represents cancer". The *server* can only learn group information of the encrypted data such as the information that $E_{\text{FE}}(\boldsymbol{X}_{y=1})$ belongs

to label $y = 1$, but the *server* cannot learn $\boldsymbol{X}_{y=1}$, as it is protected by the cryptosystems, and what $y = 1$ means. The *server* is also not able to launch the enrollment inference attack where the curious server tries to infer whether a target data is enrolled in the training or not, because the training data is encrypted via functional encryption. In particular, the adopted FE schemes have the IND-CPA security guarantee, where the ciphertexts $c_i = E_{pk_{\text{SI-FEIP}}}(x), c_j = E_{pk_{\text{SI-FEIP}}}(x)$ of the same data x is indistinguishable [4]. Let us suppose the target of enrollment inference attack is x_{target}. The *data source* encrypt x_{target} to $c_{\text{target}} = E_{pk_{\text{SI-FEIP}}}(x_{\text{target}})$. Even though the *server* has the original data x_{target}, it is not able to infer whether x_{target} is in the training dataset nor not, because the generated ciphertext of $c_{\text{server}} = E_{pk_{\text{SI-FEIP}}}(x_{\text{target}})$ by the *server* is indistinguishable from the ciphertext c_{target}.

4.2.2 Privacy Analysis

NN-EMD also ensures the privacy of the output of the secure computation protocols. Here, we present two types of inference attacks launched by an ***honest-but-curious server***.

Inference Type I. Note that our proposed *S2PHC* and *S2PVC* protocols adopt functional encryption as the underlying cryptosystems. For both functions, $f_{\text{SIIP}}(\boldsymbol{x}, \boldsymbol{w})$ and $f_{\text{MIIP}}((\boldsymbol{x}_1, ..., \boldsymbol{x}_n), \boldsymbol{w})$ as described in Appendix A, the *server* is able to acquire the decryption results (i.e., the output of the first layer in NN), and the weights of the first layer (i.e., $\boldsymbol{w}$). The security of functional encryption scheme can ensure that the *server* cannot break/infer the input $\boldsymbol{x}$ or $(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$. However, an inference attack may be possible by iteratively employing FE on a specific $\boldsymbol{x}$. Consider the iterative training such that the *curious server* may be able to collect enough polynomial equations for a specific training sample. For instance, suppose we have one training data sample $\boldsymbol{x}$. For each iteration i in the training phase, the *server* is able to acquire $f_i = \langle \boldsymbol{x}, \boldsymbol{w}_i \rangle$, where f_i and $\boldsymbol{w}_i$ are available or visible to the *server*. Obviously, with enough pairs of $(\boldsymbol{w}_i, f_i)$, the *server* is able to solve the linear equation system $\{f_i = \langle \boldsymbol{x}, \boldsymbol{w}_i \rangle\}$ and acquire $\boldsymbol{x}$. Formally, suppose that the sample $\boldsymbol{x}$ has n_{feature} features, i.e., $\boldsymbol{x} = (x_1, x_2, ..., n_{\text{feature}})$, and each sample is used once in one training epoch. Let the total number of training epoch be n_{epoch}, and the number of periodical shuffle operations is n_{shuffle}. We have the following Lemma:

Lemma 2. *NN-EMD is able to prevent Inference Type I, if* $\frac{n_{epoch}}{n_{shuffle}} < n_{feature}$

Proof. Suppose that the *curious server* has advantage ϵ to infer $\boldsymbol{x}$, which indicates it has ϵ advantage to solve the system of linear equation problems $\{f_i = \langle \boldsymbol{x}, \boldsymbol{w}_i \rangle\}$ with determined solution. According to theorem of PSSLS in linear algebra, the *curious server* has the advantage ϵ to collect n_ϵ linear equations for the specific sample $\boldsymbol{x}$, where $n_\epsilon \geq n_{\text{feature}}$.

However, in *NN-EMD*, the *server* has non-negligible advantage to distinguish the ciphertext of $\boldsymbol{x}$ among all encrypted training samples as proved in [4, 5]. After encrypted sample shuffle by the *data source*, the *server* also has non-negligible advantage to learn the position of $\boldsymbol{x}$ in the training set. Thus, the *server* only has the advantage to collect $n_\epsilon = \frac{n_{\text{epoch}}}{n_{\text{shuffle}}}$ linear equations. Here, $\frac{n_{\text{epoch}}}{n_{\text{shuffle}}} < n_{\text{feature}}$ in *NN-EMD* is subject to

the requirement of PSSLS theorem, namely, $n_\epsilon < n_{\text{feature}}$. As a result, the *curious server* has no advantage to infer $\boldsymbol{x}$. □

Inference Type II. The *curious server* could also launch another type of inference attack by specifying "malicious" $\boldsymbol{w}$ to acquire the functional private key. For instance, by specifying $\boldsymbol{w} = (1, 0, ..., 0)$, the decryption result of $\langle \boldsymbol{x}, \boldsymbol{w} \rangle$ will disclose the first element x_1 of $\boldsymbol{x}$. To prevent such an attack, we have introduced *inference weights filter* into the *TPA*. Specifically, the filter module will check the vector $\boldsymbol{w} = (1, 0, ..., 0)$ to ensure that the number of non-zero elements is greater than a threshold τ, basically, $\tau \geq 2$. As a result, it is not possible to launch above inference attack.

4.3 Experimental Evaluation

We evaluate the following aspects of *NN-EMD*: (i) To present the efficiency advantage of training time of our *NN-EMD* framework, we compare its training time with that of only those closely related solutions proposed in [129, 191]. We also explore the impact of network architecture and the number of network layers on training time in our *NN-EMD* framework. (ii) With respect to the trained model accuracy, we compare our *NN-EMD* framework in a HPT setting and a vertical partitining based training setting with a baseline model, namely, a normal DNN without any privacy-preserving settings. (iii) As the underlying cryptosystems only work on the integer field, while the training of DNNs model works on the floating-point number field, we try to evaluate the impact of the precision on the model performance after the numeric encoding/decoding.

4.3.1 Experimental Setup

To benchmark the performance of the *NN-EMD* framework, we train a model of a DNN with the same topology as the one used in [129] on the publicly available MNIST dataset of handwritten digits [106] that includes 60000 training samples and 10000 test samples. In our evaluation, each sample (28×28 image) in the MNIST dataset is mapped to a vector with length 784. Besides, we also explore the framework performance on different DNN architectures and different numbers of network layers. Essentially, we run the experiments for 5 data sources forming the *client pool*. Each data source is randomly assigned $60000/5 = 12000$ data samples from the MNIST dataset for the HPT, while in the VPT, each data source is assigned 60000 data samples but only around $784/5 \approx 157$ features for each sample. We use comparable settings when evaluating the impact of the number of data sources on the model performance. Note that in all the experiments we utilize the same model hyperparameters of a DNN model such as learning rate, l2 regularization parameter, etc.

Table 4.2: Comparison of time cost for training one mini-batch (60 samples)

Proposed Work	Network Architecture	CPU	Threads	Mem	Time
[129]	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.3GHz Intel Xeon E5-2698v3 16-Core	1	250G	$\approx$ 1.5d
[129]	$64 \mapsto 32 \mapsto 16 \mapsto 10$	2.3GHz Intel Xeon E5-2698v3 16-Core	1	250G	9h24m
[129]	$64 \mapsto 32 \mapsto 16 \mapsto 10$	2.3GHz Intel Xeon E5-2698v3 16-Core	30	250G	40m
CryptoNN [191]	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.3GHz Intel Core i7 8-Core	1	16G	$\approx$ 2d
CryptoNN [191]	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.3GHz Intel Core i7 8-Core	8	16G	$\approx$ 94m
NN-EMD (HPT)	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.3GHz Intel Core i9 8-Core	1	32G	**49.83s**
NN-EMD (VPT)	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.3GHz Intel Core i9 8-Core	1	32G	**31.71s**
NN-EMD (HPT)	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.5GHz Intel Xeon 8124M 32 vCPUs	1	128G	**55.63s**
NN-EMD (VPT)	$784 \mapsto 128 \mapsto 32 \mapsto 10$	2.5GHz Intel Xeon 8124M 32 vCPUs	1	128G	**33.67s**

Implementation Consideration. We have implemented the *NN-EMD* framework based on the *NumPy* library to use the high-level mathematical functions in *Python* programming language. The underlying cryptosystems, namely, the functional encryption schemes, are also implemented in Python based on the *gmpy2* library, which is a C-coded Python extension module that supports multiple-precision arithmetic and relies on the GNU multiple precision arithmetic (GMP) library.

Different from the implementation of functional encryption in [191], we incorporate the following acceleration techniques. By tracking the time cost of each decryption step in the functional encryption scheme, we find that the most inefficient computing step is the final step that computes the discrete logarithm of a small integer. To be specific, it involves computing f in $h = g^f$, where h, and g are big integers while f is a small integer. To accelerate such discrete logarithm computations, we employ a *bounded-table-lookup* method by initially setting up a hash table to store pairs of (g, f) with a specified public key parameter g and a positive bound f_b where $-f_b \leq f \leq f_b$. Then, the final discrete logarithm computation is a table look-up operation with complexity $\mathcal{O}(1)$ compared to traditional *baby-step giant-step* algorithm that has complexity $\mathcal{O}(n^{\frac{1}{2}})$.

Environment Setup. All the experiments have been performed on two test platforms: *Test Platform I (TP I)* that is a local Macbook Pro with 2.3GHz Intel Core i9 8-Core CPU and 32GB RAM, and *Test Platform II (TP II)* that is a remote cloud service, i.e., AWS *m5d.8xlarge* instance with 2.5GHz Intel Xeon 8124M 32 vCPUs and 128GB RAM. For the evaluations of model performance, where the client pool and the server are put in the same platform, we repeat the experiments in both the test platforms. To simulate real scenarios, we use *TP I* as the *client pool* and *TP II* as the *server*.

4.3.2 Experimental Results

Comparison with Contracted Frameworks. As shown in Table 4.2, we compare the training time of our *NN-EMD* framework with the approaches proposed in [129, 191]. Note that as the codes and experimental platforms for work in [129] are not publicly available, we report the experimental results reported in [129] directly. We also include the test environment reported in their papers. In our evaluation, we use comparable

Table 4.3: Training time cost of one mini-batch of different network architectures

NN Framework	Network architecture	Local (TP I)	Remote AWS (TP II)
NN-Normal (**Baseline**)	$784 \mapsto 256 \mapsto 10$	0.00718s	0.01169s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 10$	0.00708s	0.01088s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 32 \mapsto 16 \mapsto 10$	0.00741s	0.01083s
NN-EMD (HPT)	$784 \mapsto 256 \mapsto 10$	91.45s	111.50s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 10$	90.54s	111.28s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 32 \mapsto 16 \mapsto 10$	89.66s	111.13s
NN-EMD (VPT)	$784 \mapsto 256 \mapsto 10$	55.58s	67.48s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 10$	55.21s	67.26s
	$784 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 32 \mapsto 16 \mapsto 10$	56.19s	67.05s

† HPT indicates horizontally partitioned based training (HPT) setting, while VPT represents vertically partitioned based training setting. In both of HPT and VPT, there are 5 data sources. Each mini-batch includes *60 samples*;

experimental platforms used in [129, 191], and train the model on the same MNIST dataset with the same DNN architecture.

We evaluate the *NN-EMD* framework both in HPT and VPT settings. Our experimental results show that the training time of one mini-batch including 60 samples in our *NN-EMD* only needs *49.83* seconds and *55.63* seconds in *TP I* and *TP II* environments, respectively. Compared to the existing best result (i.e., 40 minutes) as reported in [129] where each training sample is extracted from 28×28 to 8×8 to reduce the input size and the multithreaded parallelism technique is employed in the training phase, our proposed *NN-EMD* reduces the training time by approximately 90%.

Impact of NN Architectures and Number of Layers. As reported in Table 4.2, the training time of existing solution such as the framework proposed in [129] increases significantly as the network architecture changes. To evaluate the impact of network architectures on the training time in our *NN-EMD* framework, we train DNN models with different architectures on the MNIST dataset with the same number of data sources. As presented in Table 4.3, the training time for our proposed approach is only impacted by the number of nodes in the first hidden layer. When the network architecture of the rest of the layers changes, the training time does not change compared to the normal DNNs without privacy-preserving setting.

For further verification of such a claim, we conducted additional experiments with a large number of hidden layers. As shown in Figure 4.5 (c), we measure the training time of one mini-batch in our *NN-EMD* framework in bother settings (i.e., HPT and VPT) and vary the number of hidden layers from 1 to 30 where each layer includes 64 neural nodes. As can be seen, the training time does not change drastically like in existing solutions as the number of hidden layers increases.

Evaluation of Accuracy. Except for the performance with respect to the training time, we compare our framework with a baseline DNN (Normal-NN) that has the same network architecture but without any privacy-preserving settings. As shown in Figure 4.5 (a), our proposed *NN-EMD* framework can achieve model accuracy comparable to a normal DNN both in HPT and VPT. Further, the results in Figure 4.5

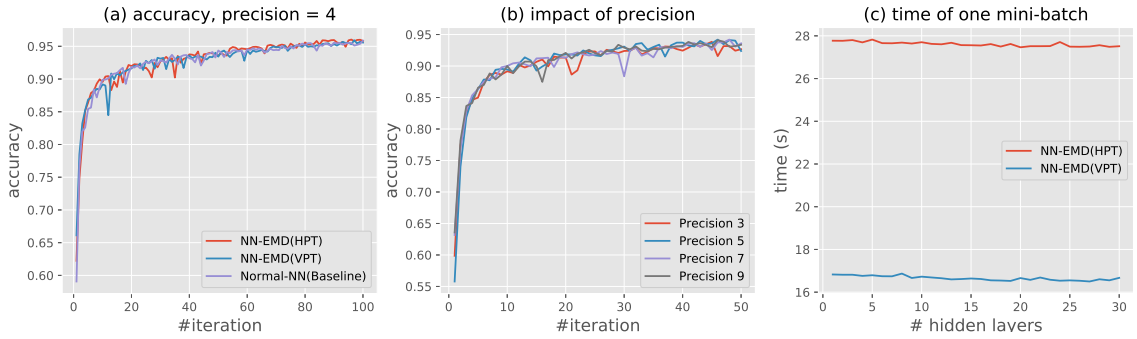


Figure 4.5: Model accuracy, impact of encoding precision and impact of hidden layers.

(b) shows that the precision setting does not have an effect on the model accuracy. Note that the network architecture used for model accuracy comparison is $784 \mapsto 512 \mapsto 256 \mapsto 128 \mapsto 64 \mapsto 32 \mapsto 10$. Each hidden layer of network architecture in right figure includes 64 neural nodes and the results are generated on the *TP II* platform.

Deployment in Client-Server Scenario. To evaluate the impact of the number of data sources on the training time, we have deployed our end-to-end *NN-EMD* system in a *client-server* scenario. In this experiment, our local machine (*TP I*) plays the role of *client pool* with varying number of data sources to pre-process the encrypted training datasets, while the remote AWS instance (*TP II*) plays the role of the *server* to train the DNN model based on these encrypted data samples.

As shown in Table 4.4, we present the training time for both the *client pool* and the *server*. All reported times for the *server* side is based on one mini-batch, while the time reported for the *client pool* is for one mini-batch per data source. In the case of the HPT, the training time of *NN-EMD* framework does not change drastically like existing solutions as the number of data sources increases. In the case of the VPT, each data source pre-processes a same number of data samples. As the total number of features is fixed, the number of features from each data source decreases as the number of data sources increase, and hence the pre-processing time decreases, while the training time still does not change drastically.

4.4 Summary and Discussion

Training DNN models over encrypted data show significant promise towards addressing strong privacy requirements, while taking full advantage of an existing ML platform as a service infrastructure. However, there is a lack of efficient and practical privacy-preserving solutions for training a DNN over privacy-sensitive datasets. In this chapter, we have proposed *NN-EMD*, a novel DNN that supports training a DNN model

Table 4.4: Time cost for different data source numbers in client-server setting

Training	Data Sources	Pre-process Time - Client (TP I)	Training Time - Remote AWS (TP II)
HPT Setting	5	1.2769s	56.02s
	7	1.0579s	55.95s
	9	1.1195s	55.86s
	11	1.1246s	55.87s
	13	1.1463s	56.04s
	15	1.1260s	56.13s
VPT Setting	5	0.3716s	33.75s
	7	0.3363s	32.48s
	9	0.2372s	31.93s
	11	0.2214s	31.50s
	13	0.1867s	31.24s
	15	0.1711s	31.17s

† The DNNs architecture used in this experiment is $784 \mapsto 128 \mapsto 32 \mapsto 10$. Note that the cost time reported here is for only one mini-batch that includes 60 samples.

on a dataset where the data is composed, both horizontally and vertically, of encrypted datasets from multiple data sources. Our evaluation shows that *NN-EMD* can reduce the training time by 90% while still providing the same model accuracy and strong privacy guarantee as compared to the most of the recent comparable approaches. Furthermore, the depth and complexity of DNNs do not affect the training time despite introducing a privacy-preserving *NN-EMD* setting. Future work includes applying the *NN-EMD* framework in a more complex distributed environments such as an edge computing environment.

5.0 Practical Secure Aggregation in Edge Computing

Recently, edge computing has attracted significant industry investment and efforts from research communities [165, 164]. For instance, the Open Edge Computing (OEC) initiative [48] was launched in June 2015 by Vodafone, Intel, and Huawei in partnership with Carnegie Mellon University (CMU), and then expanded to include Verizon, T-Mobile, Nokia, etc. Emerging edge computing technologies promise to deliver highly responsive and scalable cloud services, and enhance preprocessing services for the Internet of Things (IoT) [157]. Such enhanced preprocessing services of edge computing primarily include operations such as cleaning, formatting, filtering, and aggregation of the raw data generated by the IoT and mobile devices. These benefits are achieved by appropriately placing edge nodes with computing and storage capabilities at the edge of the Internet closer to mobile devices or sensors. Aggregation is a common and typical operation in edge and cloud computing environments [163, 148, 112, 140, 195, 174, 177, 162, 196]. Usually, rather than uploading an entire dataset, it may be sufficient to compute and upload its statistical results to the cloud; this will reduce various processing, data transfer or storage costs. For instance, in a temperature sensor based application, rather than collecting the temperature and transmitting it to the cloud every second, an average temperature over one hour period for a target environment can be computed and uploaded. Computation of such an average is a form of an aggregation.

Huge amounts of collected data in device rich environments is often subject to privacy or regulatory requirements that restrict the way data can be shared, transmitted, and used. Furthermore, while cloud servers in cloud data centers may follow strict policies and regulations, edge nodes may not have the same degree of regulatory and monitoring oversight. Especially, these edge nodes deployed in the public area for public edge computing services. A motivating example here could be a self-driving delivering system in a smart city environment, where several self-driving trucks from different commercial companies serve their customers. To acquire a more intelligent self-driving model, those self-driving trucks can collaboratively train an ML model in a federated learning manner without leaking each company's private commercial data such as the destination of served customers and delivering paths. To accelerate the training phase (e.g., FL training in real-time), these self-driving trucks can take advantage of these edge nodes deployed in the places such as traffic intersections, which is a part of a public infrastructure in a smart city. However, these edge nodes may not fully monitored and hence not fully trusted as are cloud servers by the commercial companies to process their self-driving model.

To address privacy concerns discussed above, various aggregation approaches (i.e., privacy-preserving aggregation and secure aggregation) have been proposed in the literature. These approaches focus on protecting the raw data by sanitizing or perturbing privacy sensitive information [163, 148, 174], or enhancing the data processing procedure to prevent a third-party from acquiring the privacy sensitive information [195, 112, 196, 162]. These approaches include data anonymization mechanisms (e.g., k-anonymity,

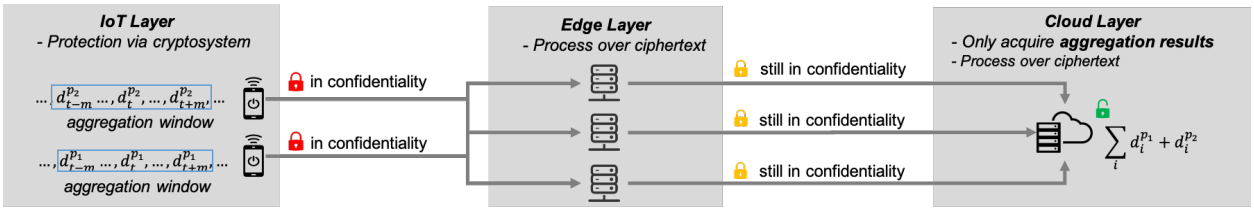


Figure 5.1: Illustration of the three-tier *CryptoEdge* platform.

l-diversity, differential privacy, etc.), secure multi-party computation (SMC) protocols, and cryptographic schemes. Compared to differential privacy based privacy-preserving aggregation, crypto-based secure aggregation approaches ensure that the original data is protected by the cryptosystem and the aggregation results are accurate.

Newly emerging cryptographic schemes such as homomorphic encryption and functional encryption have shown tremendous promise of achieving practical computation over encrypted data[129, 191]. However, most of the existing homomorphic encryption (HE) schemes or functional encryption (FE) schemes are not applicable in edge computing environments because of the following reasons: (i) the constructions of homomorphic encryption are neither efficient enough for complex computation such as privacy-preserving FL nor computationally-friendly with regards to capabilities of the IoT devices; (ii) existing FE solutions do not support three-tier edge computing based architecture; in particular if we directly apply existing FE schemes, *untrusted* edge nodes can either (a) acquire the functional results in plaintext resulting in privacy-disclosure to the *untrusted* edge nodes, or (b) not process the encrypted data at all. Thus, achieving practical secure aggregation the encrypted data in within a three-tier edge computing architecture is still a huge challenge.

In this chapter, we propose *CryptoEdge* to support practical secure aggregation over encrypted data at the edge. The approach is illustrated in Figure 5.1. Here, *IoT* devices can easily encrypt the raw data; the *untrusted* edge nodes have the capability to process the encrypted data (i.e., partially decrypt the encrypted raw data) without learning any privacy sensitive information from the data; and the cloud data center then can obtain the final aggregation results, but it still cannot learn any privacy sensitive information. In general, the cloud center can randomly specify the number of edge nodes to collaboratively *pre-process* the encrypted data and then combine the processed results to compose final aggregation results. As a result *CryptoEdge* can dramatically reduce the overall decryption time and save computational resources for the cloud, as the edge nodes are able to do preprocessing over the encrypted data.

The proposed *CryptoEdge* approach includes a novel *threshold functional encryption (TFE)* scheme, a *secure aggregation* module that supports various types of aggregation demands; and a PPFL framework for the edge environment. The goal of threshold setting in FE allows all participant edge nodes to enroll in the aggregation of encrypted data or local model without knowing the final aggregated result, while the cloud

server can acquire the final aggregated result by calling final decryption algorithm on a number of partially aggregated encrypted data (still in confidentiality) from selected edge nodes.

5.1 Threshold Functional Encryption Scheme

5.1.1 Motivation for TFE

To achieve the goals of *CryptoEdge*, namely, practical secure computation over encrypted data in three-tier edge computing architecture, we need crypto schemes that meet the following design requirements:

- The crypto scheme should support *practical aggregation* over the encrypted data; i.e., it should be efficient enough for practical deployment in edge computing environments;
- The responsibilities related to encryption, computation, and decryption should be given to separate parties; for instance, *IoT* devices are issued keys to encrypt the data; *edge nodes* are issued keys to compute over encrypted data; and *cloud servers* are able to acquire the final computation result by decrypting the encrypted data.

Existing HE and FE schemes are two promising candidates that support computation over the encrypted data.

We do not pursue HE for the following reasons: (i) As shown in [189] and [129], existing constructions of HE are not efficient enough to support complex applications over encrypted data; hence, they are not computationally friendly for IoT devices; (ii) As encryption/decryption keys in a HE scheme are generated in pairs, its adoption in edge computing environments will not fulfill the second design requirement, and hence, it will introduce extra key management burden into the system.

FE is a promising approach for secure computation over encrypted data as the encryption and decryption keys are independently issued to different parties. The recent construction of FE for the functionality of the inner-product, such as the one proposed in [165, 5], shows the possibility of using the basic DDH assumption that is more appropriate for IoT devices as compared to pairing-based and garbled circuits based cryptosystems. Furthermore, these FE constructions have been proved to be promising approaches for building practical privacy-preserving ML applications [191, 189] in two-tier cloud-based architecture. In summary, there is a need for novel cryptosystems that satisfy the above-mentioned design requirements. We propose novel threshold functional encryption schemes that address the challenges mentioned above.

5.1.2 Definition of TFE

Notations. Let $GroupGen(1^\lambda)$ be a probabilistic polynomial-time algorithm that takes as input a security parameter 1^λ, and outputs a triplet $(\mathbb{G})$, where $\mathbb{G}$ is a group of order p that is generated by $g \in \mathbb{G}$, and p is a λ-bit prime number. Furthermore, let $r \leftarrow_\$ \mathbb{Z}_p$ denote the assignment to r of an element chosen uniformly

at random from integer group $\mathbb{Z}_p$. We use $[n]$ to denote a set of n sequential natural numbers $\{1, 2, ..., n\}$. The lowercase bold variable such as $\boldsymbol{\alpha}^{1\times\eta}$ represents a vector with length, η. A capital bold variable such as $\boldsymbol{W}^{n\times\eta}$ denotes a matrix with n rows and η columns.

We define *threshold functional encryption (TFE) for functionality $\mathcal{F}$* scheme as follows.

Definition 1 (Threshold Functional Encryption Scheme (TFE))**.** *A t-of-s threshold functional encryption for functionality $\mathcal{F}$ is a tuple (Setup, PKDistribute, SKDistribute, Encrypt, ShareDecrypt, CombineDecrypt) of six algorithms, written as*

$$\mathcal{E}^{\mathcal{F}}_{TFE} = (S, PK, SK, E, SD, CD). \tag{5.1}$$

Each algorithm is defined as follows:

- *Setup(λ) outputs master public and master secret keys, (mpk,msk) based on security parameter, λ.*
- *PKDistribute(mpk, msk, idx) distributes public key, pk_{idx}, for entity idx on input master public and secret keys, (mpk, msk).*
- *SKDistribute($mpk, msk, \mathcal{K}, idx$) distributes secret key, sk_{idx}, for an entity, idx, on input master public and secret keys, (mpk, msk), and vector or matrix from $\mathcal{K}$.*
- *Encrypt($pk, \mathcal{X}$) outputs ciphertext ct on input vector or matrix from $\mathcal{X}$ and public key, pk.*
- *ShareDecrypt($pk, ct, \mathcal{K}, \{sk_{idx}\}, S$) outputs a partially decrypted ciphertext, $ct^{'}$, on inputs: a ciphertext, ct, a public key, pk, a vector or matrix from $\mathcal{K}$ and functional private key, $\{sk_{idx}\}$, from a selected sub-set of decryption parties.*
- *CombineDecrypt($pk, ct^{'}$) outputs functionality result on input public key, pk, and a collected partially decrypted ciphertext $ct^{'}$.*

5.1.3 Threshold FE Scheme for Functionality of Multi-Client Inner-Product

Here, we present our proposed *t-of-s threshold functional encryption* scheme for functionality of multi-client inner-product (MCIP) $\mathcal{F}_{\text{MCIP}}$, whose security is based on the plain *Decisional Diffie-Hellman* (DDH) assumption; here, t of s decryption parties are allowed to collaboratively acquire an inner-product over multiple encrypted vectors.

Functionality of $\mathcal{F}_{\text{MCIP}}$. In this chapter, we mainly focus on an *inner-product functionality* over integers. Let $\mathcal{F}_{\text{IP}}$ be a family of *inner-product functionality* with message space $\mathcal{X}$ and key space $\mathcal{K}$ both consisting of vectors in $\mathbb{Z}_p^{\eta}$ of norm bounded by p of length η. Here, we focus on *multiple inputs* inner-product $\mathcal{F}_{\text{MCIP}}$, defined as follows:

$$f_{\text{MCIP}}(\{\boldsymbol{x}_i\}, \boldsymbol{y}) = \sum_{i\in[n]} \sum_{j\in[\eta_i]} (x_{ij} y_{\sum_{k=1}^{i-1}\eta_k + j}) \quad \text{s.t.} \quad |\boldsymbol{x}_i| = \eta_i, |\boldsymbol{y}| = \sum_{i\in[n]} \eta_i, \tag{5.2}$$

where $f_{\text{MCIP}} \in \mathcal{F}_{\text{MCIP}}$, $\boldsymbol{x}_i \in \mathcal{X}$, and $\boldsymbol{y} \in \mathcal{K}$. Also, the length of $\boldsymbol{x}_i$ should be equal to the length of vector $\boldsymbol{y}$.

Construction. The specific construction of *TFE* scheme for $\mathcal{F}_{\text{MCIP}}$ is defined as follows:

- *Setup(λ, η, t, s, n)*: This algorithm first generates a triplet from the integer group, as $(\mathbb{G}, p, g)$, on given security parameter λ as input. Then, it randomizes a matrix of $1 \times \eta$ samples and two matrix samples with size $n \times \eta$, as follows:

$$\boldsymbol{\alpha}^{1\times\eta}, \boldsymbol{W}^{n\times\eta}, \boldsymbol{U}^{n\times\eta} \leftarrow_{\$} \mathbb{Z}_p. \tag{5.3}$$

The master public key and master private key are defined as follows:

$$msk = (\boldsymbol{W}^{n\times\eta}, \boldsymbol{U}^{n\times\eta}) \tag{5.4}$$

$$mpk = (\mathbb{G}, p, g, g^{\boldsymbol{\alpha}}, \{g^{\boldsymbol{\alpha}^\intercal \boldsymbol{W}_i}\}_{i\in[n]}) \tag{5.5}$$

Note that $\boldsymbol{\alpha}^\intercal \boldsymbol{W}_i$ denotes that the transpose of $\boldsymbol{\alpha}$ multiplies i-th row of $\boldsymbol{W}^{n\times\eta}$.

- *PKDistribute(mpk, msk, idx)*: Given the master public and secret keys, for encryption entity idx, this algorithm distributes the public keys as follows:

$$pk_{idx} = (\mathbb{G}, p, t, s, n, g, g^{\boldsymbol{\alpha}}, g^{\boldsymbol{\alpha}^\intercal \boldsymbol{W}_{idx}}, \boldsymbol{U}_{idx}). \tag{5.6}$$

The algorithm distributes the common public key $pk_{com} = (\mathbb{G}, p, g)$ to the decryption entities.

- *SKDistribute($mpk, msk, \boldsymbol{y}, idx$)*: The algorithm takes master public and secret keys, $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n)$, and distributes the functional private key to the corresponding decryption entity idx. This algorithm first defines a set of polynomial functions $(f^{(0)}(x), \{f^{(i)}(x)\}_{i\in[1,...,n]})$ as follows:

$$f^{(0)}(x) = \sum_{j=0}^{t-1} a_j x^j, \text{ where } a_0 \leftarrow \sum_{i=1}^{n} \langle \boldsymbol{y}_i, \boldsymbol{U}_i \rangle, a_{j\in[1,...,t-1]} \leftarrow_{\$} \mathbb{Z}_p \tag{5.7}$$

$$f^{(i)}(x) = \sum_{j=0}^{t-1} b_{i,j} x^j, \text{ where } b_{i,0} \leftarrow \langle \boldsymbol{y}_i, \boldsymbol{W}_i \rangle, b_{i,j\in[1,...,t-1]} \leftarrow_{\$} \mathbb{Z}_p. \tag{5.8}$$

It then generates a set of functional secret keys as follows:

$$\boldsymbol{sk} = \{v_{k,0} \leftarrow f^{(0)}(k), v_{k,1} \leftarrow \{f^{(i)}(k)\}_{i\in[n]}\}_{k\in[s]}. \tag{5.9}$$

It provides the functional private key, $sk_{idx} = (v_{idx,0}, v_{idx,1})$ to the corresponding partial decryption entity, idx,

- *Encrypt($pk_{idx}, \boldsymbol{x}_{idx}$)*: For encryption entity $idx \in [n]$, the algorithm takes as input corresponding pk_{idx} and $\boldsymbol{x}_{idx}$, and returns ciphertext ct. It first chooses a random element $r_{idx} \leftarrow_{\$} \mathbb{Z}_p$ and computes ciphertext $ct_{idx} = (ct_{idx,0}, ct_{idx,1})$ as follows:

$$ct_{idx,0} = g^{\boldsymbol{x}_{idx} + \boldsymbol{U}_{idx}} \circ (g^{\boldsymbol{\alpha}^\intercal \boldsymbol{W}_{idx}})^{r_{idx}}, \tag{5.10}$$

$$ct_{idx,1} = (g^{\boldsymbol{\alpha}})^{r_{idx}} = \prod_{k\in[\eta]} g^{\alpha_k r_{idx}}. \tag{5.11}$$

Note that symbol $\circ$ denotes an element-wise multiplication. For instance, $\boldsymbol{x}^{1\times\eta} \circ \boldsymbol{Y}^{\eta\times\eta}$ denotes the element-multiply computation of $\boldsymbol{x}$ and each row of $\boldsymbol{Y}$.

- *ShareDecrypt*($pk_{com}, \boldsymbol{ct}, \boldsymbol{y}, \{sk_j\}_{j\in S}, S$): This algorithm takes ciphertext $\boldsymbol{ct} = \{ct_{i,0}, ct_{i,1}\}_{i\in[n]}$, the common public key pk_{com} and vector $\boldsymbol{y} = \{\boldsymbol{y}_i\}_{i\in[n]}$ associated functional key sk_i from an authorized sub-set S, where $|S| \geq t$. To share with decryption entity $i \in S$, it outputs partially decrypted ciphertext $ct_j^{'} = (ct_{j,0}^{'}, ct_{j,1}^{'}, ct_{j,2}^{'})$ as follows:

$$ct_{j,0}^{'} = \prod_{i\in[n]} ct_{i,0}^{\circ \boldsymbol{y}_i}, \tag{5.12}$$

$$ct_{j,1}^{'} = \{(ct_{i,1})^{v_{j,1}L_j(j)}\}_{i\in[n]}, \tag{5.13}$$

$$ct_{j,2}^{'} = g^{v_{j,0}L_j(j)}, \tag{5.14}$$

where $L_j(j)$ is the Lagrange basis polynomials defined as $\prod_{j'\in S, j'\neq j} \frac{-j'}{j-j'}$.

- *CombineDecrypt*($pk_{com}, \boldsymbol{ct}^{'}$): This algorithm takes all received ciphertext $\boldsymbol{ct}^{'}$ and returns the inner-product $\langle\{\boldsymbol{x}_i\}_{i\in[n]}, \boldsymbol{y}\rangle$. $\forall ct_{i,0}^{'} \in \boldsymbol{ct}^{'}$; it tries verify that they are all equal. If the ***verification*** fails, it returns the stop symbol; otherwise, let $C = ct_{j,0}^{'}$. Then, it returns the final combined decryption results as follows:

$$D = \frac{C}{\prod_{i\in[n]} \prod_{j\in|\boldsymbol{ct}'|} ct_{i,j,1}^{'} \cdot \prod_{j\in|\boldsymbol{ct}'|} (ct_{j,2}^{'})^2} \tag{5.15}$$

Finally, $f_{\text{MCIP}}(\{\boldsymbol{x}_i\}_{i\in[n]}, \boldsymbol{y})$ can be recovered by computing $\frac{1}{2}\log(D)$.

Correctness. Given the common public key pk_{com} and the collected partially decrypted ciphertext $\boldsymbol{ct}^{'}$, we have that

$$D = \frac{\prod_{i\in[n]} ct_{i,0}^{\boldsymbol{y}_i}}{\prod_{i\in[n]} \prod_{j\in|\boldsymbol{ct}'|} ct_{i,j,1}^{'} \cdot \prod_{j\in|\boldsymbol{ct}'|} (ct_{j,2}^{'})^2} \tag{5.16}$$

$$= \frac{\prod_{i\in[n]} (g^{\boldsymbol{x}_i + \boldsymbol{U}_i} \circ (g^{\boldsymbol{\alpha}^\intercal \boldsymbol{W}_i})^{r_i})^{\boldsymbol{y}_i}}{\prod_{i\in[n]} (ct_{i,1})^{\sum_{j\in|\boldsymbol{ct}'|} f^{(i)}(j)L_j(j)} \cdot (ct_{j,2}^{'})^{2\sum_{j\in|\boldsymbol{ct}'|} f^{(0)}(j)L_j(j)}} \tag{5.17}$$

$$= \frac{\prod_{i\in[n]} (g^{\boldsymbol{x}_i + \boldsymbol{U}_i} \circ (g^{\boldsymbol{\alpha}^\intercal \boldsymbol{W}_i r_i})^{\boldsymbol{y}_i}}{\prod_{i\in[n]} (g^{r_i\boldsymbol{\alpha}})^{f^{(i)}(0)} \cdot g^{2f^{(0)}(0)}} \tag{5.18}$$

$$= \frac{\prod_{i\in[n]} g^{(\boldsymbol{x}_i + \boldsymbol{U}_i)\circ \boldsymbol{y}_i}}{g^{2\sum_{i=1}^{n} \langle \boldsymbol{y}_i \boldsymbol{U}_i\rangle}} \tag{5.19}$$

$$= \frac{g^{2\sum_{i=1}^{n} \langle \boldsymbol{x}_i \boldsymbol{y}_i\rangle} \cdot g^{2\sum_{i=1}^{n} \langle \boldsymbol{y}_i \boldsymbol{U}_i\rangle}}{g^{2\sum_{i=1}^{n} \langle \boldsymbol{y}_i \boldsymbol{U}_i\rangle}} \tag{5.20}$$

$$= g^{2f_{\text{MCFE}}(\{\boldsymbol{x}_i\}_{i\in[n]}, \boldsymbol{y})} \tag{5.21}$$

Security. For the security of *TFE* scheme for $\mathcal{F}_{\text{MCIP}}$, we employ the security definition, namely, *selective simulation-based security* (SEL-SIM security) as in [5]. Here, we propose Lemma 3 as follows. The specific proof will be presented in Section 5.3.1.

Lemma 3. *Assume a static adversary that corrupts maximum of t-1 players from the beginning; then, under the DDH assumption, the TFE scheme for $\mathcal{F}_{MCIP}$ achieves selective simulation-based security. Furthermore, the non-authorized player is not able to acquire the functionality result.*

5.2 *CryptoEdge* Platform

5.2.1 Overview of *CryptoEdge*

Our proposed *CryptoEdge* platform supports various applications such as (i) infrastructure-level applications that need to compute various types of secure aggregates (CryptoEdge-SA); and (ii) high-level applications that employ a privacy-preserving edge-based federated learning (CryptoEdge-PPFL) framework. In particular, *CryptoEdge-SA* applications are built on our proposed *TFE* scheme, while *CryptoEdge-PPFL* framework is built on the *CryptoEdge-SA* applications.

Architecture and Entities. As illustrated in Figure 5.1, the proposed *CryptoEdge* platform includes three tiers that include *IoT devices* layer, *edge nodes* layer, and *cloud service* layer, with following entities.

- *Third-party Authority (TPA)* is responsible for setting up the underlying cryptosystem, delivering the public key to each *IoT devices* and providing private key service to the *edge nodes* for preprocessing. The TPA is also in charge of holding the master private and public keys and is also trusted to perform distribution of public-keys and generation of a function derived secret key .
- *IoT Devices* generate stream data such as healthcare data and location data. Such privacy-sensitive data is encrypted using the key issued by the *TPA* before they are sent out. Furthermore, the *IoT devices* have limited computational capabilities and hence cannot support the complex encryption algorithm involved.
- *Edge Nodes* primarily process the encrypted data from the *IoT devices* using the key issued by the *TPA*. These *edge nodes* do not learn any information during the data processing step and the confidentiality of the processed data is maintained.
- *Cloud Service Provider* does the final decryption on the partially decrypted data collected from these *edge nodes* with the key issued by the *TPA*. Furthermore, the *cloud center* is only able to acquire the result of a function over the encrypted data rather than the original encrypted stream data.

Remark. As it is part of the cryptosystem, the TPA is not illustrated in Figure 5.1. In real-world scenarios, different application domains/environments already have entities that can take the role of a TPA. For example, central banks of the banking industry often play a fully trusted role, and some other companies in other sectors such as a service or consultancy firm can embody the TPA. Note that *the TPA has NO access to raw data generated by the IoT devices* in the *CryptoEdge* platform.

Threat Model and Assumptions. In *CryptoEdge* platform, we consider the following threat model and assumptions. A *TPA* is an independent entity widely *trusted* by all the entities in the *CryptoEdge* platform. Note that assuming such a trusted and independent entity is common in existing cryptosystems that employ a TPA as a key component [25, 4, 5]. A *limited number* of *edge nodes* can be considered *unreliable* or *untrustworthy* by the *IoT* and the *cloud* layers. We assume that a limited number of edge nodes may be controlled by an adversary with a goal of inferring private information while processing data. Note that the denial-of-service attack is not considered in *CryptoEdge*; i.e, we assume that the enrolled edge node do not

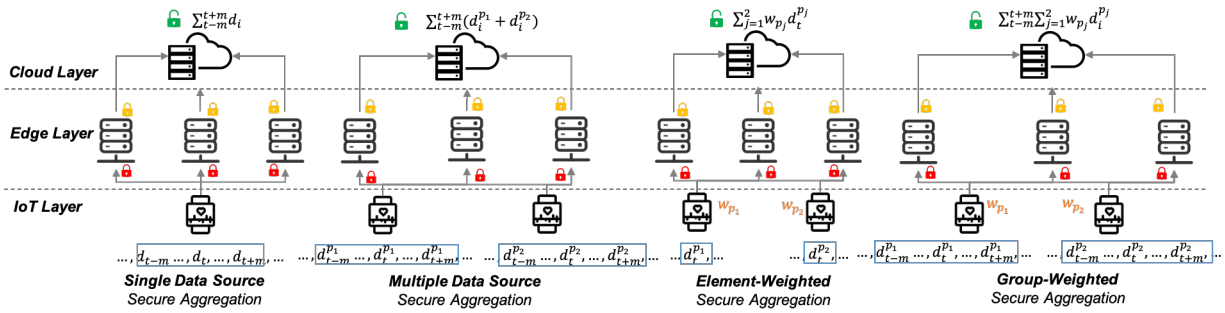


Figure 5.2: Illustration of various types of secure aggregation scenarios supported by the *CryptoEdge*

stop or exit during the execution of an algorithm or a protocol. A *cloud center* is assumed to be *honest-but-curious*; i.e., the cloud center will follow the designed algorithm/protocol but will attempt to learn all possible private information from the received encrypted data or final decrypted function result.

Platform Statement. Suppose that the *CryptoEdge* platform includes one *cloud center*, $\mathcal{P}^{\text{cloud}}$, s edge nodes, $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[s]}$, with at least *t-of-s trustworthy* edge nodes, and n IoT devices $\{\mathcal{P}_i^{\text{iot}}\}_{i\in[n]}$. With the issued public key $pk_{\mathcal{P}_i^{\text{iot}}}$, $\mathcal{P}_i^{\text{iot}}$ protects its data, denoted as $\mathcal{D}^{\mathcal{P}_i^{\text{iot}}} = \{d_k^{\mathcal{P}_i^{\text{iot}}}\}_{k\in S^{(t)}}$. For edge node $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[s]}$, with the functional private key $sk_{\mathcal{P}_j^{\text{edge}}}$ issued by the TPA, $\mathcal{P}_j^{\text{edge}}$ is able to process (i.e., partially decrypt) the encrypted data received from the IoT devices. The cloud center $\mathcal{P}^{cloud}$ can randomly select $t^{'} > t$ edge nodes to collaboratively compute the final functionality result over the IoT device's encrypted data, where t is the threshold in the *TFE* scheme. In particular, the edge nodes $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[t']}$ selected process the encrypted data as discussed above. Then, $\mathcal{P}^{cloud}$ combines these partially decrypted data and do the final decryption to acquire the functionality result.

Note that we have omitted the key issuance phase in the description; in this phase, each entity is issued a common public key, IoT-entity-specific public key, or functional private key as introduced in Section 5.1.

5.2.2 *CryptoEdge-SA*: Secure Aggregation at the Edge

Unlike privacy-preserving approaches proposed in [163, 148, 174], our proposed *CryptoEdge* platform supports *secure aggregation* where the original data is protected by the cryptosystem and the aggregation results are accurate. Figure 5.2 illustrates a set of various types of secure aggregation applications supported by *CryptoEdge*. The single data source SA is a simplified version of multiple data sources SA, and the element-weighted SA is also a simplified version of group-weighted SA, where the sliding window is reduced to one element. Thus, we only present the detailed secure aggregation approaches of later ones in the following subsections.

Secure Aggregation over Multiple Data Sources. The secure aggregation in the case of multiple data

sources attempts to securely aggregate the stream data for a specific sliding window from multiple IoTs. Each entity's operations are presented as follows:

- Entity $\mathcal{P}^{\text{iot}}$. Each IoT device $\{\mathcal{P}_i^{\text{iot}}\}_{i\in[n]}$ in the group of data sources encrypts its dataset $\mathcal{D}^{\mathcal{P}_i^{\text{iot}}} = \{d_k^{\mathcal{P}_i^{\text{iot}}}\}_{k\in S}$ with the issued public key $pk^{\mathcal{P}_i^{\text{iot}}}$, and generates the ciphertext as follows:

$$\mathcal{C}^{\mathcal{P}_i^{\text{iot}}} = \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCIP}}}.E_{pk^{\mathcal{P}_i^{\text{iot}}}}(\mathcal{D}^{\mathcal{P}_i^{\text{iot}}}). \tag{5.22}$$

  Then, each $\{\mathcal{C}^{\mathcal{P}_i^{\text{iot}}}\}_{i\in[n]}$ is emitted to nearby group of edge nodes $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[s]}$.
- Entity $\mathcal{P}^{\text{edge}}$. Each edge node $\mathcal{P}_j^{\text{edge}}$ in the selected group $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[t']}$ processes the encrypted data with the issued functional private key $sk^{\mathcal{P}_j^{\text{edge}}}$, and sends to the cloud with the processed ciphertext as follows:

$$\mathcal{C}_{\text{partial}}^{\mathcal{P}_j^{\text{edge}}} = \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCIP}}}.SD_{sk^{\mathcal{P}_j^{\text{edge}}}}(\{\mathcal{C}^{\mathcal{P}_i^{\text{iot}}}\}_{i\in[n]}). \tag{5.23}$$

- Entity $\mathcal{P}^{\text{cloud}}$. The cloud collects the processed $\{\mathcal{C}_{\text{partial}}^{\mathcal{P}_j^{\text{edge}}}\}_{j\in[t']}$ from the group of edge nodes. With the final decryption, the aggregation is achieved as follows:

$$\sum_{i\in[n]}\sum_{k\in S} d_k^{\mathcal{P}_i^{\text{iot}}} = \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCIP}}}.CD(\{\mathcal{C}_{\text{partial}}^{\mathcal{P}_j^{\text{edge}}}\}_{j\in[t']}). \tag{5.24}$$

Remark. Here, we use our proposed *TFE* scheme as an illustration for secure aggregation over multiple data sources. Note that the threshold Paillier cryptosystem is also able to address the same secure aggregation task, and hence, we take the Paillier system as the baseline to compare with.

Group-Weighted Secure Aggregation. Unlike simple aggregation discussed in Section 5.2.2, our *CryptoEdge* platform also supports the weighted-aggregation that is also a typical aggregation; for instance, a healthcare application may need to apply a set of coefficients on the time-series heart rate data generated by smartwatch monitors. Each entity's operations are presented as follows:

- Entity $\mathcal{P}^{\text{iot}}$. Each $\mathcal{P}^{\text{iot}}$ does encryption as in Section 5.2.2.
- Entity $\mathcal{P}^{\text{edge}}$. Each edge node $\mathcal{P}_i^{\text{edge}}$ in the selected group $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[t']}$ receives the coefficients $\boldsymbol{w} = \{w_k\}_{k\in[n]}$ delivered by the $\mathcal{P}^{\text{cloud}}$. Then, $\mathcal{P}^{\text{edge}}$ acquires the issued $\boldsymbol{w}$ related functional private key $sk_{\boldsymbol{w}}^{\mathcal{P}_j^{\text{edge}}}$ from the TPA using the coefficients $\boldsymbol{w}$. Finally, each edge node does processing over encrypted data as in Section 5.2.2
- Entity $\mathcal{P}^{\text{cloud}}$. The cloud specifies the coefficients $\boldsymbol{w}$ for each element in $\mathcal{D}^{\mathcal{P}^{\text{iot}}}$ and delivers to the selected edge nodes group $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[t']}$. And then, it does aggregate compuatation as in Section 5.2.2 to acquire the element-weighted aggregation $\sum_{k\in S} d_k^{\mathcal{P}^{\text{iot}}} w_k$.

Remark. Note that the element-weighted secure aggregation is omitted here, and it can also be applied here by reducing the sliding window to 1, namely, $\mathcal{D}^{\mathcal{P}_i^{\text{iot}}} = \{d_k^{\mathcal{P}_i^{\text{iot}}}\}_{k\in S}, |S| = 1$. Furthermore, note that element-weighted secure aggregation is much more *fine-grained* than the group-weighted secure aggregation, which is a *coarse-grained*, high-level aggregation. The element-weighted secure aggregation can be applied to both the cases of single data source and multiple data sources, while the group-weighted secure aggregation only works on the later one.

5.2.3 *CryptoEdge-PPFL*: Privacy-Preserving FL at the Edge

Privacy-Preserving FL: from Two-Tier to Three-Tier Architecture. FL approaches have been recently studied to address privacy concerns by allowing collaborative training of ML models among multiple participants where each participant can keep its data private. However, this approach still poses privacy risks such as inference attacks [131, 167]. To address such privacy leakage, several techniques have been adopted in PPFL. For instance, hybrid approaches, such as those proposed in [141, 173, 189], combine differential privacy techniques and secure multiparty aggregation techniques [189, 22].

In traditional two-tier FL architecture, each participant trains a model locally and exchanges only model parameters with others, instead of the active privacy-sensitive training data, with the coordination of a central entity called *coordinator*. In particular, the *coordinator* merges the model parameters collected by each participant, and then distribute the aggregated model to each participant for the next round of training. Existing secure aggregation solutions as proposed in [189, 22], however, do not support the three-tier edge computing architecture, where the *participants* (i.e., the IoTs) encrypt its local model updates, the edge nodes help process the encrypted model update (i.e., partial decryption), and only the *coordinator* (i.e., the cloud) is able to acquire the final aggregated model update. To the best of our knowledge, the secure aggregation approach adopted in [173] is based on the threshold Paillier system that is actually applicable in the three-tier edge-based PPFL systems. However, the approach proposed in [173] relies on the IoTs (i.e., participants) instead of the edge nodes to collaboratively decrypt the final aggregated model. Specifically, such a threshold Paillier based secure aggregation includes two steps: the *coordinator* combines all encrypted local model from all participants; and then the deliver the combined (encrypted) to each participant for collaborative decryption. Our proposed *CryptoEdge-PPFL* does not rely on such combination related communications.

To take advantage of the edge facilities, here, we propose our efficient solution for the privacy-preserving edge-based FL, wherein the multiple data sources based *CryptoEdge-SA* as proposed in Section 5.2.2 is adopted. Similar to the privacy-preserving FL approaches as proposed in [173, 189], where each participant adds its own *local differential privacy* noise independently and then uses *secure multi-party aggregation (SMA)* to hide the participant's input, our *CryptoEdge-PPFL* still adopts a *similar* hybrid approach, namely, *differential noise reduction through SMA* [173], and the SMA approach is replaced by our proposed multiple data sources based *CryptoEdge-SA*.

***CryptoEdge-PPFL* Framework**. In *CryptoEdge-PPFL*, each $\mathcal{P}_i^{\text{iot}}$ in $\{\mathcal{P}_i^{\text{iot}}\}_{i\in[n]}$ uses the same approach as in [173] to independently add noise (denoted as $\mathcal{N}^{\mathcal{P}_i^{\text{iot}}}$) to the local ML model update $\mathcal{M}^{\mathcal{P}_i^{\text{iot}}}$ that is trained based on local dataset $\mathcal{D}^{\mathcal{P}_i^{\text{iot}}}$. Considering n IoT participants, the total aggregated noise adds up to $n \times \mathcal{N}^{\mathcal{P}_i^{\text{iot}}}$. With the adoption of *CryptoEdge-SA* technique, each $\mathcal{P}_i^{\text{iot}}$ can add a fraction of the noise $\frac{\mathcal{N}^{\mathcal{P}_i^{\text{iot}}}}{n}$, and then the total aggregated noise is still $\mathcal{N}^{\mathcal{P}_i^{\text{iot}}}$ without leaking each IoT device's private local model update even though the local privacy budget is divided by n. The effect of such an approach is proved in [173, 189].

Figure 5.3 illustrates the framework for privacy-preserving edge-based FL in our *CryptoEdge* platform.

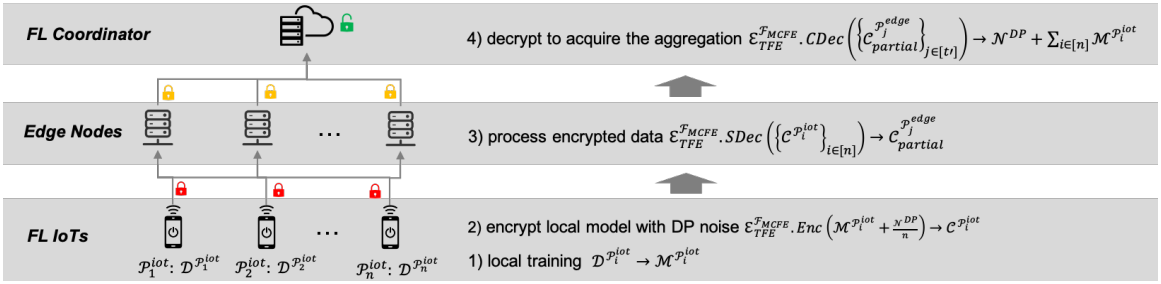


Figure 5.3: Illustration of privacy-preserving FL in *CryptoEdge*

Algorithm 7: Pseudocode of *CryptoEdge-PPFL*

Input: $\mathcal{L}$:= ML algorithm to be trained; m := minimum required aggregated replies; ϵ := privacy guarantee;

Output: Trained global model $\mathcal{M}$

function *cloud-aggregate*$(\mathcal{L}, \{\mathcal{P}_i^{iot}\}_{i\in[n]}, \{\mathcal{P}_j^{edge}\}_{j\in[s]}, m)$
  foreach $\mathcal{P}_i^{iot} \in \{\mathcal{P}_i^{iot}\}_{i\in[n]}$ **do** asynchronously query $\mathcal{P}_i^{\text{iot}}$ with $\text{msg}_q^{\mathcal{P}_i^{\text{iot}}} = (\mathcal{L}, m)$;
  randomly select $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[t']}$ from $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[s]}$;
  foreach $\mathcal{P}_j^{edge} \in \{\mathcal{P}_j^{edge}\}_{j\in[t']}$ **do** $\mathcal{S}_{\text{msg}_r}^{\mathcal{P}^{\text{edge}}} \leftarrow$ collect response $\text{msg}_r^{\mathcal{P}_j^{\text{edge}}}$;
  $\frac{1}{n}\sum_{i\in[n]}\mathcal{M}^{\mathcal{P}_i^{\text{iot}}} + \mathcal{N}^{\mathcal{P}^{\text{iot}}} \leftarrow \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCFE}}}.CD(\mathcal{S}_{\text{msg}_r}^{\mathcal{P}^{\text{edge}}})$;
  return $\mathcal{M} \leftarrow \frac{1}{n}\sum_{i\in[n]}\mathcal{M}^{\mathcal{P}_i^{\text{iot}}} + \mathcal{N}^{\mathcal{P}^{\text{iot}}}$
function *edge-process*$(\{\mathcal{P}_i^{iot}\}_{i\in[n]}, sk^{\mathcal{P}_j^{edge}}, m)$
  do $\mathcal{S}_{\text{msg}_r}^{\mathcal{P}^{\text{iot}}} \leftarrow$ collect response $\text{msg}_r^{\mathcal{P}_i^{\text{iot}}}$ **while** $|\mathcal{S}_{msg_r}^{\mathcal{P}^{iot}}| \geq m$ ***and*** *still in max waiting time*;
  $\text{msg}_r^{\mathcal{P}_j^{\text{edge}}} \leftarrow \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCFE}}}.SD_{sk^{\mathcal{P}_j^{\text{edge}}}}(\mathcal{S}_{\text{msg}_r}^{\mathcal{P}^{\text{iot}}})$;
  sends $\text{msg}_r^{\mathcal{P}_j^{\text{edge}}}$ to the cloud;
function *iot-train*$(msg_q^{\mathcal{P}_i^{iot}}, \mathcal{D}^{\mathcal{P}_i^{iot}}, pk^{\mathcal{P}_i^{iot}}, \epsilon)$
  $\mathcal{M}^{\mathcal{P}_i^{\text{iot}}} \leftarrow \mathcal{L}(\mathcal{D}^{\mathcal{P}_i^{\text{iot}}})$;
  $\mathcal{N}^{\mathcal{P}^{\text{iot}}} \leftarrow N^{\text{DP}}(\epsilon, \mathcal{M}^{\mathcal{P}_i^{\text{iot}}}, m)$;
  $\text{msg}_r^{\mathcal{P}_i^{\text{iot}}} \leftarrow \mathcal{E}_{\text{TFE}}^{\mathcal{F}_{\text{MCFE}}}.E_{pk^{\mathcal{P}_i^{\text{iot}}}}(\mathcal{M}^{\mathcal{P}_i^{iot}} + \frac{\mathcal{N}^{\mathcal{P}^{\text{iot}}}}{n})$;
  sends $\text{msg}_r^{\mathcal{P}_i^{\text{iot}}}$ to the edge nodes $\{\mathcal{P}_j^{\text{edge}}\}_{j\in[s]}$;

Unlike the traditional FL setting of two roles (i.e., participants and coordinator), we add an additional role (i.e., edge nodes) to help preprocess the encrypted model update. Note that we assume that each IoT device is able to train the local ML model independently as required in the FL setting. Those IoT devices could be monitors/sensors with Nvidia Jetson TX2i that has the basic computation capability for machine learning training.

Algorithm 7 presents the pseudocode of privacy-preserving FL at *CryptoEdge*. Here, each step for one-round of stochastic gradient descent (SGD) training in FL is presented as follows:

FL Cloud Coordinator - aggregating a global model. The cloud starts by querying all IoT devices with a specific ML algorithm that will be adopted and the minimum number of responses needed. Then, it

randomly selects a subgroup of edge nodes to help pre-process the encrypted local model and waits to collect the responses from the selected edge nodes. Finally, the cloud is able to recover the aggregated model by calling the *CombineDecrypt* algorithm of $\mathcal{E}_{\mathrm{TFE}}^{\mathcal{F}_{\mathrm{MCFE}}}$.

FL Edge Node - preprocessing encrypted model. Each edge node starts to collect the response from the IoT participants until the minimum number of responses is reached. Then, with the issued functional private key, it calls the *ShareDecrypt* algorithm of $\mathcal{E}_{\mathrm{TFE}}^{\mathcal{F}_{\mathrm{MCFE}}}$ to generate the partially decrypted data and send to the cloud.

FL IoT Participant - encrypting the trained local model. Each edge node starts with training the local model based on the local model with the specified ML algorithm (line 1). Then, it adds differential privacy noise on the trained model with input of privacy guarantee, a minimum number of participants (line 2). Finally, it does the encryption as same as in Section 5.2.2 and emits out the encrypted model to edge nodes.

Remark. Unlike the case of sum-aggregation in Section 5.2.2, it requires the average-aggregation of all generated models in the above-mentioned privacy-preserving FL application. Here, we omit the process of average that can be achieved by (i) either dividing each parameter in the aggregated model or (ii) issuing each edge node with the functional private key that is associated with the vector $(\frac{1}{m})_{j\in[t']}$ instead of vector $(1)_{j\in[t']}$ used in Section 5.2.2. Furthermore, by adjusting the functional private key associated vector, it is able to achieve privacy-preserving FL with trustworthiness on each IoT's local model.

5.3 Security and Privacy Analysis

5.3.1 Security Evaluation

Here, we use security proof methodology similar to that in [5], namely, simulation-based proof, to prove our claimed security guarantee. Then, we analyze other security aspects of the proposed TFE scheme.

To prove the security of *TFE*, we consider the following two cases: (i) $\mathcal{A}$ can break one player (i.e., a *sharing decryptor* or a *combining decryptor*); (ii) $\mathcal{A}$ can corrupt up to t-1 players, including two sub-cases: (ii.a) one *combining decryptor* with t-2 *sharing decryptors*; (ii.b) t-1 *sharing decryptors*. Note that the *sharing decryptor* and the *combining decryptor* denote the entities that run the *ShareDecrypt* and *CombineDecrypt* algorithms as illustrated in Definition 1. Then, we analyze the security from two aspects: encrypted data and functional result.

Security Proof. For the security of the encrypted data, we have the security claim as presented in Lemma 3 in Section 5.1.3. Specifically, under the DDH assumption, *TFE* scheme for $\mathcal{F}_{\mathrm{MCIP}}$ achieves selective simulation-based security (SEL-SIM-security). Below is the formal proof in detail.

Proof. For the case (i), the security risk is the same as the case of an ordinary multiple input functional encryption scheme. Hence, we adopt the same security definition and advantage of the adversary $\mathcal{A}$ as

illustrated in Appendix A.3, in the following formal proof.

To prove the *SEL-SIM-security* of TFE scheme $\mathcal{E}_{\text{TFE}}$ for $\mathcal{F}_{\text{MCIP}}$, we need to prove that for any adversary $\mathcal{A}$, $\text{Adv}^{\text{SEL-SIM}}_{\mathcal{A},\mathcal{E}_{\text{TFE}}} = 0$.

First, for the setup and encryption steps, we define the following simulator algorithms: $\mathit{Setup}^{SIM} = \mathit{Setup}^{ot}$, $\mathit{Encrypt}(msk^{SIM}) = \boldsymbol{u}_i$, and for the generation of functional private key we have the following simulator: $\mathit{KeyDerive}^{SIM}(msk^{SIM}, \boldsymbol{y}, aux) \rightarrow sk_{\boldsymbol{y}} = z$, where z is set as $\sum_{i\in[n]}\langle \boldsymbol{u}_i\boldsymbol{y}_i\rangle - aux \mod L$.

Then, we have the following fact for $\boldsymbol{u}$ and $\boldsymbol{u}-\boldsymbol{x}$: $\forall \boldsymbol{x}_i \in \{\boldsymbol{x}_i\}_{i\in[n]}$, the distributions $\{\boldsymbol{u}_i \mod L\}_{i\in[n]}$ and $\{(\boldsymbol{u}_i - \boldsymbol{x}_i) \mod L\}_{i\in[n]}$ are identical, where $\boldsymbol{x}_i \in \mathbb{Z}_L$ and $\boldsymbol{u}_i \leftarrow_{\$} \mathbb{Z}_L$. Note that symbol $\leftarrow_{\$}$ denotes that $\boldsymbol{u}_i$ is randomly sampled from $\mathbb{Z}_L$, and the independence of the $\boldsymbol{x}_i$ from $\boldsymbol{u}_i$ is only true in the selective security game.

Hence, we have the simulator to rewrite the experiment $\mathbf{REAL}^{\mathcal{E}^{ot}_{FE}}_{SEL}(1^\lambda, \mathcal{B})$ and oracle $\mathcal{O}_{H(\cdot)}$ as follows:

$\mathbf{REAL}^{\mathcal{E}^{ot}_{FE}}_{SEL}(1^\lambda, \mathcal{B})$

$\{x_i\}_{i\in[n]} \leftarrow \mathcal{B}(1^\lambda, \mathcal{F})$
$\forall i \in [n]$:
  $\boldsymbol{u}_i \leftarrow_{\$} \mathbb{Z}_L$
  $\boldsymbol{ct}_i \leftarrow \boldsymbol{u}_i \mod L$
$\alpha \leftarrow \mathcal{B}^{\mathcal{O}_{H(\cdot)}}(\{ct_i\}_{i\in[n]})$
Output : α

Oracle $\mathcal{O}_{H(\cdot)}$

$z \leftarrow \sum_{i\in[n]}\langle \boldsymbol{u}_i, \boldsymbol{y}_i\rangle - \langle \boldsymbol{x}_i, \boldsymbol{y}_i\rangle \mod L$
return : z

Therefore, the constructed $\mathbf{REAL}^{\mathcal{E}^{ot}_{FE}}_{SEL}(1^\lambda, \mathcal{B})$ is also identical to the experiment $\mathbf{IDEAL}^{\mathcal{E}^{ot}_{FE}}_{SEL}(1^\lambda, \mathcal{B})$ when executed with our simulator algorithms. We can observe that the constructed oracle $\mathcal{O}_{H(\cdot)}$ as corresponds to the oracle $\mathcal{O}(\cdot)$ (see Appendix A.3) that returns $\mathit{KeyGerive}^{SIM}(msk^{SIM}, \boldsymbol{y}, \{\langle \boldsymbol{x}_i, \boldsymbol{y}_i\rangle\}_{i\in[n]}$ for every queried $\boldsymbol{y}$. Thus, we can obtain that $\text{Adv}^{\text{SEL-SIM}}_{\mathcal{A},\mathcal{E}_{\text{TFE}}} = 0$. Hence, the adversary $\mathcal{A}$ does not have advantage to break the encrypted ciphertext.

For case (ii), the adversary $\mathcal{A}$ still has no advantage to breaking the encrypted ciphertext because $\mathcal{A}$ in the above-illustrated simulation game has been able to do as many queries as expected, that is, the increment on the corrupted players does not change the such a situation. □

Other Security of Functionality Result. Next, we analyze the security of the functionality result. The TFE scheme can ensure that the non-authorized player is not able to acquire the functionality result. Here, we do not consider the case (ii.a) because $\mathcal{A}$ is not assumed to break the authorized combine-decryptor.

For case (ii.b), suppose that $\mathcal{A}$ who corrupts t-1 players has non-negligible advantage ϵ to break the TFE to acquire the functionality result. In particular, the "master" functional private key is split into s shares via Shamir's secret share scheme [160] in the TFE scheme, so that we can construct a simulator to transfer $\mathcal{A}$'s advantage to solve the *t-of-s* Shamir's secret share scheme with t-1 shares. As proved in [160], there is no adversary who has a non-negligible advantage to solve that, and hence, $\mathcal{A}$ also does not have the non-negligible advantage to acquire the functionality result.

5.3.2 Privacy Analysis

The underlying cryptosystems is proved to be secure, indicating that the *untrusted* edge nodes and *honest-but-curious* cloud center are not able to reveal private information from the inputs of each IoT by breaking the TFE cryptosystem. As both the input and output of the edge layer are still in ciphertext forms in the *CryptoEdge* platform, there is no chance for the *untrusted* edge nodes to launch potential inference attacks. Thus, we only analyze a few inference attempts launched by an *honest-but-curious* adversary (i.e., cloud center) who is able to acquire the functionality results that are in plaintext.

Inference Attack against Secure Aggregation. There is one possible inference attempt in the single data source secure aggregation approach, namely, the *honest-but-curious* adversary may subtract the adjacent aggregation results. For instance, suppose that the stream data in the sliding window $S^{(T)}$ is denoted as $\mathcal{D}^{\mathcal{P}^{\text{iot}}}_{S^T} = \{d^{\mathcal{P}^{\text{iot}}}_{T-m}, ..., d^{\mathcal{P}^{\text{iot}}}_{T+m}\}$. Then, for the sliding window S^{T+1} would be $\mathcal{D}^{\mathcal{P}^{\text{iot}}}_{S^{T+1}} = \{d^{\mathcal{P}^{\text{iot}}}_{T-m+1}, ..., d^{\mathcal{P}^{\text{iot}}}_{T+m+1}\}$. By subtracting the aggregation results of adjacent sliding window S^T and S^{T+1}, the adversary may acquire $d^{\mathcal{P}^{\text{iot}}}_{T-m} + d^{\mathcal{P}^{\text{iot}}}_{T+m+1} = \sum_{\mathcal{D}^{\mathcal{P}^{\text{iot}}}_{S^{T+1}}} - \sum_{\mathcal{D}^{\mathcal{P}^{\text{iot}}}_{S^T}}$. Obviously, in the case of non-weighted secure aggregation, the adversary can only infer the average of $d^{\mathcal{P}^{\text{iot}}}_{T-m}$ and $d^{\mathcal{P}^{\text{iot}}}_{T+m+1}$ instead of the individual data, and hence, it does not violate our privacy guarantee. Furthermore, another inference attempt is that the adversary tries to narrow the sliding window S^{T+1} by one so that it can infer the $d^{\mathcal{P}^{\text{iot}}}_{T-m}$. However, the adjustment of the size of the sliding window needs the cooperation of the IoT, and hence, such an inference attack would be noticed by the IoT. Note that such an inference attempt can only occur at the situation of single data source secure aggregation.

Inference Attack against *CyptoEdge-PPFL*. As illustrated in [189], we consider another inference attempt launched by a *curious* adversary who may use exploited weight $\boldsymbol{w}_{\text{exploit}} = (1, 0, ..., 0)$ to assign to the edge nodes so that each edge node can be issued with function private key from the TPA. Except for the first element, the rest of the elements in $\boldsymbol{w}_{\text{exploit}}$ are all zero, and hence the curious adversary is able to acquire the first participant's model update.

To prevent such an attack, similar to a solution presented in [189], we also add *inference weights filter (IWF)* into the *TPA*. In general, the IWF module will check the vector $\boldsymbol{w} = (1, 0, ..., 0)$ to ensure that the number of non-zero elements is greater than a threshold τ, basically, $\tau \geq 2$. As a result, it is impossible to launch the above inference attack. For more details about the IWF module, we refer the reader to [189].

5.4 Experimental Evaluation

We evaluate *CryptoEdge* platform along the following lines. (i) To evaluate the performance of the basic secure aggregation, we consider ifferent edge computing settings such as the number of enrolled edge nodes, the number of IoTs, the size of the sliding window, etc. (ii) We compare the performance of our *CryptoEdge-SA* to the existing threshold Paillier based secure aggregation. (iii) We also evaluate the performance of the

proposed *CryptoEdge-PPFL* framework with different baselines: *TP-PPFL*, where the secure aggregation adopts the threshold Paillier scheme; *HybridAlpha(HA)-PPFL* - the *HybridAlpha* framework [189] - a PPFL framework in two-layer cloud scenario.

5.4.1 Experimental Setup

Implementation Consideration. We have implemented the *CryptoEdge* platform in *Python* programming language. Specifically, the underlying cryptosystems, namely, the threshold functional encryption scheme and the threshold Pailier system, are also implemented in Python based on the *gmpy2* library, which is a C-coded Python extension module that supports multiple-precision arithmetic and relies on the GNU multiple precision arithmetic (GMP) library. We also employ similar decryption acceleration techniques as adopted in [189].

The underlying ML model to be trained in the *CryptoEdge-PPFL* framework is a convolutional neural network (CNN) as the one used in [189] that is implemented using Keras with a Tensorflow as the back-end. Specifically, CNN has two internal layers of ReLU units and a softmax layer of ten classes with cross-entropy loss. The first layer contains 60 neurons and the second layer contains 1000 neurons. The total number of parameters of this CNN is 118,110. We use the same hyperparameters as reported in [189], i.e., a learning rate of 0.1, a batch rate of 0.01. and for differential privacy we use a norm clipping of 4.0, and epsilon of 0.5.
Dataset. To evaluate the performance of the secure aggregation approach, we use the randomly generated numbers for each IoT device to simulate the data in the sliding window. For the evaluation of the *CryptoEdge-PPFL* framework, we use the CNN to classify the publicly available MNIST dataset of handwritten digits [106]. We also equally split the MNIST training samples and assign them to each IoT device in the training phase of *CryptoEdge-PPFL*.
Experimental Environment. All experiments have been performed on the following test platforms: *Platform I* - a local Macbook Pro with 2.5GHz quad-core Intel Core i7 CPU and 16GB RAM - is used for the evaluation of various secure aggregation approaches; *Platform II* that is a remote cloud service to simulate the edge computing scenario including three parties: (i) one AWS *c5.18xlarge* instance with 72 vCPUs and 144GB RAM to play the role of the cloud center; (ii) one AWS *m5d.8xlarge* instance with 32 vCPUs and 128GB RAM to play the role of the edge nodes; (iii) one AWS *r5a.4xlarge* instance with 2 vCPUs and 64GB RAM to play the role of the IoT devices. Note that multiple edge nodes and multiple IoT devices are simulated at each machine platform.

5.4.2 Experimental Results

Evaluation of Secure Aggregation. As shown in Figure 5.4, we report the performance (i.e., the elaspsed time cost in millisecond) of our proposed *CryptoEdge(CE)-SA* and also compare its results with that of *Threshold Paillier(TP)-SA*. Overall, we evaluate all SA mechanisms in different edge computing settings:

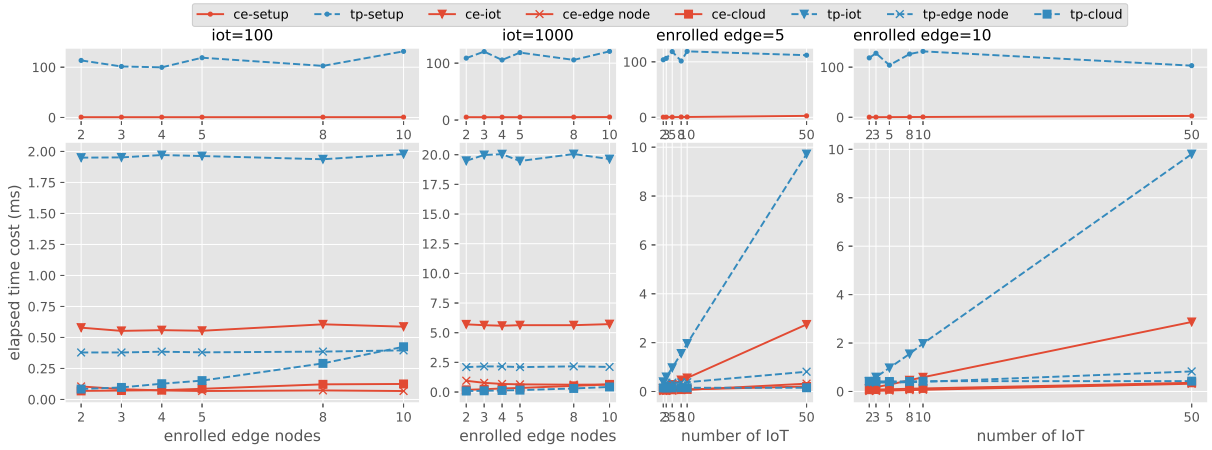


Figure 5.4: Performance comparison between weighted *CryptoEdge(CE)-SA* and *Threshold Paillier(TP)-SA* in different settings: various number of IoTs and various number of enrolled edge nodes.

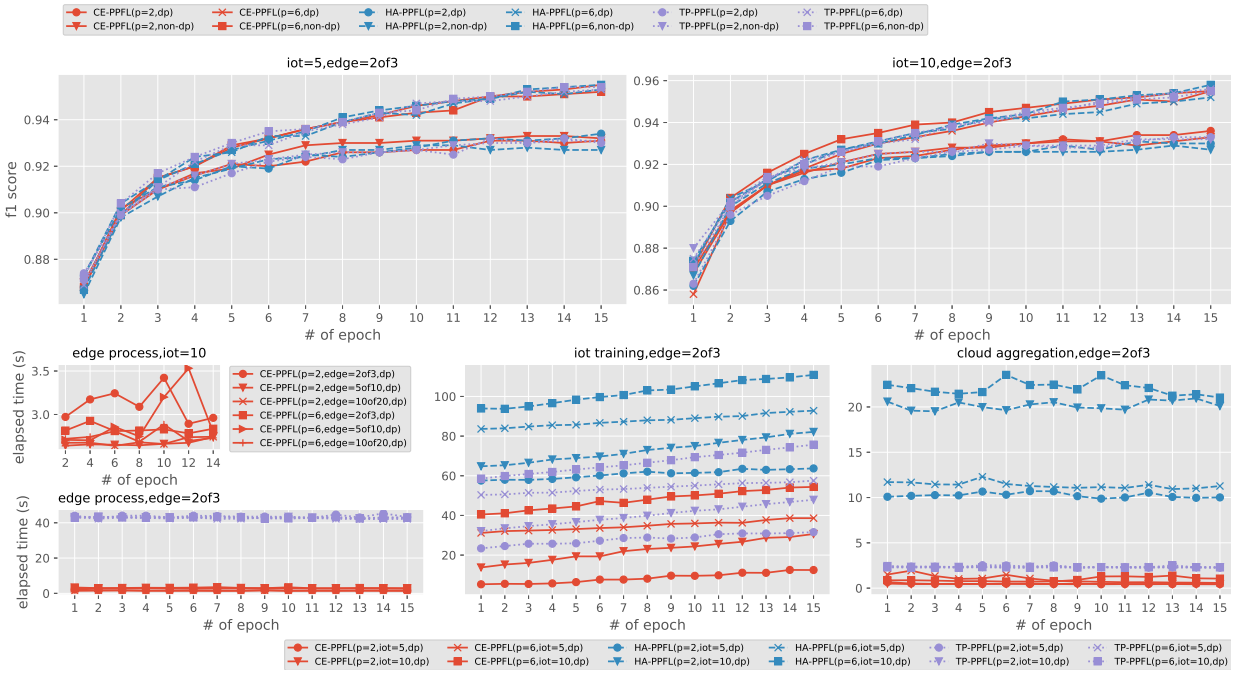


Figure 5.5: Performance comparison among our *CryptoEdge(CE)-PPFL*, *Threshold Paillier(TP)-PPFL* and *HybridAlpha(HA)-PPFL* in different settings.

(i) we fix the number of IoTs (i.e., 100 and 1000), and then we evaluate the performance by increasing the number of enrolled edge nodes; (ii) we fix the number of enrolled edge nodes (i.e., 5 and 10), and then increase the number of IoTs to evaluate performance. In the upper part of the figure, our CE-SA has lesser setup time (i.e., from 0.5 ms to 5 ms in our experiment), especially, compared to the TP-SA approach that costs more than 100ms. Regarding IoT performance, our CE-SA approach reduces 70% processing time than the TP-SA approach. Besides, we observe that the processing time of the IoT device is linear to the number of IoT devices and is not impacted by the number of enrolled edge nodes. Also, a similar conclusion is also applied for the processing time of the edge nodes. Furthermore, the processing time of cloud is only impacted by the number of edge nodes, and our CE-SA still has better performance than the TP-SA.

Evaluation of *CryptoEdge-FL*. As reported in Figure 5.5, we compare the performance of our framework (*CryptoEdge(CE)-PPFL*) with that of two baselines - *HybridAlpha(HA)-PPFL* and *Threshold Paillier(TP)-PPFL* - in the different settings such as encoding precision (p), number of edge nodes, number of IoT devices, and differential privacy (dp) settings. With regards to model accuracy (i.e., F1 score), our *CE-PPFL* can achieve accuracy that is comparable to *HA-PPFL* in traditional two-tier architecture and *TP-PPFL* in three-tier architecture. However, compared to *HA-PPFL*, *CE-PPFL* can reduce a participating IoT device's local processing time by 53% on average, and the cloud (i.e., the coordinator) processing time by 95% on average. Furthermore, compared to *TP-PPFL*, our *CE-PPFL* can reduce the IoT (i.e., the participant) local processing time 28% on average, the edge processing time 93% in average, and the cloud (i.e., the coordinator) processing time 57% in average. Besides, we also observe that the higher encoding precision setting will result in higher model accuracy, and the edge process time is a positive correlation to the number of participating IoT devices.

5.5 Summary and Discussion

To tackle the challenge of potential privacy leakage in the aggregation computation in three-tier edge computing architecture, we propose *CryptoEdge* to support practical and efficient secure aggregation over the encrypted data in edge computing environments. *CryptoEdge* includes our proposed *threshold functional encryption (TFE)* scheme, our *secure aggregation* approaches, and a privacy-preserving edge-based FL framework. Besides, we have presented formal security and privacy analysis, as well as an experimental evaluation considering various settings. Our evaluation results show that *CryptoEdge* achieves the security and privacy guarantee and provides significant performance improvements.

6.0 Transparent and Trustworthy Secure Computation Infrastructure using Blockchain

The provisioning of *openness* and *accountability*, also referred to as *transparency*, in recently proposed works such as in [104, 152, 102, 122, 58, 33, 64] is an approach to increase users' trust or confidence on infrastructure service providers, especially, the cryptographic infrastructure that supports security related services and is commonly assumed to be fully trusted. For instance, the certificate authorities (CAs), as the underlying public key infrastructure for SSL/TLS protocol, are responsible for issuing digital certificates that certify the ownership of a public key by the named principal of the certificate and allows others to rely upon signatures made by the private key corresponding to the certified public key. To address the various attacks [17, 194, 26] and mis-issuance problems [101, 158, 37] in the certificate issuing procedure, notions of *certificate transparency* [102, 104] and *transparency overlay* [33] have been proposed. Transparency overlay is actually a formal study of several specific certificate transparency frameworks.

Similar to *certificate authority*, we have proposed the notion of *authority transparency* in [190] to address similar but more complex issues related to a TPA that is critical component of many emerging cryptosystems such as FE schemes adopted in this dissertation and our proposed TFE scheme. Unlike the CAs that only need to issue a certificate that proves the *identity-to-public-key* binding, the TPA is responsible for more complex authority tasks such as setting public parameters and providing private key service for authorized entities according to various credentials such as attribute identities and functionality-related vectors. In particular, compared to *certificate transparency*, *authority transparency* can further capture multiple rounds of interactions between the TPA and other entities.

Our initial *authority transparency* [190] work presents a formal architecture to make the TPAs transparent and trustworthy. There are still limitations that hinder its deployment and application in the privacy-preserving machine learning (PPML) domains: (i) the definitions and protocols designed in [190] only work on, as well as relies on, the attribute-based encryption schemes; (ii) the implementation of *authority transparency* is based on a secure logging system. As a result, existing *authority transparency* proposal may not nicely support other emerging cryptosystems such as functional encryption (FE) family that has been used for secure computation in our proposed PPML systems. Except for the *identity-to-public-key-binding stealthy targeted attack* and the *private-key-service censorship attack* as illustrated in [190], FE-based PPML systems have additional privacy leakage issues, for instance, the inference attack by manipulating a malicious vector as illustrated in [189, 191]. Furthermore, the deployment of distributed secure logging system based authority transparency may not be widely accepted by the Internet community because (i) it requires several commercial companies or non-profit organization who have the computation and storage capability to deploy a secure logging system such as that occurs in the certificate transparency community (e.g., those secure logging system deployed by Google and Mozilla); (ii) there is also a lack of a concrete mechanism for the entities to participate in a transparency framework, and monitor and audit unintentional or malicious

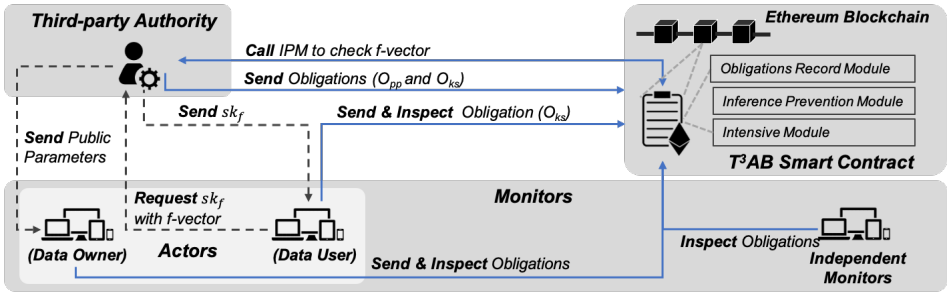


Figure 6.1: Overview of the T^3AB transparency framework.

behaviors.

To address the above-mentioned limitations, in this chapter, building up on our initial notion of *authority transparency* framework in [190], we propose T^3AB framework to provide **T**ransparency and **T**rustworthiness of **T**hird-party **A**uthority and related entities using the **B**lockchain techniques. These entities (e.g., honest-but-curious cloud server, third-party IaaS, and coordinator) and TPAs form the foundation of the secure infrastructure for our proposed PPML approaches presented in Chapter 3, Chapter 4 and Chapter 5. In general, to achieve transparency and trustworthy goals, T^3AB employs the Ethereum blockchain as the underlying public ledger infrastructure, and also uses a well-designed novel Ethereum smart contract to support automatic accountability with additional incentive mechanisms that motivates participants' auditing behavior and punishes the misbehaviors or malicious behaviors.

6.1 Transparency Framework

6.1.1 Overview of the Framework

To tackle trust issues caused by a third-party authority and curious entities in FE-based applications, here, we present our proposed T^3AB framework that increases the transparency and trustworthiness of the entities using Ethereum blockchain.

Entities. Figure 6.1 illustrates the T^3AB framework. Note that the dashed lines represent the procedures of FE-based applications, while the solid lines denote the procedures of the T^3AB framework. T^3AB consists of the following entities:

- *TPA*. The TPA is the same role as in the ordinary functional encryption cryptosystem, but with additional responsibilities to fulfill in our T^3AB, including (a) submitting the public parameters obligations (in particular, identity-to-public-key bindings), (b) reporting its fulfillment of obligations in the key

service process, and (c) verifying that the submitted/reported obligations are permanently recorded in blockchain.

- *Actors.* Actors include all users of an FE-based PPML system, namely, the entities (e.g., *participants*) that employ the encryption algorithm and the entities (e.g., *coordinator*) that employ the decryption algorithm. Besides, the actors may also need to fulfill the obligations of key service because it involves interaction between the actors and the TPA.
- *Monitors.* Monitors are responsible for inspecting the contents of the recorded auditing obligations to find suspicious obligations. In our T^3AB, the encryption entities or the additional independent entities play the role of the monitors.
- *Administrator.* An administrator is responsible for the deployment, maintenance, administration of the smart contract. The smart contract mainly includes three modules: (a) the obligation record module that provides various interaction functions for the entities to achieve recording, auditing and inspection requirements for the obligations, (b) the incentive mechanism that provides the payment and reward functions to the participants, and (c) the *inference prevention module (IPM)*, previously deployed in a TPA as illustrated in [189]. Note that the Ethereum blockchain can ensure the trustworthiness of smart contracts; it can also ensure that the recorded obligations are distributed, open, and against any malicious tampering and normal revision. Furthermore, once deployed it does not need a centralized administration.

Notations and Statement. To elaborate our T^3AB, we first present the notations, entities, and scenarios of applying our T^3AB framework in an FE-based environment. Suppose that we have a group of data owners $\{\mathcal{C}_i^{owner}\}_{i\in[n]}$ that will share their private data $\boldsymbol{x} = \{x_i\}_{i\in[n]}$ encrypted by an FE scheme where for simplicity we assume that $\mathcal{C}_i^{owner}$ owns data x_i , a group of data users $\{\mathcal{C}_j^{user}\}_{j\in[m]}$, where each data user has a vector $\boldsymbol{y}_j$ and needs to acquire the inner-product $\langle \boldsymbol{x}, \boldsymbol{y}_j \rangle$ over the ciphertext of $\boldsymbol{x}$, and a TPA $\mathcal{A}$ that provides public and private key services for these data owners and users. Furthermore, let $\{\mathcal{C}_k^{owner}\}_{k\in[l]}$ be the monitors. We use $\mathcal{B}$ representing the Ethereum blockchain, and let $\mathcal{B}_{SC}^{T^3AB}$ denote our proposed smart contract deployed in the blockchain.

6.1.2 Threat Model

Existing FE-based applications are usually based on the assumption that the TPA is assumed to be *fully trusted* and the coordinator (a.k.a., the decryption party) is assumed to be *honest-but-curious*. Hence, the threat models in such cases typically focus on an adversary who attempts to compromise the encrypted data and a*curious* entity that launches potential privacy attacks (e.g., infer the private information), while *honestly* following the protocol/algorithm.

Unlike such a threat model, our T^3AB focuses on increasing entities' trust on the TPA and the coordinator via the transparency approach. In particular, T^3AB tries to reduce the dependence of FE-based applications

on the assumptions of a *trusted* TPA and an *honest* coordinator. As the *identity-to-public-key-binding stealthy targeted attack* and the *private-key-service censorship attack* illustrated in [190], the TPA may not be trusted because of its unintentional misbehaviors or malicious behaviors. Similarly, the coordinator may also behave dishonestly. We assume that such a dishonest adversary may pretend to behave honestly without being detected by other entities. Adversaries may not follow the specification in the protocols, and/or attempt to conceal their activities. In general, the *dishonest* adversary includes the TPA and actors, where a dishonest TPA may attempt to forge a key service *proof-of-work* without actually providing a valid key service; and, a dishonest actor may try to incorrectly blame other entities for misbehavior. Note that misbehavior may be related to non-malicious misuse by normal actors or the behavior of compromised actors controlled by an attacker.

We note that unlike the secure logging system based *authority transparency* framework in [190], where the *logger* is treated as a potential dishonest adversary, in our T^3AB the Ethereum smart contract is adopted as the public ledger infrastructure that has been proved to be a trusted computation platform.

6.1.3 Framework Details

T^3AB Model. Unlike the authority transparency model in [190] that builds on the secure logging system for attribute-based encryption (ABE) cryptosystem, T^3AB uses the Ethereum blockchain, and to keep consistency, we adopt the similar concepts/notions of the authority transparency model but it considers different scenarios including FE-based applications and blockchain-based public ledger infrastructure.

Unlike ABE-based applications, in FE-based applications, there is no need to define complex attribute identities in the functional encryption scheme, and hence, we update the related concepts with consideration of the simplified identities and smart contracts.

Suppose that each entity e in T^3AB is issued or self-generates an identity-based public and private key pair $\langle \mathrm{pk}_e, \mathrm{sk}_e \rangle$. Besides, the key service occurs between entity $\mathcal{C}^{\mathrm{actor}}$ and authority $\mathcal{A}$, where each entity has already owned its public and private key pair. For instance, let $\langle \mathrm{pk}_{\mathrm{actor}}, \mathrm{sk}_{\mathrm{actor}} \rangle$ and $\langle \mathrm{pk}_{\mathrm{TPA}}, \mathrm{sk}_{\mathrm{TPA}} \rangle$ represent the public/private key pairs of the actor and the TPA, respectively. Here, we first present the notion of *public parameter audit obligation* and *key service audit obligation*, and then the formal definition of T^3AB model.

Definition 2 (Public Parameter Audit Obligation (PPAO))**.** *A PPAO $\mathcal{O}_{pp}$ of e is a map structure as follows:*

$$\mathcal{O}^{e}_{pp} := H(e_{id}) : \langle e_{id}, pk_e, Sig_{sk_e}(e_{id}, pk_e) \rangle, \tag{6.1}$$

where e_{id} represents the descriptive identifier of e, $H(\cdot)$ is the hash function, pk_e denotes the public key binding of entity e, and Sig_{sk_e} is the signature using sk_e.

Definition 3 (Key Service Audit Obligation (KSAO))**.** *A KSAO $\mathcal{O}_{ks}^{\mathcal{C}^{actor},\mathcal{A}}$ is a map structure consisting of*

a pair of key service snapshots

$$\mathcal{O}_{ks}^{\mathcal{C}^{actor},\mathcal{A}} := H(\mathcal{C}_{id}^{actor}, \mathcal{A}_{id}, r) : \langle \mathcal{S}_{req}, \mathcal{S}_{resp} \rangle, \tag{6.2}$$

where each snapshot is a 4-tuple as follows:

$$\mathcal{S}_{req} := H(\mathcal{C}_{id}^{actor}, \mathcal{A}_{id}, r) : \langle r, f, t_{\mathcal{C}^{actor}}, Sig_{sk_{actor}}(r, f, t_{\mathcal{C}^{actor}}) \rangle, \tag{6.3}$$

$$\mathcal{S}_{resp} := H(\mathcal{C}_{id}^{actor}, \mathcal{A}_{id}, r) : \langle r, Sigma, t_{\mathcal{A}}, Sig_{sk_{TPA}}(r, Sigma, t_{\mathcal{A}}) \rangle, \tag{6.4}$$

such that

$$\mathcal{S}_{req}.H(\mathcal{C}_{id}^{actor}, \mathcal{A}_{id}, r) = \mathcal{S}_{resp}.H(\mathcal{C}_{id}^{actor}, \mathcal{A}_{id}, r) \tag{6.5}$$

$$\mathcal{S}_{resp}.t_{\mathcal{A}} - \mathcal{S}_{req}.t_{\mathcal{C}^{actor}} > 0 \tag{6.6}$$

$$\mathcal{S}_{req}.t_{\mathcal{A}} - \mathcal{S}_{resp}.t_{\mathcal{C}^{actor}} < \delta_t \tag{6.7}$$

where r is the nonce selected by the key service requester, t is the timestamp of key service processed by each entity, f denotes the request content such as function related vector, Sigma represents the proof-of-work that TPA has issued the key, δ_t is the threshold of timestamp difference indicating the expected time of processing of the key service request by the TPA.

Remark. In particular, the $\mathcal{O}_{pp}$ is an *identity-to-public-key* binding with the issuer's signature, while $\mathcal{O}_{ks}^{\mathcal{C}^{\text{actor}},\mathcal{A}}$ is the *proof-of-key-service*. In the $\mathcal{O}_{ks}^{\mathcal{C}^{\text{actor}},\mathcal{A}}$, for simplicity, to provide the *proof-of-work* of issuing the functional private key sk_f for the function related materials f, let Sigma be $\text{H}(\text{sk}_f)$.

Based on the notion of *public parameter audit obligation* and *key service audit obligation*, we present the formal T^3AB model as follows:

Definition 4 (T^3AB Model)**.** *Let $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ denote the third party authority, blockchain, and actor, respectively, which are parties involved in the interactive protocols. Let $\mathcal{C}.actor$ and $\mathcal{C}.monitor$ represent the roles of the actor and monitor that execute the functional and monitoring modules, respectively. We define T^3AB model $\mathcal{M}$ as a set of five interactive protocols:*

$$\mathcal{M}_{\mathcal{O}}^{\mathcal{A},\mathcal{B},\mathcal{C}} = (Gen_{\mathcal{O}}, Log_{\mathcal{O}_{pp}}, Log_{\mathcal{O}_{ks}}, Inspect), \tag{6.8}$$

and each protocol is defined as follows:

$$(S_{\mathcal{O}_{pp}}, S_{\mathcal{O}_{ks}}) \leftarrow Run(1^{\lambda}, Gen_{\mathcal{O}}, \{\mathcal{A}, \mathcal{C}.actor\}) \tag{6.9}$$

$$(b_{\mathcal{A}}, \varepsilon) \leftarrow Run(1^{\lambda}, Log_{\mathcal{O}_{pp}}, \{\mathcal{A}, \mathcal{B}\}, (S_{\mathcal{O}_{pp}}, \varepsilon)) \tag{6.10}$$

$$(b_{\mathcal{A}}, b_{\mathcal{C}}, \varepsilon) \leftarrow Run(1^{\lambda}, Log_{\mathcal{O}_{ks}}, \{\mathcal{A}, \mathcal{C}.actor, \mathcal{B}\}, (\mathcal{O}_{ks}.\mathcal{S}_{\mathcal{A}}, \mathcal{O}_{ks}.\mathcal{S}_{\mathcal{C}}, \varepsilon)) \tag{6.11}$$

$$(b_{\mathcal{B}}, \varepsilon) \leftarrow Run(1^{\lambda}, Inspect, \{\mathcal{B}, \mathcal{C}.monitor\}, (\varepsilon, \varepsilon)) \tag{6.12}$$

Lemma 4 presents the security guarantee as follows, which is proved later.

Lemma 4. *If the hash function is collision-resistant and the signature scheme is unforgeable, then T^3AB model comprises a secure transparency framework.*

Remark. Note that the formal definition of our T^3AB model is inherited from the *authority transparency* model [190] with needed changes considering the underlying Ethereum blockchain infrastructure. Specifically, in the authority transparency model, the *gossip* protocol essentially ensures the consistency of distributed logs without being tampered by an adversary, while the *check* protocol guarantees that the submitted obligations are recorded by the logging system. As T^3AB adopts the Ethereum blockchain as the underlying public ledger infrastructure, there is no need to run the *gossip* and *check* protocols because these logging-related functions are the implicit feature in the Ethereum smart contract.

Design of $\mathcal{B}_{SC}^{T^3AB}$. The T^3AB smart contract is a critical component in our framework. To support the goal of T^3AB framework, $\mathcal{B}_{SC}^{T^3AB}$ includes various types of modules: *administrative module*, *access control module*, *obligation module*, *inspection module*, and *incentive module*. Each module is presented as follows:

Administrative module allows the role of administrator to deploy the smart contract into the Ethereum network. The module also includes functions such as opening and locking the enrollment, allowing the participants to drop out.

Access control module supports a basic role based access control mechanism that allows the account (a.k.a, the participating entities) have permission to call role-related functions. In $\mathcal{B}_{SC}^{T^3AB}$, we define four types of roles: the *TPA*, the *actors of data owner*, the *actors of data user*, the *monitors* and the *administrator* (i.e., the smart contract owner). Obviously, the administrative entity who deploys the smart contract becomes the smart contract owner. The ownership can be transferred to a new account if necessary. Besides, it is also possible to relinquish this administrative privilege, which is a common pattern after an initial stage when there is a decentralized administration requirement. After the deployment, each entity needs to register its role by calling the corresponding function before they can use the ordinary features of the smart contract.

Obligation module works on recording the *audit obligation* into the public ledger. Regarding the entity registration, it also publishes its *identity-to-public-key* binding to the Ethereum blockchain, as illustrated above. Note that the identity of the entity is the unique public address (i.e., 42 hex string characters without case-sensitivity) of the blockchain account, which is derived from the entity's private key. With regards to the key service audit obligation procedure, the key service requestor (i.e., data owner) can call the corresponding function that includes role verification with a randomly generated request identifier, the key-related request parameters, and the corresponding signature. The function then automatically analyzes the key-related request parameters via executing the *Inference Prevention Module (IPM)*. Note that $\mathcal{B}_{SC}^{T^3AB}$ has already integrated the IPM module that is previously integrated in the TPA entity as illustrated in Chapter 3, Chapter 4, and Chapter 5.2. Upon receiving the key service request with the request identifier, the TPA first checks the verification result of IPM. If the request passes the verification, the TPA will issue the functional private key and then publish a response snapshot to fulfill the key service obligation.

Inspection module mainly inspects the completeness of a pair of the key service snapshot to check whether the

TPA's fulfill its key service obligation or not. Besides, it also allows to check the published *identity-to-public-key* binding. Beside the inspection module that can prevent potential misbehaviors, we have introduced the access control mechanism to prevent partial misbehaviors and malicious behaviors as each entity only will be allowed to call corresponding functions with limited privilege.

Incentive module in the $\mathcal{B}_{SC}^{T^3AB}$ includes several functions to achieve the incentive mechanism, as depicted in Figure 6.3. The incentive mechanism is based on payment features of the Ethereum network, where the token can be exchanged to real concurrency. As illustrated in Figure 6.3, we design several functions as 'public payable', which indicates that the smart contract is able to receive the transaction value (e.g., the Ether) when the function is successfully called and executed.

In general, m data users need to equally pay for the cost of calling registration function for themselves as well as n data owners and the TPA. Each data user also needs to pay for the cost of calling *request obligation record* function and also the cost of calling *response obligation record* function by the TPA. Additionally, there exists a mechanism to punish the misbehaviors and malicious activities by a *dishonest* TPA and data users. To achieve that, the data users and the TPA first need to register and pay the cost by themselves. The data owners equally make a deposit for all the entities' registration cost after the enrollment phase. Then, the data users and the TPA can call the *disposable* reward function to withdraw the registration cost. Besides, we make the TPA and data owners make a *guarantee deposit* after the registration phase. The monitors can register and pay the cost by themselves, and then calls the inspection function to check the suspicious behaviors. If monitors find the malicious behaviors, they will acquire the reward from the fine to the corresponding entity (i.e., the guarantee deposit of the entity). Without the guarantee deposit, the corresponding entity is not allowed to operate in the system. Additionally, we discuss the quantitative analysis of the cost of each entity in $\mathcal{B}_{SC}^{T^3AB}$ in Section 6.3.

T^3AB **Procedures**. As depicted in Figure 6.2, we illustrate the four phases of the T^3AB framework with specific procedures in a typical FE-based application scenario. Note that the dashed arrows represent the functional procedures of a typical FE-based application, while the solid arrows denote procedures of the T^3AB framework. In our design, each entity in the FE-based application can also play the role of the auditor and monitor in the T^3AB framework, and we also allow additional monitors to help inspect the misbehaviors and malicious behaviors.

Here, we present the specific procedures of each phase in the T^3AB framework as follows:

Phase I: entity initialization: For each entity e with role e_{role} and identifier e_{id} in the framework, it generates a public and private key pair $\langle \mathrm{pk}_e, \mathrm{sk}_e \rangle$. Then, entity e registers its role e_{role} to $\mathcal{B}_{SC}^{T^3AB}$, and publish its *id-to-public-key binding* $\langle e_{id}, \mathrm{pk}_e \rangle$ with its signature $\mathrm{Sig}_{\mathrm{sk}_e}(e_{id}, \mathrm{pk}_e)$ to $\mathcal{B}_{SC}^{T^3AB}$.

Phase II: FE initialization. The TPA $\mathcal{A}$ setups the FE cryptosystem with the master public key and master private key pair $\langle \mathrm{mpk}^{\mathrm{FE}}, \mathrm{msk}^{\mathrm{FE}} \rangle$. Using the master keys, the TPA generates and sends the common public key $\mathrm{pk}_{com}^{\mathrm{FE}}$ for all entities (i.e., data owners and data users) in the FE-based application. Then, the TPA publishes the binding $\langle \mathcal{A}_{id}, \mathrm{pk}_{com}^{\mathrm{FE}} \rangle$ with its signature $\mathrm{Sig}_{\mathrm{sk}_{\mathcal{A}}}(\mathcal{A}_{id}, \mathrm{pk}_{com}^{\mathrm{FE}})$ to $\mathcal{B}_{SC}^{T^3AB}$.

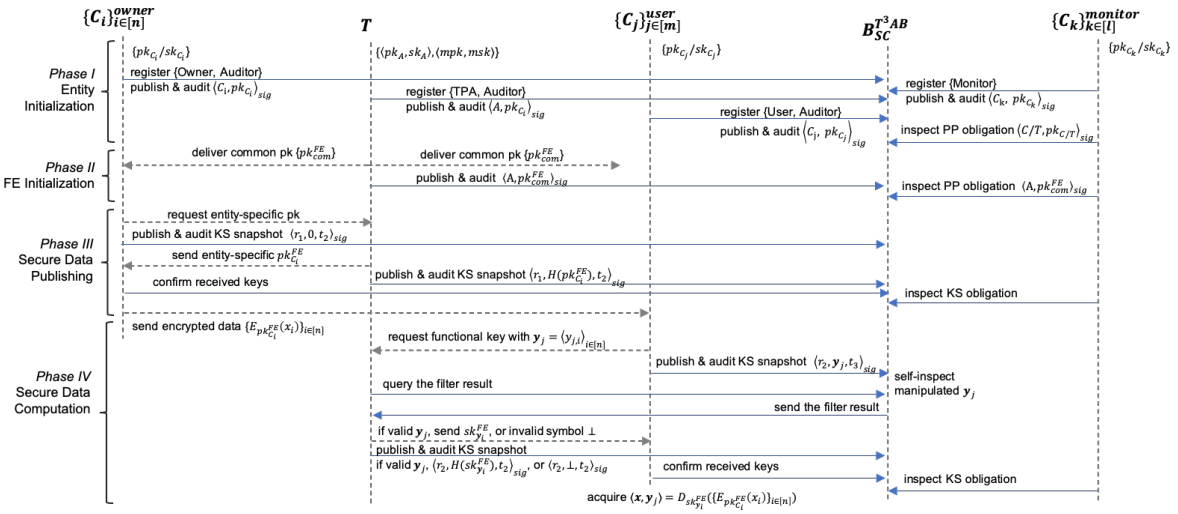


Figure 6.2: Illustration of the four phases of T^3AB framework in a FE-based application scenario

Phase III: secure data publishing. For each data owner $\mathcal{C}_i^{\text{owner}}$, it first selects a nonce r as the key service identifier. Then $\mathcal{C}_i^{\text{owner}}$ requests the entity-specific public key $\text{pk}^{\text{FE}}_{\mathcal{C}_i^{\text{owner}}}$ from the TPA with r. Meanwhile, $\mathcal{C}_i^{\text{owner}}$ also sends a request key service snapshot $\mathcal{S}^{\mathcal{C}_i^{\text{owner}}}_{\text{req}}$ to $\mathcal{B}^{T^3AB}_{SC}$ as follows:

$$\mathcal{S}^{\mathcal{C}_i^{\text{owner}}}_{\text{req}} = \langle r, 0, t_{\mathcal{C}_i^{\text{owner}}}, \text{Sig}_{\text{sk}_{\mathcal{C}_i^{\text{owner}}}}(r, 0, t_{\mathcal{C}_i^{\text{owner}}})\rangle. \tag{6.13}$$

Then, the TPA generates $\text{pk}^{\text{FE}}_{\mathcal{C}_i^{\text{owner}}}$ for $\mathcal{C}_i^{\text{owner}}$ using its master keys, and also publishes a corresponding response key service snapshot $\mathcal{S}^{\mathcal{A}}_{resp}$ to $\mathcal{B}^{T^3AB}_{SC}$ to fulfill its key service audit obligation $\mathcal{O}^{\mathcal{C}_i^{\text{owner}},\mathcal{A}}_{ks}$ with mapping key $\text{H}(\mathcal{C}^{\text{owner}}_{i,id}, \mathcal{A}_{id}, r)$ as follows:

$$\mathcal{S}^{\mathcal{A}}_{\text{resp}} = \langle r, \text{H}(\text{pk}^{\text{FE}}_{\mathcal{C}_i^{\text{owner}}}), t_{\mathcal{A}}, \text{Sig}_{\text{sk}_{\mathcal{A}}}(r, \text{H}(\text{pk}^{\text{FE}}_{\mathcal{C}_i^{\text{owner}}}), t_{\mathcal{A}})\rangle. \tag{6.14}$$

Each data owner then uses $\text{pk}^{\text{FE}}_{e_i^{\text{owner}}}$ to encrypt its data as $\{\text{Enc}_{\text{pk}^{\text{FE}}_{e_i^{\text{owner}}}}(x_i)\}_{i\in[n]}$. Finally, the data owner publishes a receipt for the received $\text{pk}^{\text{FE}}_{e_i^{\text{owner}}}$.

Phase IV: secure data computation. Suppose that a data user $\mathcal{C}_j^{\text{user}}$ who has a vector $\boldsymbol{y}_j = (y_1, ..., y_n)_j$ would apply inner-product functionality over the encrypted data $\{\text{Enc}(x_1), ..., \text{Enc}(x_n)\}$. $\mathcal{C}_j^{\text{user}}$ also select a key service identifier r' first, and then requests the functional private key $\text{sk}^{\text{FE}}_{\boldsymbol{y}_j}$ to the TPA with the vector $\boldsymbol{y}_j$ and r'. At the same time, $\mathcal{C}_j^{\text{user}}$ also sends the request key service snapshot $\mathcal{S}^{\mathcal{C}_j^{\text{user}}}_{\text{req}}$ to $\mathcal{B}^{T^3AB}_{SC}$ as follows:

$$\mathcal{S}^{\mathcal{C}_j^{\text{user}}}_{\text{req}} = \langle r', \boldsymbol{y}_j, t_{\mathcal{C}_j^{\text{user}}}, \text{Sig}_{\text{sk}_{\mathcal{C}_j^{\text{user}}}}(r', 0, t_{\mathcal{C}_j^{\text{user}}})\rangle. \tag{6.15}$$

Unlike the approaches proposed in [191, 189] that deploy the *inference prevention module (IPM)* into the TPA, we propose to deploy IPM in the smart contract as the TPA is not fully trusted in the T^3AB framework. Thus, the TPA needs to query $\mathcal{B}^{T^3AB}_{SC}$ to check the validity result of $\boldsymbol{y}_i$. If $\boldsymbol{y}_i$ is valid, the TPA generates

$\mathrm{sk}_{\boldsymbol{y}_j}^{\mathrm{FE}}$ for $\mathcal{C}_j^{\mathrm{user}}$ using its master keys, and then publishes a corresponding response key service snapshot $\mathcal{S}_{\mathrm{resp}}^{\mathcal{A}}$ to $\mathcal{B}_{SC}^{T^3AB}$ to fulfill its key service audit obligation $\mathcal{O}_{ks}^{\mathcal{C}_j^{\mathrm{user}},\mathcal{A}}$ with mapping key $\mathrm{H}(\mathcal{C}_j^{\mathrm{user}}, \mathcal{A}, r')$ as follows:

$$\mathcal{S}_{\mathrm{resp}}^{\mathcal{A}} = \langle r', \mathrm{H}(\mathrm{sk}_{\boldsymbol{y}_j}^{\mathrm{FE}}), t_{\mathcal{A}}, \mathrm{Sig}_{\mathrm{sk}_{\mathcal{A}}}(r', \mathrm{H}(\mathrm{sk}_{\boldsymbol{y}_j}^{\mathrm{FE}}), t_{\mathcal{A}})\rangle. \tag{6.16}$$

Otherwise, the TPA legally refuses the key service and also publishes key service snapshot indicating that it has legally refuse the key service. $\mathcal{S}_{\mathrm{resp}}^{\mathcal{A},\mathrm{refuse}}$ with refusing symbol $\perp$ to $\mathcal{B}_{SC}^{T^3AB}$ to fulfill its key service audit obligation as follows:

$$\mathcal{S}_{\mathrm{resp}}^{\mathcal{A},\mathrm{refuse}} = \langle r', \mathrm{H}(\perp, \boldsymbol{y}_j), t_{\mathcal{A}}, \mathrm{Sig}_{\mathrm{sk}_{\mathcal{A}}}(r', \mathrm{H}(\mathrm{sk}_{\boldsymbol{y}_j}^{\mathrm{FE}}), t_{\mathcal{A}})\rangle. \tag{6.17}$$

With the received $\mathrm{sk}_{\boldsymbol{y}}^{\mathrm{FE}}$, data user can compute the inner-product of $\langle \boldsymbol{x}, \boldsymbol{y}\rangle$ by decryting as follows:

$$\langle \boldsymbol{x}, \boldsymbol{y}_j\rangle = \mathrm{Dec}_{\mathrm{sk}_{\boldsymbol{y}_j}^{\mathrm{FE}}}(\{\mathrm{Enc}_{\mathrm{pk}_{\mathcal{C}_i^{\mathrm{owner}}}^{\mathrm{FE}}}(x_i)\}_{i\in[n]}). \tag{6.18}$$

Finally, the data owner publishes a receipt for the received $\mathrm{sk}_{\boldsymbol{y}}^{\mathrm{FE}}$.

Remark. To avoid redundant description, we do not present the roles of *auditor* and *monitor* in the above-discussed procedures. In particular, as illustrated in Figure 6.2, the *data users*, *data owners* and the *TPA* also play the role of auditor that checks whether the audit obligations are recorded into the blochchain permanently. In our design, the *data owners* also play the role of a monitor to check the suspicious obligations caused by misbehaviors and malicious behaviors from the TPA and adversarial *data users*. For instance, as illustrated in [191, 189], an adversarial data user may infer the private vector $\boldsymbol{x}$ by manipulating a vector to request the functional private key. The monitor can inspect $\mathcal{O}_{ks}^{e^{\mathrm{user}},e^{\mathrm{TPA}}}$ to find adversary's suspicious behaviors.

6.2 Analysis of Security, Privacy and Trustworthiness

6.2.1 Security Guarantee

The security for the transparency framework is defined in terms of three properties [190, 33]: (i) *log-consistency* - a dishonest public ledger cannot remain undetected if it tries to present inconsistent versions of the record obligations; (ii) *unforgeable-service* - a dishonest TPA cannot forge a key service by sending valid key service snapshots, but not provide the key service to the actors; (iii) *non-fabrication* - a dishonest TPA or actors cannot blame the public ledger for misbehavior if it has behaved honestly, and dishonest actors cannot prove the TPA for misbehavior if it has behaved honestly.

Briefly, *log-consistency* relies on the security properties of the Ethereum blockchain. The *unforgeable-service* and *non-fabrication* properties depend on the designed smart contract functions and the adopted signature scheme. Here, we use the game simulation-based reduction methodology to prove Lemma 4.

Proof. The T^3AB is built on three fundamental security components: the Ethereum blockchain as the public ledger infrastructure, the Secure Hash Algorithm 3 (SHA3) as the collision-resistance hash function,

the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign and validate the origin and integrity of messages. The security of three components has been proved in corresponding related work [184, 63, 93]. We only prove the above-mentioned three security properties.

log-consistency. Unlike the existing transparency framework, [190, 33] that relies on the customized public ledger, our T^3AB uses the public blockchain that has already been proved with secure consistency feature [184], and hence we do not present it here to avoid redundancy.

unforgeable-service. In T^3AB, there are two possible forgeable-service issues:

- the dishonest TPA may publish $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ to the blockchain, but does not send the key sk_f to the actors;
- the dishonest TPA may send an invalid key $\text{sk}_f^{'}$ to the actors, but publishes correct $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ generated from the valid key sk_f.

For the first issue, the confirmation phase cannot be achieved in our designed smart contract, and then such adversarial behavior is easily detected by the monitors. For the second issue, suppose that the dishonest TPA has the non-negligible advantage ϵ to break the *unforgeable-service* security guarantee, and hence it can forge the hashed key component $\text{H}^{SHA3}(\text{sk}_f^{'})$ for sk_f with advantage $\mathbf{Adv}^{\mathcal{A}}_{\text{H}^{SHA3}(\text{sk}_f^{'})\rightarrow \text{sk}_f} \geq \epsilon$.

To achieve that, the dishonest TPA hence needs the ability to find potential collision $\text{H}^{SHA3}(\text{sk}_f) = \text{H}^{SHA3}(\text{sk}_f^{'})$. According to the security promise of SHA3, it is impossible to find that collision with non-negligible advantage. Thus, dishonest TPA does not have a non-negligible advantage to provide an unforgeable key service without being detected.

non-fabrication. In T^3AB, the possible fabrication case is that the dishonest actors attempt to blame the TPA by publishing $\mathcal{S}_{req}^{\mathcal{C}^{\text{actor}}}$ to the blockchain but does not actually send the key request to the TPA. Suppose that the dishonest actor has the non-negligible advantage ϵ to break the *non-fabrication* security promise. To launch the fabrication case, the dishonest actor needs to forge a fake $\mathcal{S}_{\text{resp}}^{\mathcal{A}}$ so that it can accomplish the confirmation phase. Thus, the dishonest actor is able to forge a fake signature of the TPA with advantage $\mathbf{Adv}^{\mathcal{C}^{\text{actor}}}_{\text{sk}_{\mathcal{A}}} \geq \epsilon$. However, it is impossible to break the ECDSA that has been proved, namely, the unforgeability of the signature scheme. Thus, the actors do not have a non-negligible advantage to frame up the TPA. □

6.2.2 Privacy Guarantee

Unlike the *authority transparency* framework that focuses on attribute-based encryption-based applications where partial attribute identities are privacy-sensitive, the T^3AB framework targets the functional encryption-based application scenarios. There is no privacy concern regarding the identity in the FE-based applications. Furthermore, the identity of each entity in T^3AB is the public account address of the Ethereum network, which is a random 64 character hex string generated from the private key of the entity. Thus, such account identities do not reveal any private information, even public identity information in the FE-based applications.

6.2.3 Trustworthiness Goal

The purpose of the T^3AB framework is dealing with the trust issue for potential *dishonest* entities by providing transparency features for them. T^3AB is able to prevent the classic attacks such as stealthy targeted attack and censorship attack as illustrated in [190]. Specifically, each dishonest entity needs to publish the key service snapshot to prove that it has fulfilled its obligation of public parameter distribution and private key service. The designed smart contract can ensure that each entity's submitted audit obligations can be automatically cross-validated based on our designed protocols before being honestly and permanently recorded into the blockchain. Our security analysis has shown that the misbehaviors or malicious behaviors of TPA and the actors are easily detected. Besides, the inference prevention module (IPM) that helps to mitigate the inference attacks caused by the *curious* data users, which is also a critical component in FE-based applications. In our T^3AB, the IPM, previously deployed in the TPA, now is moved to the smart contract.

6.3 Experimental Evaluation

6.3.1 Implementation and Setup

The T^3AB model does not rely on the specific FE-based applications, and hence for generality, we only present the evaluation on a pure T^3AB model with the simulated audit obligations where the key-related components are generated by the FE-based application in an off-line manner.

Implemented Smart Contract. We implemented the smart contracts in *Solidity* programming language using a *Truffle* [1] framework - a development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). Figure 6.3 presents simplified T^3AB smart contract interfaces. T^3AB mainly includes four types of functions as follows:

- *Access Control Modifiers.* The *modifier* can be used to change the behavior of functions in a declarative way. In our implementation, we use the modifier to automatically check the privilege of each account that is defined in the role-based access control (RBAC) module prior to executing the function. We employ the *Ownable* and *AccessControl* smart contract [2] from *OpenZeppelin* as the base of our access control mechanism. To be specific, we define various access control modifiers in which the basic RBAC functions are integrated to satisfy our access control requirement. Except for the registration related functions, other functions are restricted by these modifiers.
- *Administrative and Incentive Functions.* We define several administrative functions such as *'enrollLock()'*, *'enrollOpen()'*, *'dropout()'* that allows the administrator to control the enrollment status. In T^3AB, each

[1] https://www.trufflesuite.com/
[2] https://openzeppelin.com/contracts/

```
pragma solidity ^0.6.0;
pragma experimental ABIEncoderV2;
import "./Ownable.sol";
import "./AccessControl.sol";
contract T3AB is Ownable, AccessControl {
  // access control related modifiers
  bytes32 public constant AUTHORITY_ROLE = keccak256("AUTHORITY_ROLE");
  bytes32 public constant ACTOR_DATA_OWNER_ROLE = keccak256("ACTOR_DATA_OWNER_ROLE");
  bytes32 public constant ACTOR_DATA_USER_ROLE = keccak256("ACTOR_DATA_USER_ROLE");
  bytes32 public constant MONITOR_ROLE = keccak256("MONITOR_ROLE");
  modifier onlyAuthority()
  modifier onlyDataOwner()
  modifier onlyDataUser()
  modifier onlyActor()
  modifier onlyMonitor()
  modifier onlyDeposit()
  modifier onlyWithdrawRegisterCost()
  // administrative and intensive
  function enrollLock() public onlyOwner
  function enrollOpen() public onlyOwner
  function depositeGuarantee() public payable onlyDeposit
  function rewardRegisterCost() public onlyWithdrawRegisterCost
  function rewardDeploymentCost() public onlyOwner
  function _checkGuaranteeDeposit(address account) private returns(bool)
  function _inferencePreventionModule(uint[1] memory y) private returns(bool)
  function dropout() public
  // Registration
  function registerAuthority(bytes memory pk, signature memory sign) public payable
  function registerActorDataOwner(bytes memory pk, signature memory sign) public
  function registerActorDataUser(bytes memory pk, signature memory sign) public
  function registerMonitor(bytes memory pk, signature memory sign) public
  // Obligation
  function recordKSPKReq(bytes32 id, uint pkReqSymbol, uint reqTime, signature memory sign) public onlyDataOwner
  function recordKSPKResp(bytes32 id, bytes32 pkHash, uint respTime, signature memory sign) public onlyAuthority
  function recordKSSKReq(bytes32 id, uint[1] memory y, uint reqTime, signature memory sign) public payable onlyDataUser returns(bool)
  function recordKSSKResp(bytes32 id, bytes32 skHash, uint respTime, signature memory sign) public onlyAuthority
  function recordKSConfirm(address tpa, bytes32 id, bytes32 keyHash, uint confirmTime,
           signature memory signRecpt, signature memory sign) public onlyActor
  // Inspection
  function inspectObligationKS(bytes32 id) public onlyMonitor returns(bool)
  function inspectObligationPP(address addr, bytes memory pk, signature memory sign) public onlyMonitor returns(bool)
}
```

Figure 6.3: Overview of the smart contract interfaces. Note that the parameter 'signature' is our defined customized *struct* that are not presented here. Such a feature is provide in *ABIEncoderV2*.

entity can register if and only if the enrollment is set as open by the administrator. After the enrollment is locked, the deposit operations are opened to the related entities. Besides, T^3AB also inherits the administrative functions such as *'transferOwnership(newOwner)'*, *'renounceOwnership()'* that are not presented in Figure 6.3. These two functions allow transferring the ownership of the contract and leave the contract without owner, respectively. Furthermore, we also defined several withdraw and deposit functions that help to establish the base of the incentive and penalty mechanism.

- *Registration Functions.* The registration functions mainly focus on the initialization phases of the T^3AB model (i.e., Phases I and II, as illustrated in Section 6.1.3), where each entity is allowed to register a role, and publish its *identity-to-public-key* binding in the blockchain.

Table 6.1: Gas cost and test time of selected functions in various test case scenarios

Test Cases	Functions	Gas Cost	Test Time	Description
Administrative	deployment	4125603	183ms	deploy the smart contract
	enrollOpen	44126	42ms	open the enrollment
	enrollLock	14531	46ms	lock the enrollment
	dropout	28293	178ms	allow to drop out and withdraw the balance
Incentive	depositGuarantee	28083	48ms	deposit the guarantee
	rewardRegisterCost	52949	43ms	reward the registration cost for non-payable entity
	rewardDeploymentCost	51584	41ms	reward the deployment for the administrator
Registeration	registerAuthority	38276	80ms	register the role of third-party authority
	registerActorDataOwner	38335	71ms	register the role of data owner
	registerActorDataUser	36555	70ms	register the role of data user
	registerMonitor	36521	72ms	register the role of monitor
Obligation	recordKSSKReq	43173	96ms	publish the key service request snapshot
	recordKSSKResp	84211	55ms	publish the key service response snapshot
	recordKSConfirm	43402	49ms	confirm the receipt of the key service obligation
Inspection	inspectObligationKS	24511	41ms	inspect the key service audit obligation
	inspectObligationPP	37482	46ms	check the correct of the public parameter

- *Obligation Functions.* The obligation functions address the core features of the T^3AB model. As illustrated in Section 6.1.3 Phases III and IV, we use a three-phase commitment approach to achieve the obligation features. To be specific, *'recordKSPKReq', 'recordKSSKReq'* allows the actors to publish the key service request snapshot, while *'recordKSPKResp', 'recordKSSKResp'* allows the TPA to record corresponding key service response snapshot. Then, *'recordKSPKResp'* function allows us to confirm the receipt of the key service.

- *Inspection Functions.* The inspection functions address the monitoring task for the recorded audit obligation as discussed in Section 6.1.3. To be specific, *'inspectObligationKS'* function allows to automatically inspect the completeness of the key service obligations, while *'inspectObligationPP'* function permits the monitor to verify the published *identity-to-public-key* binding. Regarding the incentive design, if the dishonest behavior is detected, the corresponding entity will be fined a fixed number of *ether* as the incentive reward for the monitor.

Experimental Setup. Our experiments are performed on a Macbook Pro platform with 2.3GHz 8-Core Intel Core i9 processors and 32GB DDR4 memory. Besides, we use the Ethereum official test network - Rinkeby as the experimental environment to deploy our implemented smart contract. Furthermore, we write several *JavaScript* test-cases using the automated testing framework of *Truffle* that is built on Mocha[3] and provides a cleanroom environment.

Specifically, for demonstration, we use five Ethereum accounts to simulate various entities in T^3AB, namely, the role of the *administrator*, the *TPA*, the *data owner*, the *data user* and the *monitor*. With regards to various scenarios, we write corresponding test-cases to evaluate the performance (i.e., the gas cost and the time cost) such as the administrative scenario, registration scenario, obligation scenario, etc..

[3]https://mochajs.org/

6.3.2 Experiment Results

We report the performance of T^3AB for selected typical functions in various test case scenarios in Table 6.1. In particular, the performance includes two aspects: the gas cost and the test time. Gas is spent in Ethereum for deploying smart contracts or calling functions. As reported in Table 6.1, most functions cost very little. Specifically, except for the smart contract deployment, the cost of each function is at the level of 10^5 gas in general. Regarding the highly called functions for obligation and inspection, to record an audit obligation for one key service, the functions of three-phase commitment (i.e., *recordKSSKReq*, *recordKSSKResp*, *recordKSConfirm*) cost 3.7×10^5 gas, 8.4×10^5 gas, 4.3×10^5 gas, respectively. Besides, the cost of inspection for key service and public parameter audit obligations is 2.4×10^5 gas and 3.7×10^5 gas, respectively.

Furthermore, we also measure the time it takes to test the selected functions. Except for the administrative functions, the calling time of rest of the functions is less than $100ms$. Note that the time to test each function is measured in the Ethereum test network. The testing time is related to execution time instead of time taken to confirm transaction . Thus, the deployment time of the smart contract is only $183ms$ rather than the general time taken to confirm a transaction, namely, about 6 minutes.

6.4 Summary and Discussion

In this chapter, we have proposed the T^3AB framework to provide transparency and trustworthiness of the *third-party authority (TPA)* and *honest-but-curious* entities in the recently proposed FE-based privacy-preserving machine learning (PPML) systems. T^3AB employs the Ethereum blockchain as the underlying public ledger infrastructure and also includes a novel smart contract mechanism to support accountability. In addition, it includes an incentive mechanism to motivate participants' audit and punish the misbehaviors or malicious behaviors. We presented the evaluation of the proposed framework which shows that the framework is efficient in the Ethereum official test network, and achieves the security and privacy guarantee and trustworthiness goal.

7.0 Conclusion and Future Research

7.1 Conclusion

Privacy-preserving ML (PPML) is critical for ML-powered systems as they typically need to use users' privacy-sensitive data for training and prediction phases. To tackle privacy challenges in ML, novel PPML techniques are needed. Towards this, recent approaches proposed in the literature have aimed at integrating existing privacy-preserving methods into ML models. While various types of privacy-preserving approaches have been proposed, they have their own limitations. For instance, the use of a garbled circuits-based secure computation approach incur high communication overhead because of a large volume of intermediate data that needs to be transmitted during the execution of the protocol, while the adoption of emerging differential privacy raises challenges related to trade-offs between model accuracy and privacy budget.

In this dissertation, we have discussed various PPML approaches that have been proposed in the literature and their limitations. We have mainly focused on secure computation approaches, one of the key privacy-preserving techniques in PPML, in which the released data is protected by a cryptosystem and the ML related computations occur over encrypted data. Specifically, we have focused on secure computation approaches for PPML in the context of two-tier and three-tier architectures, and emerging DL models and FL. The central piece of the secure computation solutions we have proposed is functional encryption. In this dissertation, we make the following key contributions:

- For two-tier PPML architecture, we have proposed secure computation solutions to achieve PPFL in Chapter 3. We have proposed the *HybridAlpha* framework to support efficient training in horizontal PPFL as well as providing a strong privacy guarantee. Only a few existing approaches address vertical PPFL problems, where they only support a specific machine learning model and suffer from inefficiency in terms of both secure computation and training time. To tackle those challenges, we have proposed the *FedV* framework built on a well-designed secure SGD approach that makes use of functional encryption schemes. The security and privacy analysis show that *HybridAlpha* and *FedV* achieves privacy and security goals. We also implemented and experimentally evaluated *HybridAlpha* and *FedV* and the results show that they can reduce the training time and total data transfer volume significantly without sacrificing privacy guarantee and model performance.
- We have also proposed *NN-EMD* in Chapter 4, which is a novel privacy-preserving DNN model applicable in a two-tier cloud-client PPML architecture, where the raw data is protected by functional encryption schemes and the training task is done over encrypted data. *NN-EMD* also supports multiple data sources, where the data may be composed of horizontally and vertically partitioned datasets. The evaluation shows that *NN-EMD* can reduce the training time while still providing the same model accuracy and strong privacy guarantee as compared to most of the recent comparable solutions. Unlike existing HE-based

solutions, the depth and complexity of DNNs in our *NN-EMD* do not affect the training time despite integrating a privacy-preserving component in NN-EMD.

- In edge-enabled three-tier PPML architecture, edge nodes can pre-process data from the client devices such as sensors or mobile devices. Our proposed *CryptoEdge* solution in Chapter 5 enable pre-processing at the edge and final processing at the cloud, both over the encrypted data or model; this approach helps protect the privacy of data generated by the sensors or mobile devices, where the data and model are protected by our proposed threshold functional encryption scheme. The proposed *CryptoEdge* supports various types of secure aggregations at the edge and PPFL framework in a three-tier PPML architecture. We evaluated *CryptoEdge* in a simulated cloud-edge environment, and our evaluation results show that *CryptoEdge* provides significant performance improvement on system efficiency as well as achieves the security and privacy guarantees.
- To address the issues related to trustworthiness of a third-party authority (TPA) and commercial infrastructures such as cloud service providers that are essential for the proposed secure computation approaches, we have proposed a framework in Chapter 6 to address transparency and accountability issues so as to increase the trust for entities such as a TPA and other honest-but-curious entities in the proposed FE-based PPML solutions. We show that the proposed transparency framework is effective and efficient in the Ethereum test network and guarantees the security properties of log-consistency, unforgeable-service, and non-fabrication.

7.2 Future Research

Here, we present some possible future research directions:

- Our proposed solutions for three-tier PPML architecture only support various types of secure aggregations, and hence the supported ML-enabled applications are limited to those that only need secure aggregation operations such as privacy-preserving horizontal federated learning. There is still a lack of secure computation approaches to address more complex computational tasks to support more complex PPML such as privacy-preserving vertical federated learning and PPML in a three-tier architecture. To tackle the challenge, it will be worth exploring various threshold functional encryption (TFE) schemes beyond the multi-client TFE scheme proposed in this dissertation. Based on the newly proposed TFE schemes in the literature and our proposed TFE scheme, one possible direction is to explore extensions to these to solve more complex secure computation tasks rather than the aggregation tasks in PPML.
- Note that the solutions proposed in this dissertation are built using existing single-client or multi-client functional encryption schemes and our proposed threshold functional encryption scheme. A common and critical component in these schemes is a trusted third-party authority (TPA) that provides the key service such as generating entity-specific public keys and functional private keys. Another direction

worth exploring include more efficient and decentralized functional encryption based schemes that do not depend on a TPA.

- We have explored primarily two-tier PPML architecture and three-tier PPML architecture in this dissertation. We believe there is still a need to explore various other architectures under various security assumptions. For instance, one research direction maybe addressing secure computation in multi-cloud environment, where the third-party cloud providers are not fully trusted by an enterprise to process its private business data.

In summary, ML represents a very critical technological innovation that has a huge potential for a transformative societal impact. And we are already seeing that in many ML-enabled applications in many domains. As we make widely available powerful and novel ML solutions, it is important to ensure that their privacy-implications are properly understood and addressed. The goal of this dissertation has been to identify some key research challenges related to ensuring privacy protection when various ML techniques are employed in different kinds of architectural context. Accordingly, we have proposed various approaches based on functional encryption schemes that provide practical solutions to deploying ML models while ensuring privacy guarantees. We believe that more work needs to be done to address various related challenges as discussed above and explore newer approaches that will further improve efficiency while ensuring security and privacy guarantees. Such solutions will ensure that our society can reap the benefits of ML technologies without having to worry much about individual privacy, a fundamental for our society.

Appendix A

Functional Encryption

In this chapter, we introduce the notations and definitions of functional encryption (FE). Then, we present the specific constructions of single-input functional encryption (SIFE) and multi-input functional encryption (MIFE) for functionality of inner-product.

A.1 Definitions

Following the initial definition from [25] and [5], we present the notion of functionality, the functional encryption scheme, security assumption and security definition.

A.1.1 Functionality

Definition 5 (Functionality [25]). *A functionality $\mathcal{F}$ defined over (K, X) is a function $\mathcal{F} : K \times X \to \Sigma \cup \{\perp\}$ where K is the key space, X is the message space and Σ is the output space and $\perp$ is a special string not contained in Σ.*

Note that the functionality is undefined when either the key is not in the keyspace or the message is not in the message space.

A.1.2 Functional Encryption Scheme

Definition 6 (Functional Encryption Scheme [25]). *A functional encryption (FE) scheme for functionality $\mathcal{F}$ is a tuple $\mathcal{E}^{\mathcal{F}}_{FE}$ = (Setup, KeyDerive, Encrypt, Decrypt) of four algorithms:*

- $Setup(1^\lambda)$ *outputs public and master secret keys (mpk, msk) for security parameter λ;*
- $KeyDerive(msk, k)$ *outputs secret key sk_k on input a master secret key msk and key $k \in K$;*
- $Encrypt(mpk, x)$ *outputs ciphertext ct on input public key mpk and message $x \in X$;*
- $Decrypt(mpk, ct, sk_x)$ *outputs $z \in \Sigma \cup \{\perp\}$.*

Note that the *correctness* of $\mathcal{E}^{\mathcal{F}}_{\text{FE}}$ is described as $\forall(mpk, msk) \leftarrow Setup(1^\lambda)$, $\forall k \in K, x \in X$, for $sk_k \leftarrow KeyDerive(msk, k)$ and $ct \leftarrow Encrypt(mpk, x)$, we have that $Decrypt(mpk, ct, sk_x) = \mathcal{F}(x, k)$ whenever $\mathcal{F}(x, k) \neq \perp$, except with negligible probability.

A.1.3 The Decisional Diffie-Hellman Assumption

Consider a (multiplicative) cyclic group $\mathbb{G}$ of order q, and with generator g. The *Decisional Diffie-Hellman (DDH) assumption* states that the tuples (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable, where $a, b, c \in \mathbb{Z}_p$ are chosen independently and uniformly at random.

A.1.4 Security of Functional Encryption

Definition 7 (Selective Simulation-based Secure FE [5])**.** *A functional encryption $\mathcal{E}_{FE}$ for functionality $\mathcal{F}$ is selective simulation-based secure (SEL-SIM-secure) if there exist PPT simulator algorithms*

$$\mathcal{E}_{FE}^{SIM} = (Setup^{SIM}, KeyDerive^{SIM}, Encrypt^{SIM}, Decrypt^{SIM}), \tag{A.1}$$

such that for every stateful PPT adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, the following two distributions are computationally indistinguishable:

Experiment $\mathbf{REAL}^{\mathcal{E}_{FE}}_{\text{SEL}}(1^\lambda, \mathcal{A})$
$\{x_i\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda, \mathcal{F})$
$(mpk, msk) \leftarrow Setup(1^\lambda, \mathcal{F})$
$\forall i \in [n], ct_i \leftarrow Encrypt(mpk, i, x_i)$
$\alpha \leftarrow \mathcal{A}^{KeyDerive(msk)}(mpk, \{ct_i\}_{i\in[n]})$
Output : α

Experiment $\mathbf{IDEAL}^{\mathcal{E}_{FE}}_{\text{SEL}}(1^\lambda, \mathcal{A})$
$\{x_i\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda, \mathcal{F})$
$(mpk^{\text{SIM}}, msk^{\text{SIM}}) \leftarrow Setup^{\text{SIM}}(1^\lambda, \mathcal{F})$
$\forall i \in [n], ct_i \leftarrow Encrypt^{\text{SIM}}(mpk^{\text{SIM}}, i)$
$\alpha \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(mpk^{SIM}, \{ct_i\}_{i\in[n]})$
Output : α

The oracle $\mathcal{O}(\cdot)$ in the ideal experiment above is given access to another oracle that, given $f \in \mathcal{F}$, returns $f(x_1, ..., x_n)$, and then $\mathcal{O}(\cdot)$ returns $KeyDerive^{SIM}(msk^{SIM}, f, f(x_1, ..., x_n))$.

Note that for every stateful adversary $\mathcal{A}$, we define its advantage as follows:

$$\text{Adv}^{\text{SEL-SIM}}_{\mathcal{A},\mathcal{E}_{\text{FE}}} = |\Pr[\mathbf{REAL}^{\mathcal{E}_{\text{FE}}}_{\text{SEL}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathbf{IDEAL}^{\mathcal{E}_{FE}}_{SEL}(1^\lambda, \mathcal{A})]|. \tag{A.2}$$

We also require that for every PPT $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that

$$\forall \lambda \in \mathbb{N}, \text{Adv}^{SEL-SIM}_{\mathcal{A},\mathcal{E}_{FE}} = \text{negl}(\lambda). \tag{A.3}$$

A.2 Single-Client FE for Functionality of Inner-Product

The single-input FE scheme for the functionality of inner-product $\mathcal{E}^{\mathcal{F}_{\text{SCIP}}}_{\text{FE}}$ is defined as

$$\mathcal{E}^{\mathcal{F}_{\text{SCIP}}}_{\text{FE}} = (\mathcal{E}^{\mathcal{F}_{\text{SCIP}}}_{\text{FE}}.S, \mathcal{E}^{\mathcal{F}_{\text{IP}}}_{\text{FE}}.K, \mathcal{E}^{\mathcal{F}_{\text{SCIP}}}_{\text{FE}}.E, \mathcal{E}^{\mathcal{F}_{\text{SCIP}}}_{\text{FE}}.D), \tag{A.4}$$

where the functionality is defined as

$$\mathcal{F}_{\text{SCIP}}(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{\eta} (x_i y_i). \tag{A.5}$$

Each of the algorithm is constructed as follows:

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{SCIP}}}.S \rightarrow$ *Setup*$(1^\lambda, \eta)$: The algorithm first generates two samples as $(\mathbb{G}, p, g) \leftarrow_\$$ *GroupGen*(1^λ), and $\boldsymbol{s} = (s_1, ..., s_\eta) \leftarrow_\$ \mathbb{Z}_p^\eta$ on the inputs of security parameters λ and η, and then sets pk and msk as follows:

$$pk = (g, h_i = g^{s_i})_{i \in [1,...,\eta]} \tag{A.6}$$

$$msk = \boldsymbol{s} \tag{A.7}$$

It returns the pair (pk, msk).

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{SCIP}}}.K \rightarrow$ *SKGenerate*$(msk, \boldsymbol{y})$: The algorithm outputs the function secret key $sk_f = \langle \boldsymbol{y}, \boldsymbol{s} \rangle$ on the inputs of master secret key msk and vector $\boldsymbol{y}$.
- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{SCIP}}}.E \rightarrow$ *Encrypt*$(pk, \boldsymbol{x})$: The algorithm first chooses a random $r \leftarrow_\$ \mathbb{Z}_p$ and computes $ct_0 = g^r$. For each $i \in [1, ..., \eta]$, it computes $ct_i = h_i^r \cdot g^{x_i}$. Then the algorithm outputs the ciphertext $ct = (ct_0, \{ct_i\}_{i \in [1,...,\eta]})$.
- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{SCIP}}}.D \rightarrow$ *Decrypt*$(pk, ct, sk_f, \boldsymbol{y})$: The algorithm takes the ciphertext ct, the public key mpk and functional key sk_f for the vector $\boldsymbol{y}$, and returns the discrete logarithm in basis g, as follows:

$$g^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle} = \prod_{i \in [1,...,\eta]} ct_i^{y_i} / ct_0^{sk_f}. \tag{A.8}$$

A.3 Multi-Client FE for Functionality of Inner-Product

The multi-input functional encryption scheme for the inner-product $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}$ is defined as

$$\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}} = (\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.S, \mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.PK, \mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.SK, \mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.E, \mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.D). \tag{A.9}$$

where the functionality is defined as

$$\mathcal{F}_{\mathrm{MCIP}} = \langle \{\boldsymbol{x}_i\}_{i \in [n]}, \boldsymbol{y} \rangle = \sum_{i \in [n]} \sum_{j \in [\eta_i]} (x_{ij} y_{\sum_{k=1}^{i-1} \eta_k + j}) \quad \text{s.t.} \quad |\boldsymbol{x}_i| = \eta_i, |\boldsymbol{y}| = \sum_{i \in [n]} \eta_i, \tag{A.10}$$

The specific construction of each algorithm is defined follows:

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.S \rightarrow$ *Setup*$(1^\lambda, \vec{\boldsymbol{\eta}}, n)$: The algorithm first generates secure parameters as $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow_\$$ *GroupGen*(1^λ), and then generates several samples as $a \leftarrow_\$ \mathbb{Z}_p$, $\boldsymbol{a} = (1, a)^\intercal$, $\forall i \in [1, ..., n] : \boldsymbol{W}_i \leftarrow_\$ \mathbb{Z}_p^{\eta_i \times 2}$, $\boldsymbol{u}_i \leftarrow_\$ \mathbb{Z}_p^{\eta_i}$. Then, it generates the *master public key* and *master private key* as

$$mpk = (\mathcal{G}, g^{\boldsymbol{a}}, g^{\boldsymbol{W}\boldsymbol{a}}), \tag{A.11}$$

$$msk = (\boldsymbol{W}, (\boldsymbol{u}_i)_{i \in [1,...,n]}). \tag{A.12}$$

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.PK \rightarrow$ *PKDistribute*(mpk, msk, id_i): It looks up the existing keys via id_i and returns the *public key* as

$$pk_i = (\mathcal{G}, g^{\boldsymbol{a}}, (\boldsymbol{W}\boldsymbol{a})_i, \boldsymbol{u}_i). \tag{A.13}$$

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.SK \rightarrow SKGenerate(mpk, msk, \boldsymbol{y})$: The algorithm first partitions $\boldsymbol{y}$ into $(\boldsymbol{y}_1||\boldsymbol{y}_2||...||\boldsymbol{y}_n)$, where $|\boldsymbol{y}_i|$ is equal to η_i. Then it generates the function derived key as

$$sk_{f,\boldsymbol{y}} = (\{\boldsymbol{d}_i^{\intercal} \leftarrow \boldsymbol{y}_i^{\intercal}\boldsymbol{W}_i\}, z \leftarrow \sum \boldsymbol{y}_i^{\intercal}\boldsymbol{u}_i). \tag{A.14}$$

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.E \rightarrow Encrypt(pk_i, \boldsymbol{x}_i)$: The algorithm first generates a random nonce $r_i \leftarrow_R \mathbb{Z}_p$, and then computes the ciphertext as

$$\boldsymbol{ct}_i = (\boldsymbol{t}_i \leftarrow g^{\boldsymbol{a}r_i}, \boldsymbol{c}_i \leftarrow g^{\boldsymbol{x}_i} g^{\boldsymbol{u}_i} g^{(\boldsymbol{W}\boldsymbol{a})_i r_i}). \tag{A.15}$$

- $\mathcal{E}_{\mathrm{FE}}^{\mathcal{F}_{\mathrm{MCIP}}}.D \rightarrow Decrypt(\boldsymbol{ct}, sk_{f,\boldsymbol{y}})$: The algorithm first calculates as follows:

$$C = \frac{\prod_{i\in[1,...,n]}([\boldsymbol{y}_i^{\intercal}\boldsymbol{c}_i]/[\boldsymbol{d}_i^{\intercal}\boldsymbol{t}_i])}{z}, \tag{A.16}$$

and then recovers the function result as

$$f((\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n), \boldsymbol{y}) = \log_g(C). \tag{A.17}$$

Appendix B

Differential Privacy

B.1 Differential Privacy

Differential privacy (DP) [60, 62] is a rigorous mathematical framework where an algorithm may be described as differentially private if and only if the inclusion of a single instance in the training dataset causes only statistically insignificant changes to the algorithm's output. The formal definition for DP is as follows:

Definition 8 (Differential Privacy [60])**.** *A randomized function $\mathcal{K}$ gives (ϵ, δ)-differential privacy if for all data sets D and $D^{'}$ differing on at most one element, and all $S \subseteq \texttt{Range}(\mathcal{K})$,*

$$\Pr[\mathcal{K}(D) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{K}(D^{'}) \in S] + \delta. \tag{B.1}$$

The probability is taken over the coin tosses of $\mathcal{K}$.

The additive term δ allows for the possibility that plain ϵ-differential privacy is broken with probability δ (which is preferably smaller than $1/|d|$). Usually, a paradigm of an approximating a deterministic function $f : \mathcal{D} \rightarrow \mathbb{R}$ with a differentially private mechanism is via *additive noise* calibrated to function's *sensitivity* S_f that is defined as the maximum of the absolute distance $|f(d) - f(d^{'})|$. The representative and common additive noise mechanisms for real-valued functions are Laplace mechanism and Gaussian mechanism, as respectively defined as follows:

$$\mathcal{M}_{\text{Gauss}}(d; f, \epsilon, \delta) = f(d) + \mathcal{N}(\mu, \sigma^2) = f(d) + \mathcal{N}(0, \frac{2\ln(1.25/\delta)}{\epsilon^2} \cdot S_f^2) \tag{B.2}$$

$$\mathcal{M}_{\text{Lap}}(d; f, \epsilon) = f(d) + Lap(\mu, b) = f(d) + Lap(0, \frac{S_f^2}{\epsilon}) \tag{B.3}$$

Note that ϵ-*differential privacy* can be treated as a special case of (ϵ, δ)-*differential privacy* where $\delta = 0$. To achieve DP, multiple mechanisms designed to inject noise to the algorithm's output have been proposed. These mechanisms add noise proportional to the sensitivity of the output, a measure of the maximum change of the output resulting by the inclusion of a single data point. Popular mechanisms include Laplacian and Gaussian mechanisms, where the Gaussian mechanism for a dataset D is defined as $M(D) = f(D) + N(0, S_f^2\sigma^2)$, where $N(0, S_f^2\sigma^2)$ is the normal distribution with mean 0 and standard deviation $S_f\sigma$. By applying the Gaussian mechanism to function f with sensitivity S_f satisfies (ϵ, δ)-differential privacy [61].

B.2 Noise Reduction through SMC

SMC allows multiple parties to compute a function over their inputs, without revealing their individual inputs [21, 49]. SMC can be achieved using different techniques such as garbled circuit with oblivious transfer, fully or partially homomorphic encryption, and functional encryption.

Prior work has shown that it is possible to maintain the same DP guarantee achieved by *local differential privacy* [95, 145], i.e., each party adds its own noise independently, and uses SMC to hide individual inputs. Concretely, using the Gaussian mechanism defined above, local differential privacy requires each participant to independently add $N(0, S_f^2\sigma^2)$. Considering n parties, the total noise adds up to n. However, when applying SMC each participant can add a fraction of the noise $N(0, \frac{1}{n}S^2\sigma^2)$ and then use a SMC technique to share the value for aggregation. As shown in [173], this ensures the same DP guarantee while reducing the amount of total noise injected by a factor of n.

Appendix C

Appendix of *FedV*

C.1 Specific Analysis of Lemma 1

Here, we present our detailed analysis of Lemma 1. Note that we skip the discussion on how to compute ∇R in the rest of the proof such as in (C.3), since the coordinator can compute it independently.

C.1.1 Linear models in *FedV*

Here we formally analyze the details of how our proposed *2Phase-SA* approach is applied in a vertical federated learning framework with underlying linear machine learning model. Suppose the a generic *linear model* is defined as:

$$f(\boldsymbol{x};\boldsymbol{w}) = w_0x_0 + w_1x_1 + ... + w_jx_j, \tag{C.1}$$

where $x_0^{(i)} = 1$ represents the bias term. For simplicity, we use the vector-format expression in the rest of the proof, described as: $f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^\intercal\boldsymbol{x}$, where where $\boldsymbol{x} \in \mathbb{R}^{d+1}, \boldsymbol{w} \in \mathbb{R}^{d+1}, x_0 = 1$. Suppose that the loss function here is least-squared function, defined as

$$\mathcal{L}(f(\boldsymbol{x};\boldsymbol{w}), y) = (f(\boldsymbol{x};\boldsymbol{w}) - y)^2 \tag{C.2}$$

and here we use L2-norm as the regularization term, defined as: $R(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n} w_i^2$. According to equations (3.2), (C.1) and (C.2), the gradient of $E(\boldsymbol{w})$ computed over a mini-batch $\mathcal{S}$ of $n_\mathcal{S}$ data samples is as follows:

$$\nabla E_\mathcal{S}(\boldsymbol{w}) = \nabla\mathcal{L}_\mathcal{S}(\boldsymbol{w}) + \nabla R_\mathcal{S}(\boldsymbol{w}) = \frac{2}{n_\mathcal{S}}\sum_{i}^{n_\mathcal{S}}(\boldsymbol{w}^\intercal\boldsymbol{x}^{(i)} - y^{(i)})\boldsymbol{x}^{(i)}$$

The secure gradient computation is described as (suppose that p_1 is the *active party* with labels y)

$$\nabla\mathcal{L} = \frac{2}{n}\sum_{i}^{n}(w_0x_0^{(i)} + \underbrace{w_1x_1^{(i)} - y^{(i)}}_{u_{p1}^{(i)}} + ... + \underbrace{w_jx_j^{(i)}}_{u_{pl}^{(i)}})\boldsymbol{x}^{(i)} = \frac{2}{n}\sum_{i}^{n}\underbrace{\sum_{j}^{d}(u_{p_j}^{(i)})}_{\text{F-SA}}\boldsymbol{x}^{(i)} \tag{C.3}$$

Next, let $u^{(i)}$ be the intermediate value to represent the *difference-loss* for current $\boldsymbol{w}$ over one sample $\boldsymbol{x}^{(i)}$, which is also the aggregation result of *F-SA*. Then, the updated gradient $\nabla E(\boldsymbol{w})$ is continually computed as follows:

$$\nabla\mathcal{L} = \frac{2}{n}\sum_{i}^{n}u^{(i)}(x_0^{(i)}, \underbrace{x_1^{(i)}}_{p_1}, ..., \underbrace{x_{j-1}^{(i)}}_{p_{l-1}}, \underbrace{x_j^{(i)}}_{p_l}) = \frac{2}{n}(\sum_{i}^{n}u^{(i)}x_0^{(i)}, \underbrace{\sum_{i}^{n}u^{(i)}x_{1,p_1}^{(i)}}_{\text{S-SA}}, ..., \underbrace{\sum_{i}^{n}u^{(i)}x_{j,p_l}^{(i)}}_{\text{S-SA}}) \tag{C.4}$$

To deal with the secure computation task of training loss as described in Algorithm 3, we only apply *F-SA* approach. As the average loss function here is least-squares function, the secure computation is described as

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) = \underbrace{\frac{1}{n}\sum_{i}^{n}(\underbrace{\boldsymbol{w}^{\intercal}\boldsymbol{x}^{(i)} - y^{(i)}}_{\text{F-SA}})^2}_{\text{Normal Computation}} \tag{C.5}$$

Obviously, the *F-SA* could satisfy the computation task in the above equation.

C.1.2 Generalized linear models in *FedV*

Here we formally analyze the details of applying our *2Phase-SA* approach to train generalized linear models in *FedV*.

We use *logistic regression* as an example, which has the following fitting (prediction) function:

$$f(\boldsymbol{x};\boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{w}^{\intercal}\boldsymbol{x}}} \tag{C.6}$$

For binary label $y \in \{0,1\}$, the loss function could be defined as:

$$\mathcal{L}_{\mathcal{D}}(f(\boldsymbol{x};\boldsymbol{w}),y) = \begin{cases} -\log(f(\boldsymbol{x};\boldsymbol{w})) \text{ if } y=1 \\ -\log(1-f(\boldsymbol{x};\boldsymbol{w})) \text{ if } y=0 \end{cases} \tag{C.7}$$

The gradient computation over a mini-batch S of size n can be described as:

$$\nabla E_S(\boldsymbol{w}) = \frac{1}{n}\sum_{i\in S}^{n}(\frac{1}{1+e^{-\boldsymbol{w}^{\intercal}\boldsymbol{x}^{(i)}}} - y^{(i)})\boldsymbol{x}^{(i)} \tag{C.8}$$

Note that we also do not include the regularization term $\lambda R(\boldsymbol{w})$ here for the same aforementioned reason. Here we show the above-mentioned two solutions in detail:

(i) *Taylor approximation.* In this approach, the Taylor series expansion of function $\log(1+e^{-z})$=$\log 2 - \frac{1}{2}z + \frac{1}{8}z^2 + O(z^4)$ is applied to the equation (C.7) to approximately generate gradient as follows:

$$\nabla E_S(\boldsymbol{w}) \approx \frac{1}{n}\sum_{i\in S}^{n}(\frac{1}{4}\boldsymbol{w}^{\intercal}\boldsymbol{x}^{(i)} - y^{(i)} + \frac{1}{2})\boldsymbol{x}^{(i)} \tag{C.9}$$

Similar to equation (C.3), we are able to apply the *2Phase-SA* approach in the secure computation of equation (C.9).

(ii) *Decomposition-then-2Phase-SA*. Different from above-discussed approximation approach, we present our *decomposition-then-2Phase-NS* approach to compute the exact value of the gradient which is expressed as a nonlinear formula. First, although the prediction function (C.6) is a non-linear function, it can be decomposed to $f(\boldsymbol{x};\boldsymbol{w}) = g(h(\boldsymbol{x};\boldsymbol{w}))$ as follows:

$$g(h(\boldsymbol{x};\boldsymbol{w})) = \frac{1}{1+e^{-h(\boldsymbol{x};\boldsymbol{w})}} \rightarrow h(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^{\intercal}\boldsymbol{x} \tag{C.10}$$

Algorithm 8: FedV-SLC

Input: current model weights $\mathbf{w}$; participants $\mathcal{P}$, where each participant p_i has assigned data pre-process parameter π_{p_i}, dataset $\mathcal{D}_{p_i}$ and public key pk_i

function *coordinator-process*$(\boldsymbol{w}, \mathcal{P}, \{\pi_{p_i}\}, \boldsymbol{y})$

  foreach $p_i \in \mathcal{P}$ **do** query p_i with $\text{msg}_{q,p_i} = (\boldsymbol{w}_{p_i}, \pi_{p_i})$;

  do $\mathcal{S}_{\text{msg}_r} \leftarrow$ collect participants' response msg_{r,p_i} **while** *enough* $\mathcal{S}_{msg_r}$ ***and*** *still in max waiting time*;

  specify $\boldsymbol{w}_{\mathcal{P}'}$ according to $\mathcal{S}_{\text{msg}_r}$;

  foreach $msg_{r,p_i} \in \mathcal{S}_{msg_r}$ **do** $\boldsymbol{u} \leftarrow$ add result of *F-SA* on $\text{msg}_{r,p_i}.\boldsymbol{c}_{\text{fd}}$ with $\boldsymbol{w}_{\mathcal{P}'}$;

  $E_{\mathcal{D}}(\boldsymbol{w}) \leftarrow \boldsymbol{y} \log \frac{1}{1+e^{-\boldsymbol{u}}} + (1-\boldsymbol{y}) \log(1 - \frac{1}{1+e^{-\boldsymbol{u}}})$;

We can see that the sigmoid function $g(z) = \frac{1}{1+e^{-z}}$ is not a linear function, while $h(\boldsymbol{x}; \boldsymbol{w})$ is linear. We then apply *2Phase-SA* on linear $h(\boldsymbol{x}; \boldsymbol{w})$ instead. To be more specific, the formal description of secure gradient computation is presented below:

$$\nabla \mathcal{L} = \frac{1}{n} \sum_{i \in S} \underbrace{(\frac{1}{1 + e^{-\sum_j^d (u_{p_j}^{(i)})\} \rightarrow \text{F-SA}}} - y^{(i)})}_{\text{Normal Computation} \rightarrow u^{(i)}} \boldsymbol{x}^{(i)} = \frac{1}{n} (\sum_i^n u^{(i)} x_0^{(i)}, \underbrace{\sum_i^n u^{(i)} x_{1,p_1}^{(i)}}_{\text{S-SA}}, ..., \underbrace{\sum_i^n u^{(i)} x_{j,p_l}^{(i)}}_{\text{S-SA}})$$

Note that the output of *F-SA* is in plaintext, and hence the coordinator is able to compute sigmoid function $g(\cdot)$ and labels. The secure loss computation is described as

$$E_{\mathcal{D}}(\boldsymbol{w}) = -\frac{1}{n} \sum_{i \in \mathcal{D}}^n y^{(i)} \log \frac{1}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)}\} \rightarrow \text{F-SA}}} + \underbrace{(1 - y^{(i)}) \log(1 - \frac{1}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)}\} \rightarrow \text{F-SA}}})}_{\text{Normal Computation}} \tag{C.11}$$

Similar to secure gradient descent computation, however, we only have the *F-SA* with subsequent normal computation.

Note that in this *decomposition-then-2Phase-SA* approach, it requires to expose labels to the coordinator. Here we iterate two different cases: (i) if the VFL framework has the role of *coordinator*, our solution requires the *active participant* expose its labels to the *coordinator*; (ii) if the VFL framework does not have the role of *coordinator*, usually, the *active participant* will play the role of the "coordinator", and hence, there is no need to expose labels.

Here, we also use another machine learning model, *SVMs with Kernels* as an example.

The model could be presented as

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i=1}^n w_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}), \tag{C.12}$$

where $k(\cdot)$ is the kernel function. For kernel functions such as linear kernel $\boldsymbol{x}_i^\intercal \boldsymbol{x}_j$, polynomial kernel $(\boldsymbol{x}_i^\intercal \boldsymbol{x}_j)^d$, sigmoid kernel $tanh(\beta \boldsymbol{x}_i^\intercal \boldsymbol{x}_j + \theta)$. In the SGD algorithm, to train a SVM with kernels model, we need to iteratively update the initialize $\boldsymbol{w}^{(t)}$ by comparing the result of $\sum_{i=1}^n w_i^{(t)} y_i k(\boldsymbol{x}_i, \boldsymbol{x_j})$ to the label. As such supported kernel functions is based on inner-product computation, which is also supported by our *F-SA* and *S-SA* protocols.

C.2 Secure Loss Computation in *FedV*

Unlike the *secure loss computation (SLC)* protocol in the contrasted VFL framework [83], the SLC approach in *FedV* is much simpler. Here we use the logistic regression model as an example. As illustrated in Algorithm 8, unlike the SLC in [83] that is separate and different from the secure gradient computation, the SLC in does not need additional operations for the participants. The loss result is computed by reusing the result of the *F-SA* in the *FedV-SecGrad*.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product functional encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–582. Springer, 2019.

[3] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In *IACR International Workshop on Public Key Cryptography*, pages 128–157. Springer, 2019.

[4] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In *IACR International Workshop on Public Key Cryptography*, pages 733–751. Springer, 2015.

[5] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings. In *Annual International Cryptology Conference*, pages 597–627. Springer, 2018.

[6] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4):1–35, 2018.

[7] Charu C Aggarwal and Philip S Yu. On privacy-preservation of text and sparse binary data with sketches. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 57–67. SIAM, 2007.

[8] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. Quotient: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1231–1247, 2019.

[9] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[10] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 21–40. Springer, 2011.

[11] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[12] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S Dov Gordon, Stefano Tessaro, and David A Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In *IMA International Conference on Cryptography and Coding*, pages 65–84. Springer, 2013.

[13] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In *Theory of Cryptography Conference*, pages 174–198. Springer, 2019.

[14] Nuttapong Attrapadung and Benoît Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In *International Workshop on Public Key Cryptography*, pages 384–402. Springer, 2010.

[15] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[16] Raghavendran Balu and Teddy Furon. Differentially private matrix factorization using sketching techniques. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, pages 57–62, 2016.

[17] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. Design, analysis, and implementation of arpki: an attack-resilient public-key infrastructure. *IEEE Transactions on Dependable and Secure Computing*, 15(3):393–408, 2016.

[18] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 578–590, 2016.

[19] Daniel Bernau, Philip-William Grassal, Jonas Robl, and Florian Kerschbaum. Assessing differentially private deep learning with membership inference. *arXiv preprint arXiv:1912.11328*, 2019.

[20] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 470–491. Springer, 2015.

[21] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[22] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[23] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[24] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 563–594. Springer, 2015.

[25] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[26] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Cloud strife: mitigating the security risks of domain-validated certificates. In *NDSS*. Internet Society, 2018.

[27] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[28] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[29] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 486–497, 2007.

[30] Brent Carmer, Alex J Malozemoff, and Mariana Raykova. 5gen-c: multi-input functional encryption and program obfuscation for arithmetic circuits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 747–764, 2017.

[31] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[32] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable and efficient secure two-party computation for machine learning. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 496–511. IEEE, 2019.

[33] Melissa Chase and Sarah Meiklejohn. Transparency overlays and applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 168–179, 2016.

[34] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.

[35] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[36] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[37] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2060–2068. IEEE, 2018.

[38] Rui Chen, Noman Mohammed, Benjamin CM Fung, Bipin C Desai, and Li Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[39] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

[40] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[41] Diego Chialva and Ann Dooms. Conditionals in homomorphic encryption and machine learning applications. *arXiv preprint arXiv:1810.12380*, 2018.

[42] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. *IACR Cryptology ePrint Archive*, 2020.

[43] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 703–732. Springer, 2018.

[44] Hiba Chougrad, Hamid Zouaki, and Omar Alheyane. Deep convolutional neural networks for breast cancer screening. *Computer methods and programs in biomedicine*, 157:19–30, 2018.

[45] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[46] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, pages 415–423. IEEE, 2015.

[47] Martine de Cock, Rafael Dowsley, Anderson CA Nascimento, and Stacey C Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2015.

[48] Open Edge Computing. Open edge computing initiative, 2017.

[49] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure multiparty computation*. Cambridge University Press, 2015.

[50] Jack LH Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. Doing real work with fhe: the case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–12, 2018.

[51] Ivan Damgård and Mads Jurik. A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[52] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[53] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[54] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[55] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[56] Mahir Can Doganay, Thomas B Pedersen, Yücel Saygin, Erkay Savaş, and Albert Levi. Distributed privacy preserving k-means clustering with additive secret sharing. In *Proceedings of the 2008 international workshop on Privacy and anonymity in information society*, pages 3–11, 2008.

[57] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800, 2013.

[58] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *European Symposium on Research in Computer Security*, pages 140–158. Springer, 2016.

[59] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[60] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *STOC*, volume 9, pages 371–380. ACM, 2009.

[61] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[62] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010.

[63] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, NIST, 2015.

[64] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate transparency with privacy. *Proceedings on Privacy Enhancing Technologies*, 2017(4):329–344, 2017.

[65] Liyue Fan. A survey of differentially private generative adversarial networks. In *The AAAI Workshop on Privacy-Preserving Artificial Intelligence*, 2020.

[66] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[67] Arik Friedman, Ran Wolff, and Assaf Schuster. Providing k-anonymity in data mining. *The VLDB Journal*, 17(4):789–804, 2008.

[68] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[69] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Secure linear regression on vertically partitioned datasets. *IACR Cryptology ePrint Archive*, 2016:892, 2016.

[70] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[71] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

[72] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[73] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[74] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 578–602. Springer, 2014.

[75] Francisco-Javier González-Serrano, Adrián Amor-Martín, and Jorge Casamayón-Antón. Supervised machine learning using encrypted training data. *International Journal of Information Security*, 17(4):365–377, 2018.

[76] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[77] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[78] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[79] Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.

[80] Rob Hall, Stephen E Fienberg, and Yuval Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27(4):669, 2011.

[81] Neil M Haller. The s/key (tm) one-time password system. In *Symposium on Network and Distributed System Security*, pages 151–157, 1994.

[82] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 2019.

[83] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[84] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1220–1237. IEEE, 2019.

[85] Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.

[86] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[87] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Deep neural networks classification over encrypted data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2019.

[88] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[89] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. Cryptology ePrint Archive, Report 2019/723, 2019. `https://eprint.iacr.org/2019/723`.

[90] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. Memguard: Defending against black-box membership inference attacks via adversarial examples. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–274, 2019.

[91] Kaifeng Jiang, Dongxu Shao, Stéphane Bressan, Thomas Kister, and Kian-Lee Tan. Publishing trajectories with differential privacy guarantees. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2013.

[92] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222. ACM, 2018.

[93] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[94] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *The International Conference on Learning Representations (ICLR)*, 2019.

[95] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. In *Advances in neural information processing systems*, pages 2879–2887, 2014.

[96] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. Random-data perturbation techniques and privacy-preserving data mining. *Knowledge and Information Systems*, 7(4):387–414, 2005.

[97] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Annual International Cryptology Conference*, pages 365–391. Springer, 2018.

[98] Daniel Kifer and Johannes Gehrke. Injecting utility into anonymized datasets. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 217–228, 2006.

[99] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[100] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[101] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. Tracking certificate misissuance in the wild. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 785–798. IEEE, 2018.

[102] Ben Laurie. Certificate transparency. *Queue*, 12(8):10, 2014.

[103] Ben Laurie and Emilia Kasper. Revocation transparency. *Google Research*, 2012.

[104] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, IETF, 2013.

[105] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[106] Yann LeCun, Corinna Cortes, and Burges Christopher J.C. MNIST handwritten digit database. *The MNIST Database*, 2010.

[107] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Workload-aware anonymization. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–286, 2006.

[108] Kevin Lewi, Alex J Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 981–992, 2016.

[109] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. *arXiv preprint arXiv:1909.05830*, 2019.

[110] Jiacheng Li, Ninghui Li, and Bruno Ribeiro. Membership inference attacks and defenses in supervised learning via generalization gap. *arXiv preprint arXiv:2002.12062*, 2020.

[111] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[112] Ping Li, Jin Li, Zhengan Huang, Chong-Zhi Gao, Wen-Bin Chen, and Kai Chen. Privacy-preserving outsourced classification in cloud computing. *Cluster Computing*, 21(1):277–286, 2018.

[113] Tian Li, Zaoxing Liu, Vyas Sekar, and Virginia Smith. Privacy for free: Communication-efficient learning with differential privacy using sketches. *arXiv preprint arXiv:1911.00972*, 2019.

[114] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54. Springer, 2000.

[115] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[116] Zaoxing Liu, Tian Li, Virginia Smith, and Vyas Sekar. Enhancing the privacy of federated learning with sketching. *arXiv preprint arXiv:1911.01812*, 2019.

[117] Qian Lou, Bo Feng, Geoffrey C Fox, and Lei Jiang. Glyph: Fast and accurately training deep neural networks on encrypted data. *arXiv preprint arXiv:1911.07101*, 2019.

[118] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[119] Paulo Martins, Leonel Sousa, and Artur Mariano. A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–33, 2017.

[120] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[121] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[122] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. In *USENIX Security*, pages 383–398, 2015.

[123] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. *arXiv preprint arXiv:1508.06110*, 2015.

[124] Redmond Microsoft Research. Microsoft SEAL (release 3.5), 2020.

[125] Azalia Mirhoseini, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Cryptoml: Secure outsourcing of big data machine learning applications. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pages 149–154. IEEE, 2016.

[126] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[127] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 591–602, 2015.

[128] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 38th IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[129] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[130] Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 634–646, 2018.

[131] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, ACM, 2019.

[132] Yurii Nesterov. Introductory lectures on convex programming volume i: Basic course. *Lecture notes*, 3(4):5, 1998.

[133] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE, 2013.

[134] Richard Nock, Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Entity resolution and federated learning get a federated resolution. *arXiv preprint arXiv:1803.04035*, 2018.

[135] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[136] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[137] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[138] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with pate. In *Proceedings of the 2018 Sixth International Conference on Learning Representations*, 2018.

[139] Heejin Park, Pyung Kim, Heeyoul Kim, Ki-Woong Park, and Younho Lee. Efficient machine learning over encrypted data with non-interactive communication. *Computer Standards & Interfaces*, 58:87–108, 2018.

[140] Cunchao Peng, Dapeng Li, Feng Tian, and Yongan Guo. Renewable energy powered iot data traffic aggregation for edge computing. In *International Conference in Communications, Signal Processing, and Systems*, pages 861–869. Springer, 2018.

[141] Martin Pettai and Peeter Laud. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 421–430. ACM, ACM, 2015.

[142] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 250–267. Springer, 2009.

[143] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[144] Jianwei Qian, Xiang-Yang Li, Chunhong Zhang, and Linlin Chen. De-anonymizing social networks and inferring private attributes using knowledge graphs. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[145] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 192–203. ACM, 2016.

[146] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Transactions on Data Privacy*, 11(1):61–79, 2018.

[147] Iyad Rahwan, Manuel Cebrian, Nick Obradovich, Josh Bongard, Jean-François Bonnefon, Cynthia Breazeal, Jacob W Crandall, Nicholas A Christakis, Iain D Couzin, Matthew O Jackson, et al. Machine behaviour. *Nature*, 568(7753):477–486, 2019.

[148] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 735–746, 2010.

[149] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[150] Pierangelo Rosati, Peter Deeney, Mark Cummins, Lisa Van der Werff, and Theo Lynn. Social media and stock price reaction to data breach announcements: Evidence from us listed companies. *Research in International Business and Finance*, 47:458–469, 2019.

[151] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[152] Mark Dermot Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *NDSS*. Internet Society, 2014.

[153] Théo Ryffel, Edouard Dufour Sans, Romain Gay, Francis Bach, and David Pointcheval. Partially encrypted machine learning using functional encryption. *arXiv preprint arXiv:1905.10214*, 2019.

[154] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. In *Proceedings of Privacy Preserving Machine Learning Workshop with NeurIPS 2018*, 2018.

[155] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. Distributed federated learning for ultra-reliable low-latency vehicular communications. *IEEE Transactions on Communications*, 68(2):1146–1159, 2019.

[156] Ashish P Sanil, Alan F Karr, Xiaodong Lin, and Jerome P Reiter. Privacy preserving regression modelling via distributed computation. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 677–682, 2004.

[157] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[158] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, Dave Choffnes, Alan Mislove, et al. A first look at certification authority authorization (caa). *ACM SIGCOMM Computer Communication Review*, 48(2):10–23, 2018.

[159] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. A novel error-tolerant anonymous linking code. *German Record Linkage Center, Working Paper Series No. WP-GRLC-2011-02*, 2011.

[160] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[161] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, volume 20, pages 41–440, 1971.

[162] Jian Shen, Dengzhi Liu, Xiaofeng Chen, Jin Li, Neeraj Kumar, and Pandi Vijayakumar. Secure real-time traffic data aggregation with batch verification for vehicular cloud in vanets. *IEEE Transactions on Vehicular Technology*, 69(1):807–817, 2019.

[163] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS*, volume 2, pages 1–17. Citeseer, 2011.

[164] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[165] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[166] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[167] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[168] Linus Sjöström and Carl Nykvist. How certificate transparency impact the performance, 2017.

[169] Aleksandra B Slavkovic, Yuval Nardi, and Matthew M Tibbits. Secure logistic regression of horizontally and vertically partitioned distributed databases. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 723–728. IEEE, 2007.

[170] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

[171] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[172] Aleksei Triastcyn and Boi Faltings. Generating artificial data for private deep learning. *arXiv preprint arXiv:1803.03148*, 2018.

[173] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 1–11, 2019.

[174] Muhammad Usman, Mian Ahmad Jan, and Deepak Puthal. Paal: A framework based on authentication, aggregation, and local differential privacy for internet of multimedia things. *IEEE Internet of Things Journal*, 7(4):2501–2508, 2019.

[175] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[176] Naga Vemprala and Glenn Dietrich. A social network analysis (sna) study on data breach concerns over social media. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[177] Huaqun Wang, Zhiwei Wang, and Josep Domingo-Ferrer. Anonymous and secure aggregation scheme in fog-based public cloud computing. *Future Generation Computer Systems*, 78:712–719, 2018.

[178] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and S Yu Philip. Private model compression via knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1190–1197, 2019.

[179] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 21–37, 2017.

[180] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 39–56, 2017.

[181] Yue Wang, Cheng Si, and Xintao Wu. Regression model fitting under differential privacy and model inversion attack. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[182] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *Annual Cryptology Conference*, pages 678–697. Springer, 2015.

[183] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to deanonymize social network users. In *2010 IEEE Symposium on Security and Privacy*, pages 223–238. IEEE, 2010.

[184] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[185] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[186] Xi Wu, Matthew Fredrikson, Somesh Jha, and Jeffrey F Naughton. A methodology for formalizing model-inversion attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 355–370. IEEE, 2016.

[187] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[188] Jie Xu and Fei Wang. Federated learning for healthcare informatics. *arXiv preprint arXiv:1911.06270*, 2019.

[189] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 13–23, 2019.

[190] Runhua Xu and James Joshi. Trustworthy and transparent third party authority. *ACM Transactions on Internet Technology (TOIT)*, 2020.

[191] Runhua Xu, James Joshi, and Chao Li. Cryptonn:training neural networks over encrypted data. In *2019 39th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1199–1209. IEEE, 2019.

[192] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[193] Hwanjo Yu, Jaideep Vaidya, and Xiaoqian Jiang. Privacy-preserving svm classification on vertically partitioned data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 647–656. Springer, 2006.

[194] Jiangshan Yu, Mark Ryan, and Cas Cremers. Decim: Detecting endpoint compromise in messaging. *IEEE Transactions on Information Forensics and Security*, 13(1):106–118, 2017.

[195] Jiale Zhang, Yanchao Zhao, Jie Wu, and Bing Chen. Lvpda: A lightweight and verifiable privacy-preserving data aggregation scheme for edge-enabled iot. *IEEE Internet of Things Journal*, 7(5):4016–4027, 2020.

[196] Liehuang Zhu, Meng Li, Zijian Zhang, Chang Xu, Ruonan Zhang, Xiaojiang Du, and Nadra Guizani. Privacy-preserving authentication and data aggregation for fog-based smart grid. *IEEE Communications Magazine*, 57(6):80–85, 2019.