DVFL: Decentralized Privacy-Preserving Vertical Federated Learning using Functional Encryption

1st Peng Long, 2nd Xiaofen Wang, 3rd Yongqiang Wang
University of Electronic Science and Technology of China
Chengdu, China
xfwang@uestc.edu.cn

4th Bo Zhang, 5th Xiaosong Ding
China Telecom Sichuan Branch
Chengdu, China
6th Lei Zheng
China Mobile Internet Co., LTD
Guangzhou, China

***Abstract*—Federated learning, as a privacy-preserving distributed machine learning approach, has been proposed to enable multiple participants to collaboratively train a model without sharing their original training data. Most existing studies have focused on privacy protection for horizontal federated learning, while there is a significant demand for vertical federated learning in real-world scenarios. Under the premise of privacy protection, current vertical federated learning (VFL) methods struggle to balance model accuracy, training time, and communication overhead. Most of these methods either sacrifice model accuracy for higher training efficiency or compromise training time and communication overhead for higher model accuracy. The emerging functional encryption mechanism can better balance these two aspects while maintaining privacy preserving, but in many scenarios, it relies on the support of a completely trusted third-party authority (TPA). To address these issues, we propose DVFL, a vertical federated learning framework based on decentralized multi-client functional encryption. This framework operates without the need of TPA support, utilizes gradient descent as the model optimization method, and is compatible with any machine learning or deep learning models that take linear layer as input layer. Our experimental evaluation demonstrate that, compared to state-of-the-art privacy preserving VFL methods, DVFL achieves a slight improvement in model accuracy while reducing training time by 20% to 71% and communication overhead by 34% to 56%.**

***Index Terms*—vertical federated learning, privacy-preserving, decentralized multi-client functional encryption.**

I. Introduction

Federated Learning [9] is an emerging distributed machine learning technology that enables multiple participants to collaboratively build and train machine learning models without directly sharing raw data, thereby effectively protecting data privacy. Depending on the overlap of samples held by each participant, Federated Learning (FL) is divided into Horizontal Federated Learning (HFL) [16] and Vertical Federated Learning (VFL) [11]. In the scenario of VFL, participants typically possess the same sample space but differ in feature space, meaning their data overlaps in the sample dimension and complement each other in the feature dimension. This characteristic of data distribution makes VFL highly suitable for scenarios requiring cross-institutional cooperation, such as medical diagnosis and financial services, where the data involved often contains sensitive information that should not be directly shared.

Compared to centralized machine learning, Vertical Federated Learning (VFL) retains raw data locally, transmitting only the intermediate results of model computations among participants, which reduces the risk of data exposure. However, as research delves deeper, it has been discovered that even within this distributed training framework, VFL systems are still subject to potential privacy breaches, such as model parameter reverse inference attacks [6] and gradient information leakage attacks [18]. Several privacy protection technologies have been proposed and applied to VFL, including differential privacy [5], gradient compression [10], [15], and homomorphic encryption [7], [8]. However, these methods either reduce model accuracy due to noise introduction or approximation, or increase communication and computational complexity. In response to these limitations, scholars have integrated Functional Encryption (FE) [1], an emerging privacy protection technology, into VFL. Unlike traditional privacy protection methods, FE not only allows the users to decrypt and obtain the complete plaintext information but also enables the users with specific decryption keys to access a function's value from encrypted data without revealing any additional information about the underlying plaintext.

However, the functional encryption technologies currently applied to VFL all require the support of an absolutely trustworthy Third Party Authority (TPA), which is difficult to achieve in real-world scenarios in terms of security requirements. For instance, FedV [13], the first FE-based VFL solution, requires TPA support and allows the aggregator to publish original model weights and calculate weight vector gradients, potentially enabling privacy inference. Moreover, due to the characteristics of functional encryption technology, when a decryptor decrypts the functional results of a batch of

encrypted data, they need to request a decryption key specific to that function from the TPA, which in turn has given rise to various inference attacks targeting FE in the VFL scenario [14] [13]. The decryptor, usually the aggregator, can request decryption keys for multiple functions to obtain the results of multiple function calculations, thereby inferring the privacy information of the ciphertext data from other participants based on these results.

To address this limitation, we propose DVFL, a Vertical Federated Learning architecture based on decentralized multi-client functional encryption. This architecture does not require a trusted third-party entity to participate in key distribution and decryption processes, making it more feasible in practical scenarios. Compared with FedV [13], our scheme does not necessitate the server to transmit model parameters of the interaction layer to the clients during each round of iterative training, thereby further enhancing the privacy of client-side data. Additionally, this framework supports any machine learning or deep learning models that use a linear layer as the input layer, as well as model optimization algorithms based on gradient descent, meaning that any nonlinear operations can be performed on the data following the input layer. Based on this framework, we have provided a detailed model training algorithm, including forward and backward propagation. This algorithm can effectively fend off adversaries' inference attacks on participants' data privacy information. Experiments demonstrate that our algorithm significantly reduces communication and computational overhead during model training without compromising model accuracy.

We summarize our main contributions as follows:

- We propose DVFL, a fully decentralized privacy-preserving Vertical Federated Learning framework. This framework is based on decentralized multi-client functional encryption, completely eliminating the need for an absolutely trustworthy Third Party Authority (TPA) in model training among participants. Moreover, clients only need one round of communication with each other during the system establishment, after which they only interact with the server, with no point-to-point communication between clients.
- DVFL supports any machine learning or deep learning models that use a linear layer as the model's input layer. DVFL treats the linear model input layer as the interaction layer between the server and clients, thereby providing broader model support and also reducing the complexity of computing model gradients.
- We have proposed a model training algorithm based on the DVFL framework that can protect the data privacy of participants and evaluated its performance on multiple datasets. The experimental results show that our scheme reduces training time by 20% to 71% and data transmission costs by 34% to 56% compared to FedV [13] without affecting model accuracy.

II. Preliminaries

A. Vertical Federated Learning

Vertical federated learning is a special paradigm of federated learning, suitable for scenarios where multiple data providers have the same user group but different feature information. In vertical federated learning, there are two entities: one server and multiple clients. Only the server has sample labels, while clients have sample data without labels. The computational process of the model in the vertical federated learning paradigm is as follows: each client first aligns their local dataset [8], then calculates their local model output, and transmits the output data to the server; the server takes the model outputs received from each client as inputs for its model and computes the final model output.

B. Decentralized Multi-Client Functional Encryption

Decentralized Multi-Client Functional Encryption (DMCFE) is an advanced encryption model that extends traditional Functional Encryption (FE). It allows multiple clients to independently encrypt data and perform specific function calculations on this data without relying on any trusted third party, without disclosing any unnecessary input information. In the DMCFE model, each client can control his own encrypted data and authorize the generation of functional decryption keys. This means that clients can ensure the security and privacy of their data even without a central authority. This decentralized feature not only reduces the risk of single points of failure but also enhances the privacy protection capabilities of the entire system.

The decentralized multi-client functional encryption on $\mathbf{M}$ among a set of $\mathbf{n}$ senders $(\mathbf{S}_i)_i$, for $i = 1, \ldots, n$, and a functional decrypter $\mathbf{FD}$ is defined in [3] by the setup protocol and four algorithms:

- **SetUp**(λ): A protocol among the senders $(S_i)_i$ that eventually generate their own secret keys sk_i and encryption keys ek_i, as well as the public parameters mpk;
- **Encrypt**(ek_i, x_i, l): Takes as input a user encryption key ek_i, a value x_i to encrypt, and a label l, and outputs the ciphertext $C_{l,i}$;
- **DKeyGenShare**(sk_i, l_f): Takes as input a user secret key sk_i and a label l_f, and outputs the partial functional decryption key $dk_{f,i}$ for a function $f : M^n \to R$ that is described in l_f;
- **DKeyComb**$((dk_{f,i})_i, l_f)$: Takes as input the partial functional decryption keys $dk_{f,i}$ $(i \in [1, n])$ and the label l_f, and eventually outputs the functional decryption key dk_f;
- **Decrypt**(dk_f, l, C): Takes as input a functional decryption key dk_f, a label l, and an n-vector ciphertext $C = \{C_{l,1}, \cdots, C_{l,n}\}$, and outputs $f(x)$ if C is a valid encryption of $x = (x_i)_i \in M^n$ for the label l, or $\perp$ otherwise.

C. Decisional Diffie-Hellman Assumption

The Decisional Diffie-Hellman Assumption states that, in a prime-order group $\mathbb{G} \xleftarrow{\$} \mathbf{GGen}(1^\lambda)$, no PPT adversary can

distinguish between the two following distributions with non-negligible advantage:

$$\left\{([a],[r],[ar]) \mid a, r \xleftarrow{\$} \mathbb{Z}_p\right\} \text{ and } \left\{([a],[r],[s]) \mid a, r, s \xleftarrow{\$} \mathbb{Z}_p\right\}$$

III. The Proposed DVFL Framework

A. System Model

As shown in Fig 1, the DVFL system consists of $\mathbf{M}$ participants $\mathbf{P} = \{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_M\}$, one of which also acts as the server, which we assume to be $\mathbf{P}_1$. Correspondingly, we refer to the other participants as clients. All of these participants possess $\mathbf{N}$ identical samples, but the sample features owned by each participant are disjoint, and only the server has the sample labels.

1) Server: Before model training, the server collaborates with all clients to initialize the system, which involves two steps: (1) First, the server generates the public parameters for the DMCFE algorithm and sends them to all clients, then collaborates with all clients to negotiate a unique mask vector for each. After receiving the partial functional decryption keys from each client based on this mask vector, it aggregates them to derive the functional decryption key. (2) Next, the server initializes the global model and vertically partitions the weight of the input layer of the global model according to the sample dimensions of each participant, assigning them to each participant including all clients and the server itself as the weights of their local models. The remaining part of the model is referred to as the server model. During each round of model training, the server aggregates the output results of all participants' local models as the input for the server model for subsequent model calculations, then uses the chain rule to calculate the gradients of the server model and the output values of each local model, and updates the server model parameters. Upon receiving the masked samples from the clients, it computes the gradients of each client's local model with the mask included and sends them back to the clients.

2) Client: Clients are synchronized in time. Before model training, clients also go through a two-step initialization process: (1) the clients secretly negotiate with other participants, including the server, to determine their own random masking vectors such that the sum of all participants' random masking vectors equals zero, and then calculate the partial decryption key based on this masking vector and send it to the server. (2) Next, the clients receive the input layer model weights from the server and initialize their local partial models with them. During each round of model training, clients first calculate the output values of their local partial models and send the ciphertexts of these outputs encrypted with DMCFE, along with their labels and masked samples, to the server. After receiving the masked local partial model gradients from the server, clients remove the gradient masks and use these gradients to update their local partial models.

As shown in algorithm 1, each round of the model's iterative training process is divided into two steps: forward propagation and backward propagation. Forward propagation is the process

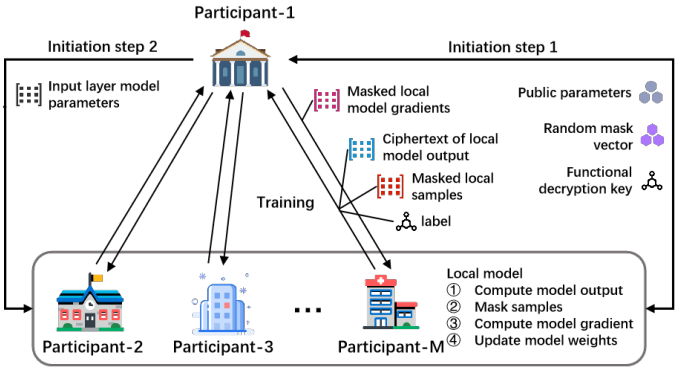

Fig. 1. DVFL Framework

from inputting sample data to calculating the loss value in the model, while backward propagation is the process of using the chain rule to retroactively calculate the gradient values of all model parameters from the model output value gradients and update the model parameters. In summary, in our framework, the training process of the vertical federated learning model is the process where the server, in conjunction with the clients, continuously performs forward propagation and backward propagation until the model converges.

In our proposal, each participant $\mathbf{P}_m$ ($m \in [1, M]$) has a dataset $\mathbf{D}_m$ and a unique feature set $\{\mathbf{f}_{m,1}, \mathbf{f}_{m,2}, \ldots, \mathbf{f}_{m,d_m}\}$, where the dataset $\mathbf{D}_m$ contains N identical samples. The goal is to train a model $\mathbf{F} = \mathbf{f}_0(\theta_0; \mathbf{f}_{n,1}, \ldots, \mathbf{f}_{n,M})$, where $\mathbf{f}_{n,m} = \mathbf{f}_m(\theta_m; \mathbf{x}_{n,m})$, for $m \in \{1, \ldots, M\}$, $n \in \{1, \ldots, N\}$, and $\mathbf{f}_m(\theta_m; \mathbf{x}_{n,m})$ represents the local model of the m-th client, $\mathbf{f}_{n,m}$ is the output result of the m-th client's model, $\mathbf{f}_0$ represents the server's model, and $\theta_0, \ldots, \theta_m$ represent their respective model parameters.

In the privacy-preserving vertical federated learning scenario, it is necessary to ensure the privacy of the datasets of each client, hence we need to protect the privacy of the data exchanged between the clients and the server to prevent adversaries from conducting inference attacks. In practical application scenarios, the Private Entity Resolution method [8] used to align the data of each client before model training is already more than sufficient. Therefore, in our proposal it is assumed that the datasets of each client have already been aligned.

B. Thread Model and Assumptions

The objective of this framework is to enable multiple participants to collaboratively train a machine learning or deep learning model while ensuring the privacy of each participant's data during the model training process. The adversary's goal is to infer the private data of the participants.

Here, we elucidate the assumptions made about the various participants in this system. Firstly, we assume that the server is semi-honest, meaning it will adhere to the protocol and correctly execute the computational process of the algorithm, but it may attempt to obtain private information from other participants; secondly, we assume there are no more than $M -$

2 semi-honest participants who correctly follow the protocol but may eavesdrop the messages sent from other clients to the server or collude with other malicious participants; finally, we assume that the computational results transmitted between the server and other participants are accurate and trustworthy, and there is no denial of service attacks and man-in-the-middle attack or backdoor attack that would degrade the performance of the model, as this exceeds the scope of this paper.

Algorithm 1: DVFL Framework

Input : security parameter(λ), total participants(M), batch size(bn), total batches per epoch(B), MaxEpochs
Output: global model F

```
system_initiation(λ, M);
foreach epoch in MaxEpochs do
    [B] ← shuffle {1, 2, ..., B};
    foreach b in [B] do
        select a batch data [{X_{b,m}}_{m∈[1,M]}, Y_b];
        forward_propagation({X_{b,m}}_{m∈[1,M]}, Y_b);
        backward_propagation();
    end
end
return F
```

IV. Training Process in DVFL

A. System Initiation

The server initially employs a polynomial-time pairing group generator **PGGen** to create a pairing group for the decentralized functional encryption, generating an asymmetric pairing group $\mathbf{PG} = (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e)$ with the security parameter λ. Here, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbf{G}_T$ are cyclic groups of prime order p with 2λ-bits, P_1 and P_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and e is an efficient bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. The server also selects a full-domain hash function H_1 that maps to $\mathbb{G}_1^2$, resulting in the master public key $mpk = (\mathbf{PG}, H_1)$. The server then initializes the model and divides the input layer weights $\mathbf{W}$ vertically according to the data dimensions of all participants, including itself, into $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_M\}$, and assigns $\mathbf{W}_m$ to participant $\mathbf{P}_m$, while sharing the public parameters with all clients. Lastly, the server establishes a small-scope dictionary dic on $\mathbb{G}_T$ to map group elements to actual values, facilitating the efficient resolution of the discrete logarithm problem during decryption.

Each participant $\mathbf{P}_m$, where $m \in \{1, \ldots, M\}$, initializes their local model with the input layer weights assigned by the server, and uniformly and randomly selects two random numbers $\{\mathbf{s}_{m,1}, \mathbf{s}_{m,2}\} \leftarrow \mathbb{Z}_p^2$, denoted as:

$$\mathbf{s}_m = \left(\mathbf{s}_{m,1}, \mathbf{s}_{m,2}\right)^T,$$

obtaining the private key $sk_m = \mathbf{s}_m$. They then secretly negotiate with other participants to determine a random masking vector $\mathbf{T}_m \leftarrow \mathbb{Z}_p^2$ such that

$$\sum_{m \in [M]} \mathbf{T}_m = 0,$$

compute the partial decryption key

$$\mathrm{dk}_m = [\mathbf{s}_m + \mathbf{T}_m]_2 \in \mathbb{G}_2^2,$$

and send it to the server. Here, $[\cdot]_i$ denotes the mapping of $\cdot$ in group $\mathbb{G}_i$, and $[\mathbf{expr}]_i$ denotes the operation of **expr** in group $\mathbb{G}_i$. After receiving all partial decryption keys dk_m ($m \in \{1, \cdots, M\}$) from the participants, the server computes the function decryption key

$$\mathrm{dk} = \sum_{m=1}^{M} \mathrm{dk}_m.$$

At this point, the initialization process is completed. Note that the clients only need to interact once during the system initialization to generate the random masking vector $\mathbf{T}_m$. No further interaction among clients is needed during subsequent training processes.

Algorithm 2: System Initiation

Input : security parameter λ, participant number M

Server:

$\mathbf{PG} \leftarrow \mathbf{GGen}(1^\lambda)$;
$(\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) = \mathbf{PG}$;
select a hash function $H_1 : \mathbb{G}_1^2 \xleftarrow{\$} \{0,1\}^*$;
select $\mathbf{s}_1 \xleftarrow{\$} \mathbb{Z}_p^2, \mathbf{T}_1 \xleftarrow{\$} \mathbb{Z}_p^2$;
compute $\mathbf{dk}_1 \leftarrow [\mathbf{s}_1 + \mathbf{T}_1]_2 \in \mathbb{G}_2^2$;
init global model F;
vertically partition the input layer weights: $\mathbf{W} \rightarrow \{\mathbf{W}_1, \cdots, \mathbf{W}_M\}$;
for m *in* $2 : M$ **do**
    send $\mathbf{W}_m$, $\mathbf{PG}$ and H_1 to participant $\mathbf{P}_m$;
    receive $\mathbf{dk}_m$ from participant $\mathbf{P}_m$;
end
$\mathbf{dk} \leftarrow \sum_{m=1}^{M} \mathbf{dk}_m$;

Client:

select $\mathbf{s}_m \xleftarrow{\$} \mathbb{Z}_p^2, \mathbf{T}_m \xleftarrow{\$} \mathbb{Z}_p^2$;
compute $\mathbf{dk}_m \leftarrow [\mathbf{s}_m + \mathbf{T}_m]_2 \in \mathbb{G}_2^2$;
init local model f_m with $\mathbf{W}_m$;
send $\mathbf{dk}_m$ to server.

B. Forward Propagation

1) Forward Propagation on Participants: Assuming the dataset is divided into B batches, with each batch containing b_n samples, participant $\mathbf{P}_m$ selects a batch of local private samples $X_{(b,m)}$, where $b \in \{1, ..., B\}$. Then, a random number r_m is uniformly and randomly chosen from $\{1, ..., p\}$ as a

mask, and the masked sample matrix $C_{\mathbf{X}_{b,m}}$ and the local partial model output matrix $f_{b,m}$ are calculated as:

$$C_{\mathbf{X}_{b,m}} = r_m \cdot \mathbf{X}_{b,m},$$

$$f_{b,m} = \mathbf{X}_{b,m}\mathbf{W}_m^T.$$

Subsequently, a timestamp is selected as the label l_m, and $\mathbf{u}_{l_m} = H_1(l_m) \in \mathbb{G}_1^2$ is computed. The ciphertext of matrix $f_{b,m}$ is obtained as:

$$C_{f_{b,m}} = [\mathbf{u}_{l_m}^T \mathbf{s}_m + f_{b,m}]_1 \in \mathbb{G}_1,$$

and the ciphertext of the model output value $C_{f_{b,m}}$, label l, and $C_{\mathbf{X}_{b,m}}$ are sent to the server. Here, $[\cdot]_i$ denotes the mapping of $\cdot$ in group $\mathbb{G}_i$, and $[\text{expr}]_i$ denotes the operation of expr in group $\mathbb{G}_i$. Note that during forward propagation, all participating parties select the same samples for computation, and their timing is synchronized, which means their selected labels are the same.

2) Forward Propagation on the Server: Assume the server's label is l. After receiving the ciphertexts of the partial model outputs and the labels from all the other participants, the server first checks if the labels sent from the clients are l. If any participant's label differs from l or if more than $M-1$ labels are received, indicating an error occurs in the received data, the server aborts the current training round. If all clients' labels are l, the server selects the same sample batch $\mathbf{X}_{b,1}$ and computes the ciphertext of its partial model output as $C_{f_{b,1}}$. Then, the server calculates

$$C_{f_b} = \sum_{m=1}^{M} \left[e(C_{f_{b,m}}, [1]_2) - e(u_l^T, dk)\right]$$

to obtain the ciphertext of the aggregated results of the partial model outputs. Finally, the server indirectly solves the discrete logarithm through a dictionary query dic to get the server model's input f_b. The server then continues to compute the final global model output and the global model loss value using the sample labels.

C. Backward Propagation

1) Backward Propagation on Server: The server utilizes the chain rule to iteratively determine the gradients of the server model, which includes the gradients of the output values of the input layer, ∇_{f_b}, effectively capturing the gradients of the partial model outputs from each participant. Utilizing these gradients, the server updates its own model and calculates the gradients for its partial model to perform updates. Subsequently, for each participant $\mathbf{P}_m$ where $m \in \{2, \ldots, M\}$, the server computes the masked local model gradients by taking the transpose of ∇_{f_b} and multiplying it with the masked sample matrix $C_{\mathbf{X}_{b,m}}$, resulting in

$$C_{\nabla_{W_m}} = \nabla_{f_b}^T C_{\mathbf{X}_{(b,m)}}$$

These masked gradients are then transmitted back to the respective participant $\mathbf{P}_m$.

Algorithm 3: Forward Propagation

Input : batch samples $\{\mathbf{X}_{b,m}\}_{m\in[1,M]}$ with label $\mathbf{Y}_b$
Output: global model loss L

Client:

$r_m \xleftarrow{\$} \mathbb{N}_p^+$;
$C_{\mathbf{X}_{b,m}} \leftarrow r_m \cdot \mathbf{X}_{b,m}$;
$f_{b,m} \leftarrow \mathbf{X}_{b,m}\mathbf{W}_m^T$;
$l_m \leftarrow$ current timestamp;
$\mathbf{u}_{l_m} \leftarrow H_1(l_m) \in \mathbb{G}_1^2$;
$C_{f_{b,m}} \leftarrow \left[\mathbf{u}_l^T \mathbf{s}_m + f_{b,m}\right]_1 \in \mathbb{G}_1$;
send $C_{\mathbf{X}_{b,m}}, C_{f_{b,m}}, l_m$ to server;

Server:

$l_1 \leftarrow$ current timestamp;
for m *in 2 :* M **do**
  receive $C_{\mathbf{X}_{b,m}}$, $C_{f_{b,m}}$ and l_m from participant $\mathbf{P}_m$;
end
if *any* $l_m \neq l_1$ *or received more than* $M-1$ *labels* **then**
  return $\perp$;
end
$f_{b,1} \leftarrow \mathbf{X}_{b,1}\mathbf{W}_1^T$;
$\mathbf{u}_{l_1} \leftarrow H_1(l_1) \in \mathbb{G}_1^2$;
$C_{f_{b,1}} \leftarrow \left[\mathbf{u}_l^T \mathbf{s}_1 + f_{b,1}\right]_1 \in \mathbb{G}_1$;
$C_{f_b} \leftarrow \sum_{m=1}^{M} e(C_{f_{b,m}}, [1]_2) - e(u_l^T, d_k)$;
$f_b \leftarrow dic(C_{f_b})$;
$L \leftarrow LossFunction(ServerModel(f_b), \mathbf{Y}_b)$;
return L

2) Backward Propagation on participants: After receiving its masked local model gradient $C_{\nabla_{\mathbf{W}_m}}$, participant $\mathbf{P}_m$ computes its model gradient $\nabla_{\mathbf{W}_m} = \frac{C_{\nabla_{\mathbf{W}_m}}}{r_m \cdot b_n}$, where r_m is the random mask and b_n is the number of samples in each batch. The participant then uses this gradient $\nabla_{\mathbf{W}_m}$ to update its local model.

Algorithm 4: Backward Propagation

Server:

compute $\bigtriangledown \mathbf{W}_{server}$, $\bigtriangledown_{f_b}$ by chain rule from L;
update server model by $\bigtriangledown \mathbf{W}_{server}$;
for m *in 2 :* M **do**
  $C_{\bigtriangledown \mathbf{W}_m} \leftarrow \bigtriangledown_{f_b}^T C_{\mathbf{X}_{b,m}}$;
  send $C_{\bigtriangledown \mathbf{W}_m}$ to participant $\mathbf{P}_m$;
end

Client:

upon receiving $C_{\bigtriangledown \mathbf{W}_m}$ *from server*:
  $\bigtriangledown \mathbf{W}_m \leftarrow \frac{C_{\bigtriangledown \mathbf{W}_m}}{r_m \times bn}$;
  update local model by $\bigtriangledown \mathbf{W}_m$;

V. Security and Privacy Analysis

Our proposal aims to collaboratively train a machine learning or deep learning model while ensuring the privacy of each participant's data. Below, we will analyze the security of our solution from two different aspects: the security of the cryptographic scheme and the security of the DVFL framework.

A. Security of the Cryptographic Approach

Our proposal aims to collaboratively train a machine learning or deep learning model among all participating parties while ensuring the privacy of their data. In the following, we will analyze the security of the cryptographic scheme.

DMCFE is the foundational construction for the forward iteration algorithm in our VEL framework, hence the security of this cryptographic scheme is crucial to our proposal. Our cryptographic algorithm fully adopts the DMCFE scheme from [3]. According to the formal security proof provided in [3], the DMCFE encryption algorithm in our VFL framework possesses ciphertext indistinguishability and is resistant to adaptive attacks under the classical DDH assumption. To avoid redundancy, we will not discuss the correctness and security proof of DMCFE here, and the readers can refer to [3] for more detailed information.

B. Privacy of FL Framework

1) The correctness of server-side aggregation computing: The output values of each participant's local model are encrypted using the DMCFE encryption algorithm, and the server aggregates the computations entirely through the functional decryption algorithm of DMCFE. Based on the properties of DMCFE, only if the server faithfully executes the DMCFE decryption algorithm can it compute the correct aggregated results of the local model outputs from all participants. If the server receives labels from a participant that do not match the current round of training or more than $M-1$ labels, it indicates that a attacker outside the system has submitted incorrect data. The server will consider the data received in this round of training to be erroneous and abandon this round of training. Therefore, when an attacker outside the system does not know the correct labels or colludes with participants within the system, the server will resist any disruption to the aggregation computation.

2) Resistance to Inference Attack: Below, we will consider two types of inference attacks: a curious server's inference attack and a limited number of colluding participants' inference attack. During the forward and backward propagation of the model, semi-honest servers and participants may attempt to infer the output results of each participant's local model, private sample data, and the gradients of the local models.

(i) Considering the adversary's inference attack on the output results of other participants' local models. Since the output values of each client's local model are encrypted using the DMCFE encryption algorithm, due to its All-or-Nothing Encapsulation property, even if the server and some curious participants collude, they cannot directly infer the true values of the output of the honest participant's local model. Alternatively, the server might attempt to exploit the characteristics of conventional functional encryption to calculate the difference between the ciphertext aggregation results of M and $M-1$ participants to indirectly infer the true value of the remaining ciphertext. However, since it cannot compute the decryption key for functions involving fewer than M participants, this inference method is also infeasible.

(ii) Considering an adversary's inference attack on other participants' private sample data and local model gradients. Since the private sample data sent from each participant to the server is perturbed by masking, and the local model gradients sent back from the server to the participants are also perturbed by masking, only the specific participant knows the masking value corresponding to their masked local model gradients. Moreover, the masking values chosen by the participants in each round are random. Even if the server colludes with other curious participants, they cannot infer the masking values selected by other participants, and thus cannot infer the actual private sample data and local model gradients. Therefore, DVFL can effectively resist gradient leakage attacks.

(iii) Considering an adversary's inference attack on the local model parameters of other participants. Unlike FedV [13], during the training process of our scheme, the different parts of the input layer model parameters are held by different participants, and thus even if the server colludes with other curious participants, they can at most infer the aggregated results of the output values from the remaining honest participants, but cannot deduce the partial input layer model parameters held by the remaining honest participants based on the aggregated results. Therefore, DVFL can also resist model parameter reverse inference attacks.

VI. Performance Evaluation

A. Experimental Setup

Our proposal is an improvement based on FedV [13], hence to evaluate the performance of our scheme, we have used FedV as the experimental baseline for comparison, and like FedV [13], employed an image dataset MNIST [17] along with three datasets from the open-source UCI Machine Learning Repository [4], which are website phishing, landsat satellite, and optical recognition of handwritten digits (optdigits). In all experiments, each dataset was vertically and equally divided based on the number of participants involved in model training. Detailed information about each dataset is presented in Table I.

TABLE I
Detailed information of each dataset.

Datasets	Instances	Number of features	Number of classes
phishing	11055	30	2
satellite	5100	36	2
optdigits	1797	64	10
MNIST	70000	784	10

Our scheme is applicable to any machine learning or deep learning model that takes a linear layer as the input layer. To ensure that our experimental results are more generalizable,

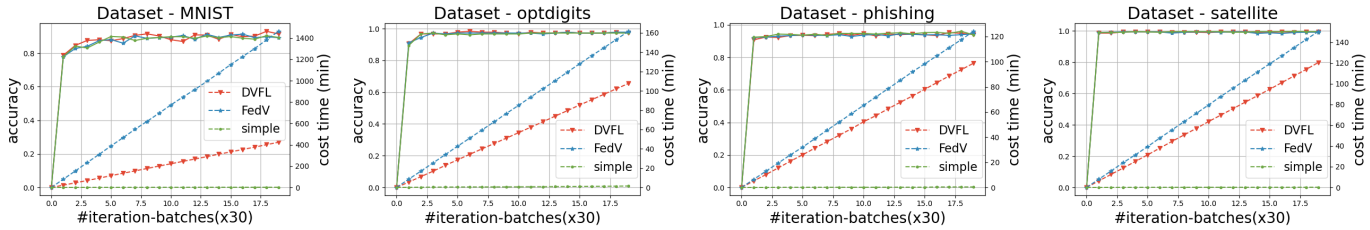

Fig. 2. The comparison of model accuracy and training time between DVFL, FedV, and simple with two participants.

our experiments were conducted using a Multilayer Perceptron (MLP) [12] classification model with three linear layers on the aforementioned datasets. The model training for the phishing and satellite datasets is a binary classification task, while the model training for the optdigits and MNIST datasets is a multi-class classification task. Therefore, we used the cross-entropy loss function during model training, and the final output of our model provides the probabilities of each class.

We implemented the MLP-based FedV scheme and our DVFL using Python's PyTorch framework, and realized the Multi-input functional encryption algorithm [2] and single-input functional encryption algorithm [1] of FedV's underlying layer with C++'s Cryptopp library, and the DMCFE algorithm from [3] with C's PBC library. We then compiled the cryptographic schemes implemented in C and C++ into Python modules for use during federated learning model training. Since the computational results of the aforementioned encryption schemes are within a small range, to address the issue of excessive computational overhead in solving discrete logarithm problems, we established a dictionary mapping from the ciphertext space to the plaintext space within a small range during the initialization of the cryptographic scheme. This transforms the discrete logarithm problem into a dictionary lookup with almost constant computational complexity. As the numerical computations in the cryptographic algorithms only support integers, we retained only four decimal places of precision, multiplied the decimals by 10^4, and then input them into the cryptographic algorithms for computation. Afterward, we divided the results by 10^4 before placing them back into the federated learning model for computation.

Experimental Environment. All experiments were conducted on a platform equipped with an Intel(R) Core(TM) i5-13400F CPU, which has 10 cores and 32GB of RAM. Please note that both FedV and our scheme were run on the same machine using a multi-process approach, where each process represents a participant, hence in our experiments, we did not measure network latency.

B. Experimental Results

We implemented the FedV framework, our DVFL framework, and a non-encrypted scheme based on the DVFL framework, which we call "simple", across the four datasets mentioned earlier. To maintain fairness, the same hyperparameter settings were used for training on each dataset for all three schemes, with each batch consisting of 64 samples and sampling recorded after every 30 batches.

Figure 2 illustrates the performance of each scheme in terms of model testing accuracy and training time expenditure during the first 20 samplings across different datasets. The results indicate that our scheme and the non-encrypted simple scheme have comparable efficiency in improving model accuracy across the four datasets. However, our scheme significantly reduced training time by approximately 20% to 71% compared to FedV. For instance, DVFL reduced training time by about 71% on the MNIST dataset and by about 20% on the phishing and satellite datasets. The varying degrees of training time reduction across different datasets are due to the differing sizes of training samples in each dataset. Additionally, comparing DVFL with the non-encrypted simple scheme in Table II leads to the conclusion that the impact on the final model accuracy from the encryption algorithm's truncation of the intermediate values of the input layer model calculations to four decimal places is negligible. The improvement in model accuracy of DVFL over FedV across the four datasets is expected, as FedV calculates model gradients based on encrypted samples during backpropagation, which results in numerical precision loss.

TABLE II
When the number of participants is 2, the final model accuracy of Simple, DVFL, and FedV on four datasets.

	MNIST	**optdigits**	**phishing**	**satellite**
simple	0.9461	0.9944	0.9629	0.9980
DVFL	0.9496	0.9889	0.9629	0.9961
FedV	0.9378	0.9722	0.9575	0.9941

Figure 3 contrasts the communication costs of FedV and DVFL after 20 batches of training on four datasets. The results indicate that DVFL's communication costs are substantially lower than those of FedV across all datasets, with a reduction ranging from 34% to 56%. This reduction is attributed to DVFL's lack of need to send encrypted sample data to the server, as each encrypted data unit is 256 bits in size. The communication cost on the MNIST dataset is notably higher than on the other three datasets due to its sample dimension reaching 784, while the dimensions of samples in the other datasets are all less than 100.

VII. Conclusion

Some existing privacy-preserving vertical federated learning schemes that utilize functional encryption are only applicable in situations supported by Trusted Third Parties (TPA), and their secure computation efficiency is not high. Moreover, in real-world scenarios, it is challenging to ensure the absolute

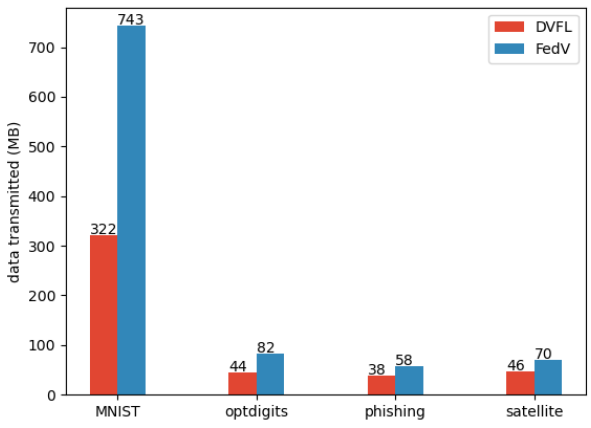

Fig. 3. The comparison of total communication costs between FedV and DVFL across different datasets after 20 batches of training.

trustworthiness required of TPAs. We propose DVFL, a fully decentralized vertical federated learning framework based on functional encryption that does not require support from any trustworthy third-party organizations. More importantly, this framework is applicable to any machine learning or deep learning models that use linear layers as the model's input layer, thus supporting a broader range of model training needs. Based on this framework, we have provided a detailed model training algorithm, and experiments show that, with a slight improvement in model accuracy, DVFL reduces training time by 20% to 71% and data transmission costs by 34% to 56% compared to TPA-based FedV.

REFERENCES

[1] 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.

[2] 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 *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*, pages 597–627. Springer, 2018.

[3] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II 24*, pages 703–732. Springer, 2018.

[4] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml, 2017.

[5] Fatima Zahra Errounda and Yan Liu. Adaptive differential privacy in vertical federated learning for mobility forecasting. *Future Generation Computer Systems*, 149:531–546, 2023.

[6] 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.

[7] Maoguo Gong, Yuanqiao Zhang, Yuan Gao, A Kai Qin, Yue Wu, Shanfeng Wang, and Yihong Zhang. A multi-modal vertical federated learning framework based on homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 2023.

[8] 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.

[9] Jakub Konečnỳ. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[10] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[11] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[12] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[13] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, James Joshi, and Heiko Ludwig. Fedv: Privacy-preserving federated learning over vertically partitioned data. In *Proceedings of the 14th ACM workshop on artificial intelligence and security*, pages 181–192, 2021.

[14] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, Swanand Kadhe, and Heiko Ludwig. Detrust-fl: Privacy-preserving federated learning in decentralized trust setting. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 417–426. IEEE, 2022.

[15] Kuihe Yang, Ziying Song, Yingchao Zhang, Yufan Zhou, Xiaohan Sun, and Jianxuan Wang. Model optimization method based on vertical federated learning. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[16] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[17] Yann LeCun and Corinna Cortes and CJ Burges. MNIST handwritten digit database. Technical Report 2, ATT Labs, [Online]. Available: http://yann.lecun.com/exdb/mnist, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist.

[18] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.