Quadratic Functional Encryption for Secure Training in Vertical Federated Learning

Shuangyi Chen[*], Anuja Modi[†], Shweta Agrawal[†], and Ashish Khisti[*]
[*]University of Toronto, {shuangyi.chen@mail.utoronto.ca, akhisti@ece.utoronto.ca}
[†]IIT Madras, {cs21d405@cse.iitm.ac.in, shweta.a@cse.iitm.ac.in}

***Abstract*—Vertical federated learning (VFL) enables the collaborative training of machine learning (ML) models in settings where the data is distributed amongst multiple parties who wish to protect the privacy of their individual data. Notably, in VFL, the labels are available to a single party and the complete feature set is formed only when data from all parties is combined. Recently, Xu et al. [1] proposed a new framework called *FedV* for secure gradient computation for VFL using multi-input functional encryption. In this work, we explain how some of the information leakage in Xu et al. can be avoided by using Quadratic functional encryption when training generalized linear models for vertical federated learning.**

I. Introduction

In many emerging applications, a machine learning (ML) model must be trained using private data that is distributed among multiple parties. We study the setting of *vertical federated learning* (VFL) where each individual party has access to a subset of features and labels and must cooperate to train a ML model that makes use of all the features. When privacy of user data is required, homomorphic encryption (HE) [2], [3], which enables the computation on encrypted data, provides a natural solution. In recent years, there has been a significant interest in HE based VFL systems, see e.g., [5]–[12]. Some works such as [7], [8], [11] consider a two-party protocol without the trusted coordinator, while others [9], [10]. consider a multi-party settings. Those frameworks require a large amount of peer-to-peer communications. References [5], [6] propose frameworks comprised of one trusted coordinator, storing the global weights, and two parties, each with a subset of vertically partitioned data. However, these frameworks require the trusted coordinator to share plaintext global weights with parties, which undermines the model's confidentiality.

In a recent work, Xu et. al. [1] proposed a generic and efficient privacy-preserving vertical Federated Learning (VFL) framework known as *FedV* in the multiparty setting. *FedV* makes use of *single-input function encryption* (SIFE) and *multi-input function encryption* (MIFE), and makes the communication between the clients and the aggregator a one-round interaction. However, *FedV* still have some key drawbacks. The protocol can reveal more information to the aggregator than just the final gradient in each iteration. Moreover, the protocol reveals the respective updated weights in each iteration to clients, which additionally creates leakage. For more details, please see Section IV.

A. Our Results.

We observe that the leakage created in *FedV* is caused by choosing an multi-input functional encryption (MIFE) scheme that only supports linear functions. Due to this, the weights are required to be provided to each party for inclusion in encryption, which creates unnecessary leakage. We observe that for linear models, this leakage can be prevented by using a more powerful MIFE scheme, namely MIFE for *quadratic functions* which can also be constructed using standard assumptions in cryptography [17], [18]. As our main contribution in this work, we demonstrate how such a function encryption scheme can be applied in VFL training by proposing a novel construction of function vectors that serve as a basis for generating decryption keys. Our approach leads to direct computation of the gradients, without leakage of any intermediate results as is the case with *FedV*. We discuss our proposed protocol, *SFedV*, for linear model training in Section IV and the extension to logistic regression model in Appendix C. We provide a thorough analysis of both security and efficiency in Section IV.

II. System Model

A. System Overview

Our system model involves three types of entities: aggregator, N clients, and Trusted Third Party (TTP). In the tth iteration for $t \in [T]$, each client holds a subset of features $\boldsymbol{X}_i^t \in \mathbb{R}^{S \times F_i}$ where F_i is the number of features that client party i holds and S is the batch size. A complete feature set of the current iteration is expressed as $\boldsymbol{X}^t = [\boldsymbol{X}_0^t \| \ldots \| \boldsymbol{X}_{N-1}^t] \in \mathbb{R}^{S \times F}$. One of the client parties holds the corresponding labels $\boldsymbol{y}^t \in \mathbb{R}^{S \times 1}$. The aggregator holds the entire model weights $\boldsymbol{w}^t = [\boldsymbol{w}_0^t \parallel \boldsymbol{w}_1^t \parallel \ldots \parallel \boldsymbol{w}_{N-1}^t] \in \mathbb{R}^{F \times 1}$ where $\boldsymbol{w}_i^t \in \mathbb{R}^{F_i \times 1}$ is the partial weights that pertains to $\boldsymbol{X}_i^t$. The aggregator is responsible for computing the model weights and the TTP is responsible for the generation of keys. In this work we focus on linear models of the form: $f(\boldsymbol{X}^t, \boldsymbol{w}^t) = \boldsymbol{X}^t \cdot \boldsymbol{w}^t$ with a squared-error loss function: $L(\boldsymbol{w}^t) = \frac{1}{S}||\boldsymbol{y}^t - \boldsymbol{X}^t \cdot \boldsymbol{w}^t||^2$. In our discussion, we will define the prediction error as:

$$\boldsymbol{u}^t = (\boldsymbol{y}^t - \boldsymbol{X}_0^t \cdot \boldsymbol{w}_0^t - \ldots - \boldsymbol{X}_{N-1}^t \cdot \boldsymbol{w}_{N-1}^t). \tag{1}$$

The gradient of $L(\boldsymbol{w}^t)$ with respect to $\boldsymbol{w}^t$ is expressed as

$$g(\boldsymbol{w}^t) = -\frac{2}{S}\begin{bmatrix} {\boldsymbol{y}^t}^\top \boldsymbol{X}_0^t - \sum_{i=0}^{N-1} {\boldsymbol{w}_i^t}^\top {\boldsymbol{X}_i^t}^\top \boldsymbol{X}_0^t \\ \ldots \| \\ {\boldsymbol{y}^t}^\top \boldsymbol{X}_{N-1}^t - \sum_{i=0}^{N-1} {\boldsymbol{w}_i^t}^\top {\boldsymbol{X}_i^t}^\top \boldsymbol{X}_{N-1}^t \end{bmatrix} \in \mathbb{R}^{1 \times F} \tag{2}$$

The gradient is used to update the global weights in each iteration according to $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \alpha g(\boldsymbol{w}^t)$ where α is the learning rate. We also discuss logistic regression model in Appendix C. In each iteration of the training phase, our protocol takes as input an encrypted copy of the features $\boldsymbol{X}_i^t$ and encrypted labels $\boldsymbol{y}^t \in \mathbb{R}^S$ from clients, and collaboratively and securely computes the gradients $g(\boldsymbol{w}^t)$.

Our threat model is defined as follows: we assume the aggregator is honest-but-curious meaning it correctly follows the algorithms and protocols but will try to infer clients' private data. Additionally, we assume that the aggregator does not collude with anyone. Similarly, the trusted third party is assumed not to collude with anyone. With respect to the clients, we assume that there are at most $N-1$ dishonest clients who may collude together and share their data to infer honest clients' information.

The protocol enables all the entities to collaboratively compute the gradient using vertically partitioned data. During the training process, we aim to achieve the following privacy requirements: 1) The client i and the aggregator should learn nothing about data $\boldsymbol{X}_j$ of client j for $i \neq j$. 2) Any client should learn nothing about the trained global model weights $\boldsymbol{w}$, intermediate results including the prediction error as in (1) and the gradient $g(\boldsymbol{w})$. Moreover, ith client should not learn anything about his/her own corresponding weights $\boldsymbol{w}_i$.

III. Preliminaries

A. Functional Encryption

Functional Encryption [14]–[16] is a public key encryption scheme that enables fine-grained access control over the encrypted data. In Single Input Functional Encryption(SIFE), the secret key is associated with a function f, and the ciphertext is associated with the vector $\boldsymbol{x}$. The decryption of ciphertext using the secret key outputs $f(\boldsymbol{x})$. Intuitively, the security says that the adversary learns nothing about the input $\boldsymbol{x}$ beyond what is revealed by $\{f_i(\boldsymbol{x})\}_i$ for any set of secret keys corresponding to the functions $\{f_i\}_i$ that the adversary holds.

B. Multi-Input Functional Encryption

Goldwasser et al. [13] generalized the functional encryption to support functions with multiple inputs. Multi-Input Functional Encryption, denoted as MIFE supports functions with arity greater than one. In MIFE, the secret key is associated with a function f, and the ith ciphertext is associated with the vector $\boldsymbol{x}_i$ for $i \in [N]$ where N is the arity of the function f. The decryption of all the ciphertexts using the secret key outputs $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. We now describe this notion in more detail.

Definition 1 (Multi-Input Functional Encryption (MIFE) [18]). **Syntax.** Let N be the number of encryption slots, and $\mathcal{F} = \{\mathcal{F}_N\}_{N\in\mathbb{N}}$ be a function family such that, for all $f \in \mathcal{F}_N$, $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_N \rightarrow \mathcal{Y}$. Here $\mathcal{X}_i$ and $\mathcal{Y}$ be the input and output spaces (respectively). A multi-input functional encryption (MIFE) scheme for function family $\mathcal{F}$ consists of the following algorithms.

$\mathsf{Setup}(1^\lambda, 1^N) \rightarrow (\mathsf{PP}, \{\mathsf{EK}_i\}_i, \mathsf{MSK})$**.** It takes a security parameter 1^λ, number of slots 1^N, and outputs a public parameter PP, N encryption keys $\{\mathsf{EK}_i\}_{i\in[N]}$ and a master secret key MSK. (The remaining algorithms implicitly take PP as input.)

$\mathsf{Enc}(\mathsf{EK}_i, \boldsymbol{x}) \rightarrow \mathsf{CT}_i$**.** It takes the ith encryption key EK_i and an input $\boldsymbol{x} \in \mathcal{X}_i$, and outputs a ciphertext CT_i.

$\mathsf{KeyGen}(\mathsf{MSK}, f) \rightarrow \mathsf{SK}$**.** It takes the master secret key MSK and function $f \in \mathcal{F}$ as inputs, and outputs a secret key SK.

$\mathsf{Dec}(\mathsf{CT}_1, \ldots, \mathsf{CT}_N, \mathsf{SK}) \rightarrow y$**.** It takes n ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_N$ and secret key SK, and outputs a decryption value $y \in \mathcal{Y}$ or a special abort symbol $\perp$.

Correctness. An MIFE scheme for the function family $\mathcal{F}$ is correct if for all $\lambda, N \in \mathbb{N}$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_N$, $f \in \mathcal{F}_N$, we have

$$\Pr\left[y = f(x_1, \ldots, x_N) : \begin{array}{l} (\mathsf{PP}, \{\mathsf{EK}_i\}_i, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ \{\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{EK}_i, \boldsymbol{x})\}_{i\in[N]} \\ \mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f) \\ y = \mathsf{Dec}(\mathsf{CT}_1, \ldots, \mathsf{CT}_N, \mathsf{SK}) \end{array} \right] = 1.$$

Security. Intuitively, security says that no information about the messages can be learned by the adversary except what is revealed by virtue of functionality – in more detail, an adversary possessing some ciphertexts and secret keys can perform decryption and learn the output of the functionality, which itself leaks something about the underlying plaintext. But besides this necessary leakage, the adversary does not learn anything. We provide the formal definition of security in Appendix A.

Multi-Input FE for Quadratic Functions. Agrawal, Goyal, and Tomida [18] constructed a multi-input functional encryption for quadratic functions (qMIFE). Let us define the N input quadratic function f as $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \langle \boldsymbol{c}, \boldsymbol{x} \otimes \boldsymbol{x} \rangle$ where $\boldsymbol{x} = (\boldsymbol{x}_1 || \ldots || \boldsymbol{x}_N)$. Here $\otimes$ denotes the Kronecker product. A n-input MIFE scheme for the function class $\mathcal{F}_{m,n}$ is defined as: each ith client encrypts $\boldsymbol{x}_i \in \mathbb{Z}^m$ using ith encryption key EK_i to get the ith ciphertext CT_i for $i \in [n]$. The KeyGen algorithm issues the secret key SK for $\boldsymbol{c} \in \mathbb{Z}^{(mn)^2}$ where $\boldsymbol{c}$ is the vector representation of the function $f \in \mathcal{F}_{m,n}$. The Dec algorithm uses the secret key SK to decrypt $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ to get $\langle \boldsymbol{c}, \boldsymbol{x} \otimes \boldsymbol{x} \rangle$ and nothing else.

C. FedV

As the system model of *SFedV* (Section II-A), *FedV* involves an aggregator, N clients, and a Trusted Third Party (TTP). Each client holds a subset of features $\boldsymbol{X}_i \in \mathbb{R}^{S\times F_i}$, and the first client also has the corresponding labels $\boldsymbol{y} \in \mathbb{R}^S$ along with its subset of features. The aggregator holds the complete model weights $\boldsymbol{w} = [\boldsymbol{w}_0 || \boldsymbol{w}_1 || \ldots || \boldsymbol{w}_{N-1}]$ where $\boldsymbol{w}_i \in \mathbb{R}^{F_i}$ is the partial weights that pertains to $\boldsymbol{X}_i$. In each iteration, there are two steps to compute the gradient

$$g(\boldsymbol{w}) = -\frac{2}{S}\left[\boldsymbol{u}^\top \boldsymbol{X}_0 \parallel \ldots \parallel \boldsymbol{u}^\top \boldsymbol{X}_{N-1}\right] \tag{3}$$

where $(\boldsymbol{u})^\top = (\boldsymbol{y} - \boldsymbol{X}_0\boldsymbol{w}_0 - \ldots - \boldsymbol{X}_{N-1}\boldsymbol{w}_{N-1})^\top$.

In the first step called Feature Dimension Secure Aggregation, *FedV* uses a Multi-Input Functional Encryption (MIFE) scheme for the inner product functionality [19], [20] to securely compute the prediction error $\boldsymbol{u}$. In this step, the aggregator sends each ith client the partial weights $\boldsymbol{w}_i$. Then each ith client encrypts each sample of $(-\boldsymbol{X}_i\boldsymbol{w}_i) \in \mathbb{R}^S$ and sends the ciphertext set $\mathsf{CT}^{\mathsf{MIFE}}_{-\boldsymbol{X}_i\boldsymbol{w}_i}$ to the aggregator. The first client, holding the label $\boldsymbol{y}$, encrypts each sample of $\boldsymbol{y}-\boldsymbol{X}_1\boldsymbol{w}_1$ and sends the ciphertext set $\mathsf{CT}^{\mathsf{MIFE}}_{\boldsymbol{y}-\boldsymbol{X}_1\boldsymbol{w}_1}$ to the aggregator. The aggregator asks the TTP for the secret key $\mathsf{SK}^{\mathsf{MIFE}}_{\boldsymbol{v}}$ corresponding to the fusion vector $\boldsymbol{v}$. This vector $\boldsymbol{v}$ can be a binary vector where one in ith position means that the aggregator has received ciphertext from client i. Using the secret key $\mathsf{SK}^{\mathsf{MIFE}}_{\boldsymbol{v}}$, the aggregator decrypts the ciphertexts $\{\{\mathsf{CT}^{\mathsf{MIFE}}_{-\boldsymbol{X}_i\boldsymbol{w}_i}\}_{i=1}^{N-1}, \mathsf{CT}^{\mathsf{MIFE}}_{\boldsymbol{y}-\boldsymbol{X}_1\boldsymbol{w}_1}\}$ to get the prediction error $\boldsymbol{u}$ (Equation (1)), which is the inner product of the fusion vector $\boldsymbol{v}$ and the partial predictions from clients.

In the second step called Sample Dimension Secure Aggregation, *FedV* uses Single-Input Functional Encryption (SIFE) scheme to compute the gradient $g(\boldsymbol{w})$. In this step, each client i encrypts each element of $\boldsymbol{X}_i$ and sends the ciphertext set $\mathsf{CT}^{\mathsf{SIFE}}_{\boldsymbol{X}_i}$ to the aggregator. On receiving the secret key $\mathsf{SK}^{\mathsf{SIFE}}_{\boldsymbol{u}}$ corresponding to the prediction error $\boldsymbol{u}$ from TTP, the aggregator decrypts the ciphertexts to get $\{\boldsymbol{u}^\top\boldsymbol{X}_i\}_{i\in[N]}$. The aggregator further processes the decryption results $\{\boldsymbol{u}^\top\boldsymbol{X}_i\}_{i\in[N]}$ according to Equation (3) to get the gradient $g(\boldsymbol{w})$. Using the gradients, the model weights are updated and then the training of the next epoch starts. Note the transmission of MIFE ciphertext ($\mathsf{CT}^{\mathsf{MIFE}}_{\boldsymbol{X}_i\boldsymbol{w}_i}$ or $\mathsf{CT}^{\mathsf{MIFE}}_{\boldsymbol{y}-\boldsymbol{X}_1\boldsymbol{w}_1}$) and SIFE ciphertext ($\mathsf{CT}^{\mathsf{SIFE}}_{\boldsymbol{X}_i}$) can be simultaneous. Thus the communication between each client and the aggregator is a one-round interaction in each iteration.

Leakage in FedV. While *FedV* preserves each client's data, it reveals the intermediate result, the prediction error $\boldsymbol{u}$ to the aggregator. Moreover, in the Feature Dimension Secure Aggregation step, the ith client is required to know the respective weight $\boldsymbol{w}_i$. Additionally, the aggregator can use the secret key of tth iteration to decrypt the ciphertext of some other iteration t' for $t' \neq t$ to infer more information about client data.

IV. The Protocol

Now we introduce our protocol with Multi-input Quadratic Functional Encryption qMIFE [18] as a privacy enhancement technology to do training in VFL setting.

At the beginning of the training phase, the aggregator initializes the global weights $\boldsymbol{w}^0$ and starts training. The training phase is iterative, where in the tth iteration, the TTP runs the $\mathsf{qMIFE.Setup}$ algorithm to get the public parameters PP^t, N encryption keys $\{\mathsf{EK}_i\}^t_{i\in[N]}$ and a master secret key MSK^t, then delivers the encryption key EK^t_i to the corresponding client i. After receiving the encryption key and determining the batch $\boldsymbol{X}^t$ used for training in this iteration, each client uses EK^t_i to encrypt $\boldsymbol{x}^t_i$ which is the vectorized $\boldsymbol{X}^t_i$ ($\mathsf{vec}(\cdot)$ stacks the columns of a matrix into a vector) to get ciphertext CT^t_i. Each client sends CT^t_i to the aggregator. The client that holds the labels encrypts $\boldsymbol{x}^t_i$ and $\boldsymbol{y}^t$ with EK^t_i to get ciphertexts CT^t_i and $\mathsf{CT}^t_{\boldsymbol{y}}$ respectively and then sends $(\mathsf{CT}^t_i, \mathsf{CT}^t_{\boldsymbol{y}})$ to the aggregator. At the same time, the aggregator computes a set of function vectors C^t according to the model weights $\boldsymbol{w}^t$ of the current iteration and sends them to TTP to generate a set of decryption keys. Detailed procedure is described in Section IV-A. Then, the aggregator decrypts all the ciphertexts $(\{\mathsf{CT}^t_i\}_{i=0}^{N-1}, \mathsf{CT}^t_y)$ that were received from clients using the secret keys received from TTP, to get each element of (2) respectively. By concatenating those elements and further processing the results, the aggregator gets the gradients. After this, it can update the global weights and start the training of the next iteration. Algorithm 1 shows the training procedure for linear models and also supports the training for logistic regression as discussed in Appendix C.

A. Construction of Function Vectors

Our goal is to compute the gradient $g(\boldsymbol{w})$ (Equation (2)). For simplicity, we drop the superscript t in our discussion. We define $\boldsymbol{x} = [\boldsymbol{x}_0||...||\boldsymbol{x}_{N-1}||\boldsymbol{y}]$ where $\boldsymbol{y}$ is the label vector and $\boldsymbol{x}_i$ is the vectorized $\boldsymbol{X}_i$. Recall $g(\boldsymbol{w})$ is a vector of length F, where F is the total number of features. The key insight is that for the $f^{\text{th}}, f \in [F]$ element in (2), we construct a function vector $\boldsymbol{c}_f$ based on weights $\boldsymbol{w}$ of current iteration, such that $g(\boldsymbol{w})[f] = -\frac{2}{S}\langle \boldsymbol{c}_f, \boldsymbol{x}\otimes\boldsymbol{x}\rangle$. Then the aggregator concatenates $g(\boldsymbol{w})[f], f \in [F]$ to obtain the gradients.

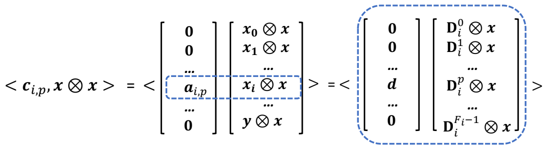
For simplicity, we define the following:

$$\boldsymbol{z}_i = \boldsymbol{u}^\top\boldsymbol{X}_i, \quad \boldsymbol{b}^i_j = \boldsymbol{w}_j^\top\boldsymbol{X}_j^\top\boldsymbol{X}_i, \quad \boldsymbol{b}^i_y = \boldsymbol{y}^\top\boldsymbol{X}_i. \tag{4}$$

Now we decompose $g(\boldsymbol{w})$ and $\boldsymbol{x}\otimes\boldsymbol{x}$ to reduce the assignment. Note that to compute $g(\boldsymbol{w})$ it suffices to compute $\boldsymbol{z}_0, ..., \boldsymbol{z}_{N-1}$ as in (3). Here we define a set of function vectors $C = \{C_i\}_{i=0}^{N-1}$ and $C_i = \{\boldsymbol{c}_{i,p}\}_{p=0}^{F_i-1}$, where C_i is a subset of function vectors that are used to compute elements in $\boldsymbol{z}_i$, $\boldsymbol{c}_{i,p}$ is the function vector to compute pth element of $\boldsymbol{z}_i$ as in (5).

$$\boldsymbol{z}_i[p] = \langle \boldsymbol{c}_{i,p}, \boldsymbol{x}\otimes\boldsymbol{x}\rangle \tag{5}$$

We construct $\boldsymbol{c}_{i,p}$ block by block according to the decomposition of $\boldsymbol{x}\otimes\boldsymbol{x}$. Consider dividing $\boldsymbol{x}\otimes\boldsymbol{x}$ into $N+1$ blocks as in the middle of Figure 1.



$< \boldsymbol{c}_{i,p}, \boldsymbol{x}\otimes\boldsymbol{x} > = <$
$\mathbf{0}$
$\mathbf{0}$
...
$\boldsymbol{a}_{i,p}$
...
$\mathbf{0}$
$\boldsymbol{x}_0\otimes\boldsymbol{x}$
$\boldsymbol{x}_1\otimes\boldsymbol{x}$
...
$\boldsymbol{x}_i\otimes\boldsymbol{x}$
...
$\boldsymbol{y}\otimes\boldsymbol{x}$
$> = <$
$\mathbf{0}$
$\mathbf{0}$
...
$\boldsymbol{d}$
...
$\mathbf{0}$
$\mathbf{D}_i^0\otimes\boldsymbol{x}$
$\mathbf{D}_i^1\otimes\boldsymbol{x}$
...
$\mathbf{D}_i^p\otimes\boldsymbol{x}$
...
$\mathbf{D}_i^{F_i-1}\otimes\boldsymbol{x}$
$>$

Fig. 1: Decomposition of $\langle \boldsymbol{c}_{i,p}, \boldsymbol{x}\otimes\boldsymbol{x}\rangle$

Since in the computation of $\boldsymbol{z}_i$, only the component $\boldsymbol{x}_i\otimes\boldsymbol{x}$ is required, we set $\mathbf{0}$ vector of the corresponding lengths as the coefficients of the blocks $\{\boldsymbol{x}_j\otimes\boldsymbol{x}\}_{j=0}^{N}$ if $j \neq i$. We design $\boldsymbol{a}_{i,p}$ to make $\boldsymbol{z}_i[p] = \langle \boldsymbol{a}_{i,p}, \boldsymbol{x}_i\otimes\boldsymbol{x}\rangle$.

Let D^f_i denotes the fth column of $\boldsymbol{X}_i$. Thus $\boldsymbol{x}_i = [\mathsf{D}^0_i; ...; \mathsf{D}^{F_i-1}_i]$. Note that in the computation of $\boldsymbol{z}_i[p]$ only the column D^p_i is used, thus we set $\mathbf{0}$ vector as the coefficients of $\{\mathsf{D}^q_i\otimes\boldsymbol{x}\}_{q=0}^{F_i-1}$ if $q \neq p$. Hence we can express $\langle \boldsymbol{a}_{i,p}, \boldsymbol{x}_i\otimes\boldsymbol{x}\rangle = \langle \boldsymbol{d}, \mathsf{D}^p_i\otimes\boldsymbol{x}\rangle$. We design $\boldsymbol{d}$ to achieve $\boldsymbol{z}_i[p] = \langle \boldsymbol{d}, \mathsf{D}^p_i\otimes\boldsymbol{x}\rangle$.

From (2), (3) and (4) note that we can express:

$$\boldsymbol{z}_i[p] = \sum_{j=0}^{N-1} -\boldsymbol{b}^i_j[p] + \boldsymbol{b}^i_y[p] \tag{6}$$

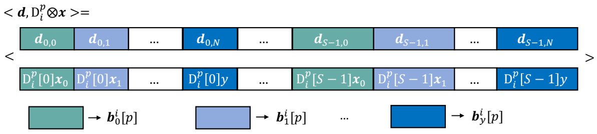
Algorithm 1 Training Procedure

```
procedure TRAINING-AGGREGATOR($\boldsymbol{w}^t, s, \{F_i\}_{i=0}^{N-1}, N$)
    $\boldsymbol{res} = \boldsymbol{0}^F$
    for each $i \in 0, ..., N-1$ do
        for each $p \in 0, ..., F_i - 1$ do
            $\boldsymbol{c}_{i,p}^t$ := CGEN($\boldsymbol{w}^t, S, F, N, i, p$)
        end for
    end for
    $C^t := \{\boldsymbol{c}_{i,p}^t, i \in [N], p \in [F_i]\}$
    $\{\{\mathsf{qMIFE.SK}_{c_{i,p}}^t\}_{p=0}^{F_i}\}_{i=0}^{N-1}$ = obtain-dk-from-TTP($C^t$)
    for each $i \in 0, ..., N-1$ do
        if party $i$ has label $\boldsymbol{y}^t$ then
            ($\mathsf{CT}_i^t, \mathsf{CT}_y^t$)= obtain-ct-from-client()
        else
            $\mathsf{CT}_i^t$ = obtain-ct-from-client()
        end if
    end for
    $\mathsf{CT}^t = \{\{\mathsf{CT}_i^t\}_{i=0}^{N-1}, \mathsf{CT}_y^t\}$
    for each $n \in 0, ..., N-1$ do
        for each $p \in 0, ..., F_n - 1$ do
            idx = $\sum_{i=0}^{n-1} F_i + p$
            $\boldsymbol{res}$[idx] = qMIFE.Dec($\mathsf{CT}^t$, $\mathsf{qMIFE.SK}_{c_{n,p}}^t$)
        end for
    end for
    $\nabla L(\boldsymbol{w}^t) = -\frac{2}{S}\boldsymbol{res} + \lambda \nabla R(\boldsymbol{w}^t)$
    $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \alpha \nabla L(\boldsymbol{w}^t)$
end procedure
procedure TRAINING-CLIENT($\boldsymbol{X}_i^t$)
    function OBTAIN-CT-FROM-CLIENT()
        $\mathsf{qMIFE.EK}_i^t$= obtain-ek-from-TTP()
        $\boldsymbol{x}_i^t$ := vec($\boldsymbol{X}_i^t$)
        if party $i$ has label $\boldsymbol{y}^t$ then
            $\mathsf{CT}_i^t$ := qMIFE.Enc($\mathsf{qMIFE.EK}_i^t, \boldsymbol{x}_i^t$)
            $\mathsf{CT}_y^t$ := qMIFE.Enc($\mathsf{qMIFE.EK}_i^t, \boldsymbol{y}^t$)
            Return ($\mathsf{CT}_i^t, \mathsf{CT}_y^t$) to Aggregator
        else
            $\mathsf{CT}_i^t$ := qMIFE.Enc($\mathsf{qMIFE.EK}_i^t, \boldsymbol{x}_i^t$)
            Return $\mathsf{CT}_i^t$ to Aggregator
        end if
    end function
end procedure
procedure TRAINING-TTP($1^\lambda, 1^N$)
    function OBTAIN-EK-FROM-TTP()
        ($\mathsf{PP}^t, \{\mathsf{EK}_i\}_i^t, \mathsf{MSK}^t$) ← qMIFE.Setup($1^\lambda, 1^N$)
        Deliver $\mathsf{qMIFE.EK}_i^t$ to party $i$, $i \in [N]$
    end function
    function OBTAIN-DK-FROM-TTP($C^t$)
        for each $i \in 0, ..., N-1$ do
            for each $p \in 0, ..., F_i$ do
                qMIFE.KeyGen($\mathsf{qMIFE.MSK}^t, \boldsymbol{c}_{i,p}^t$) → $\mathsf{qMIFE.SK}_{c_{i,p}}^t$
            end for
        end for
        Return $\{\{\mathsf{qMIFE.SK}_{c_{i,p}}^t\}_{p=0}^{F_i}\}_{i=0}^{N-1}$
    end function
end procedure
```

where we have $\boldsymbol{z}_i[p] = (\boldsymbol{u}^\top \boldsymbol{X}_i)[p]$, $\boldsymbol{b}_j^i[p] = (\boldsymbol{w}_j^\top \boldsymbol{X}_j^\top \boldsymbol{X}_i)[p]$,$\boldsymbol{b}_y^i[p] = (\boldsymbol{y}^\top \boldsymbol{X}_i)[p]$. We expand $\boldsymbol{b}_j^i[p]$ by first performing multiplication term-by-term and then summing products as in (7). Based on this equation, we determine the method for constructing the coefficients vector $\boldsymbol{d}$.

$$\begin{aligned}\boldsymbol{b}_j^i[p] &= (\boldsymbol{w}_j^\top \boldsymbol{X}_j^\top \boldsymbol{X}_i)[p] = (\boldsymbol{w}_j^\top \boldsymbol{X}_j^\top)\mathsf{D}_i^p \\ &= \left[\sum_{f=0}^{F_j-1} \boldsymbol{w}_j^\top[f]\mathsf{D}_i^f[0] \,\|\, ... \,\|\, \sum_{f=0}^{F_j-1} \boldsymbol{w}_j^\top[f]\mathsf{D}_i^f[S-1]\right]\mathsf{D}_i^p \\ &= \sum_{s=0}^{S-1}\sum_{f=0}^{F_j-1} \boldsymbol{w}_j[f]\mathsf{D}_j^f[s]\mathsf{D}_i^p[s] \qquad (7)\end{aligned}$$

Now the goal is to construct $\boldsymbol{d}$ such that $\boldsymbol{z}_i[p] = \langle \boldsymbol{d}, \mathsf{D}_i^p \otimes \boldsymbol{x} \rangle$. We keep decomposing $\boldsymbol{d}$ and $\mathsf{D}_i^p \otimes \boldsymbol{x}$ to blocks as in Figure 2.



$< \boldsymbol{d}, \mathrm{D}_i^p \otimes \boldsymbol{x} >=$
<
$\boldsymbol{d}_{0,0}$
$\boldsymbol{d}_{0,1}$
...
$\boldsymbol{d}_{0,N}$
...
$\boldsymbol{d}_{S-1,0}$
$\boldsymbol{d}_{S-1,1}$
...
$\boldsymbol{d}_{S-1,N}$
>
$\mathrm{D}_i^p[0]\boldsymbol{x}_0$
$\mathrm{D}_i^p[0]\boldsymbol{x}_1$
...
$\mathrm{D}_i^p[0]y$
...
$\mathrm{D}_i^p[S-1]\boldsymbol{x}_0$
$\mathrm{D}_i^p[S-1]\boldsymbol{x}_1$
...
$\mathrm{D}_i^p[S-1]y$
→ $\boldsymbol{b}_0^i[p]$
→ $\boldsymbol{b}_1^i[p]$
...
→ $\boldsymbol{b}_y^i[p]$

Fig. 2: Decomposition of $\langle \boldsymbol{d}, \mathsf{D}_i^p \otimes \boldsymbol{x} \rangle$

In Figure 2, blocks with the same color will be designed to compute the corresponding term on the right side of Equation (6). Considering $\boldsymbol{d}_{s,j}, s \in [S]$, we can set the following relations:

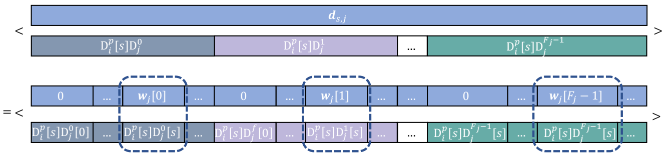
$$\boldsymbol{b}_j^i[p] = \sum_{s=0}^{S-1} \langle \boldsymbol{d}_{s,j}, \mathsf{D}_i^p[s]\boldsymbol{x}_j \rangle \qquad (8)$$

$$\boldsymbol{b}_y^i[p] = \sum_{s=0}^{S-1} \langle \boldsymbol{d}_{s,N}, \mathsf{D}_i^p[s]\boldsymbol{y} \rangle \qquad (9)$$

Next, we introduce the approach to construct $\boldsymbol{d}_{s,j}, j \in [N]$ according to the corresponding weight piece. We remove the outer summation in (7) and (8) to obtain:

$$\sum_{f=0}^{F_j-1} \boldsymbol{w}_j[f]\mathsf{D}_j^f[s]\mathsf{D}_i^p[s] = \langle \boldsymbol{d}_{s,j}, \mathsf{D}_i^p[s]\boldsymbol{x}_j \rangle \qquad (10)$$

We design $\boldsymbol{d}_{s,j}$ to achieve (10) as Figure 3 shows. We decompose $\langle \boldsymbol{d}_{s,j}, \mathsf{D}_i^p[s]\boldsymbol{x}_j \rangle$ into blocks $\mathsf{D}_i^p[s]\mathsf{D}_j^f, f \in [F_j]$. In each block $\mathsf{D}_i^p[s]\mathsf{D}_j^f$, we take one entry $\mathsf{D}_i^p[s]\mathsf{D}_j^f[s]$ and set its coefficient to $\boldsymbol{w}_j[f]$ just as the left side of (10). For other unneeded terms, we set the coefficient to $\boldsymbol{0}$.



<
$\boldsymbol{d}_{s,j}$
>
$\mathrm{D}_i^p[s]\mathrm{D}_j^0$
$\mathrm{D}_i^p[s]\mathrm{D}_j^1$
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^{Fj-1}$
= <
0
...
$\boldsymbol{w}_j[0]$
...
0
...
$\boldsymbol{w}_j[1]$
...
...
0
...
$\boldsymbol{w}_j[F_j-1]$
...
>
$\mathrm{D}_i^p[s]\mathrm{D}_j^0[0]$
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^0[s]$
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^f[0]$
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^1[s]$
...
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^{Fj-1}[s]$
...
$\mathrm{D}_i^p[s]\mathrm{D}_j^{Fj-1}[s]$
...

Fig. 3: Decomposition of $\langle \boldsymbol{d}_{s,j}, \mathsf{D}_i^p[s]\boldsymbol{x}_j \rangle$

The approach to construct $\boldsymbol{d}_{s,j}, j \in [N]$ can be easily extended to the case for $\boldsymbol{d}_{s,N}$. By designing the blocks of $\boldsymbol{d}$ this way, we can achieve $\boldsymbol{z}_i[p] = \langle \boldsymbol{d}, \mathsf{D}_i^p \otimes \boldsymbol{x} \rangle$. The algorithms to construct the function vectors $\boldsymbol{c}_{i,p}$ and d are provided in Appendix B.

B. Privacy Analysis

Recall the aim of our framework. We want the client i and the aggregator to learn nothing about data $\boldsymbol{X}_j$ of client j for $i \neq j$. We also want that any client should learn nothing about the trained global model weights $\boldsymbol{w}$, intermediate results including error between labels and feed-forward output as in (1) and the gradient $g(\boldsymbol{w})$. Moreover, ith client should not learn anything about his/her own corresponding weights $\boldsymbol{w}_i$. In this section, we prove that we have achieved the above-stated goal.

Theorem IV.1. *If Quadratic MIFE* (qMIFE) *is secure according to definition 2, then in each training iteration t, ith client's data $\boldsymbol{X}_i^t$ for $i \in [N]$ is hidden from client j and the aggregator, trained global model weights $\boldsymbol{w}^t$ and intermediate results $\boldsymbol{u}^t$ as in* (1) *are hidden from the clients and ith client learns nothing about weight $\boldsymbol{w}_i$.*

Let us fix the iteration number to be t. In each iteration, the TTP runs the qMIFE.Setup algorithm to get public parameters PP^t, N encryption keys $\{\mathsf{EK}_i\}_{i\in[N]}^t$ and a master secret key MSK^t. The clients encrypt their respective data and send the ciphertexts to the aggregator. The aggregator asks the TTP for the secret key corresponding to the set of vectors C^t. Each vector $\boldsymbol{c}_{i,p}^t$ is set in such a way that the qMIFE.Dec only reveals the inner product $\langle \boldsymbol{c}_{i,p}^t, \boldsymbol{x} \otimes \boldsymbol{x} \rangle = ((\boldsymbol{u}^t)^\top \boldsymbol{X}_i^t)[p]$. Quadratic MIFE ensures that nothing about $\boldsymbol{X}_i^t$ and $\boldsymbol{u}^t$ is revealed to the aggregator. Moreover, each client encrypts their data using different encryption keys. The ciphertexts are indistinguishable; hence, clients cannot predict other clients' data.

Unlike *FedV*, in our framework, the client runs the qMIFE.Enc algorithm which only takes their respective data and encryption keys as input. Hence, each client i learns nothing about their respective weight $\boldsymbol{w}_i^t$. Moreover, Quadratic MIFE ensures that client i learns nothing about the global weight $\boldsymbol{w}^t$. The aggregator does not share the gradients in any form with the clients. Therefore, the gradient $g(\boldsymbol{w}^t)$ is also not revealed.

Importance of using new qMIFE instance for each iteration. Let in the iteration t, the ciphertext be CT^t and secret key be $\mathsf{qMIFE.SK}^t$. Suppose the TTP uses the same MSK to generate secret keys $\mathsf{qMIFE.SK}^{t+1}$ for some other iteration, say $t+1$, then the aggregator may use $\mathsf{qMIFE.SK}^{t+1}$ to decrypt the ciphertext CT^t instead of using it to decrypt the ciphertext CT^{t+1}. Using this "mix-and-match" attack by performing decryptions of secret key and ciphertexts from different iterations, he will know $g(\boldsymbol{w}) = -\frac{2}{S}\left[(\boldsymbol{u}^{t+1})^\top \boldsymbol{X}_0^t || ... || (\boldsymbol{u}^{t+1})^\top \boldsymbol{X}_{N-1}^t\right]$.

If TTP generates different qMIFE instance for every iteration, then decryption of CT^t with secret key $\mathsf{qMIFE.SK}^{t+1}$ will give some garbage value which will be irrelevant for the aggregator. Therefore, it is important for the TTP to generate a new qMIFE instance for every iteration.

Comparison of our framework with FedV. Unlike *FedV*, our framework does not leak the intermediate result $\boldsymbol{u}$ to the aggregator. The global weights $\boldsymbol{w}$ are kept secret from the clients and each client also learns nothing about their respective weights. In addition to this, we also ensure that the aggregator cannot use the mix-and-match attack to learn some useful information.

C. Efficiency Analysis

Communication. Regarding communication complexity, *SFedV* requires one-way client-aggregator communication, while *FedV* needs one-round client-aggregator communication due to the delivery of global weights by the aggregator. Additionally, *SFedV* uses a new qMIFE instance in each iteration to prevent mix-and-match attacks. Thus, an increase of communication between TTP and clients becomes necessary. Note that *FedV* can also prevent mix-and-match attacks by using new instances of MIFE and SIFE in each iteration. In such a scenario, the client-TTP communication complexity for each iteration will be the same for both *FedV* and *SFedV*.

TABLE I: Comparison of *FedV* and *SFedV* regarding the number of encryption processes on each client and the number of decryption processes on the aggregator in each iteration.

	FedV		*SFedV*
	MIFE	SIFE	qMIFE
Encryptions on each client	S	$S \cdot F_i$	1
Decryptions on the aggregator	S	F	F

S: Batch size. F: Total number of features. F_i: Number of features belonging to client i.

Computation. Table I provides a comparison between *FedV* and *SFedV* in terms of the number of encryption and decryption processes in each iteration. The significant improvement of *SFedV* is attributed to the advancement of quadratic MIFE and the careful design of function vectors.

In terms of the number of the vector corresponding to which the secret keys are generated, *FedV* uses two vectors for two steps: $\boldsymbol{v}$ for feature dimension secure aggregation and $\boldsymbol{u}$ for sample dimension secure aggregation. In contrast, our *SFedV* framework employs F vectors $\boldsymbol{c}$, where F is the total number of features. The increase in size can be justified by our use of a quadratic MIFE scheme instead of inner product MIFE.

V. Conclusions

Prior N-party VFL framework *FedV* incurs information leakage which seriously undermines individual data privacy. In this work, to address the privacy issues, We propose a leak-free protocol, called *SFedV*, for multiparty VFL regression model training. Our approach simplifies the VFL pipeline and preserves the privacy of client data, model weights, and intermediate results, by designing special function vectors and using a quadratic MIFE scheme to compute gradients directly.

References

[1] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, J. Joshi, and H. Ludwig, "Fedv: Privacy-preserving federated learning over vertically partitioned data," 03 2021.

[2] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer, 1999, pp. 223–238.

[3] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[4] M. Mohamad, M. Önen, W. Ben Jaballah, and M. Conti, "Sok: Secure aggregation based on cryptographic schemes for federated learning," in *PETS 2023, 23rd Privacy Enhancing Technologies Symposium, 10-14 July 2023, Lausanne, Switzerland (Hybrid Conference)*, Lausanne, 2023.

[5] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," 11 2017.

[6] K. Yang, T. Fan, T. Chen, Y. Shi, and Q. Yang, "A quasi-newton method based vertical federated learning framework for logistic regression," *ArXiv*, vol. abs/1912.00513, 2019.

[7] S. Yang, B. Ren, X. Zhou, and L. Liu, "Parallel distributed logistic regression for vertical federated learning without third-party coordinator," *arXiv preprint arXiv:1911.09824*, 2019.

[8] H. Sun, Z. Wang, Y. Huang, and J. Ye, "Privacy-preserving vertical federated logistic regression without trusted third-party coordinator," in *2022 The 6th International Conference on Machine Learning and Soft Computing*, ser. ICMLSC 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 132–138. [Online]. Available: https://doi.org/10.1145/3523150.3523171

[9] D. He, R. Du, S. Zhu, M. Zhang, K. Liang, and S. Chan, "Secure logistic regression for vertical federated learning," *IEEE Internet Computing*, vol. 26, no. 2, pp. 61–68, 2021.

[10] D. Zhao, M. Yao, W. Wang, H. He, and X. Jin, "Ntp-vfl - a new scheme for non-3rd party vertical federated learning," in *2022 14th International Conference on Machine Learning and Computing (ICMLC)*, ser. ICMLC 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 134–139. [Online]. Available: https://doi.org/10.1145/3529836.3529841

[11] X. Yu, W. Zhao, D. Tang, K. Liang, and J. Du, "Privacy-preserving vertical collaborative logistic regression without trusted third-party coordinator," *Sec. and Commun. Netw.*, vol. 2022, jan 2022. [Online]. Available: https://doi.org/10.1155/2022/5094830

[12] Q. Li, Z. Huang, W.-j. Lu, C. Hong, H. Qu, H. He, and W. Zhang, "Homopai: A secure collaborative machine learning platform based on homomorphic encryption," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1713–1717.

[13] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou, "Multi-input functional encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 578–602.

[14] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology – EUROCRYPT 2005*, R. Cramer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 457–473.

[15] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th Conference on Theory of Cryptography*, ser. TCC'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 535–554.

[16] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography*, Y. Ishai, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 253–273.

[17] S. Agrawal, R. Goyal, and J. Tomida, "Multi-input quadratic functional encryption from pairings," in *Crypto*, 2021, https://ia.cr/2020/1285.

[18] ——, "Multi-input quadratic functional encryption: Stronger security, broader functionality," in *Theory of Cryptography*, E. Kiltz and V. Vaikuntanathan, Eds. Cham: Springer Nature Switzerland, 2022, pp. 711–740.

[19] M. Abdalla, F. Benhamouda, and R. Gay, "From single-input to multi-client inner-product functional encryption," in *Advances in Cryptology – ASIACRYPT 2019*, S. D. Galbraith and S. Moriai, Eds. Cham: Springer International Publishing, 2019, pp. 552–582.

[20] B. Libert and R. ŢiŢiu, "Multi-client functional encryption for linear functions in the standard model from lwe," in *Advances in Cryptology – ASIACRYPT 2019*, S. D. Galbraith and S. Moriai, Eds. Cham: Springer International Publishing, 2019, pp. 520–551.

Algorithm 2 Construct c

```
function CGEN($\boldsymbol{w}, S, F, N, i, p$)
    $\mathcal{A} = \{\boldsymbol{a}_{i,n}\}_{n=0}^{N}$
    for each $n \in 0, ..., N$ do
        if $n \neq i$ and $i \neq N$ then
            $\boldsymbol{a}_{i,n} = \mathbf{0}^{S^2(F+1)F_i}$
        else if $n == i$ and $i \neq N$ then
            $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=0}^{F_i-1}$
            for each $f \in 0, ..., F_i - 1$ do
                if $f == p$ then
                    $\boldsymbol{d}_f = \boldsymbol{d} =$ SUBCGEN($\boldsymbol{w}, S, F_j, N$)
                else if $f \neq p$ then
                    $\boldsymbol{d}_f = \mathbf{0}^{S^2(F+1)}$
                end if
                $\boldsymbol{a}_{i,n} = [\boldsymbol{d}_0; \ldots; \boldsymbol{d}_{F_i-1}]$
            end for
        else if $i == N$ then
            $\boldsymbol{a}_{i,n} = \mathbf{0}^{S^2(F+1)}$
        end if
    end for
    Set $\boldsymbol{c}_{i,p} = [\boldsymbol{a}_{i,0}; \ldots; \boldsymbol{a}_{i,N}]$
    return $\boldsymbol{c}_{i,p}$
end function
```

Algorithm 3 Construct subc

```
function SUBCGEN($\boldsymbol{w}, S, F_j, N$)
    $d = \{\boldsymbol{d}_{s,n}\}, s \in [S], n \in [N+1]$
    for each $s \in 0, ..., S - 1$ do
        for each $j \in 0, ..., N$ do
            if $j \neq N$ then
                $\boldsymbol{d}_{s,j} = \mathbf{0}^{SF_j}$                ▷ Initialization
                for each $f \in 0, ..., F_j - 1$ do
                    $\boldsymbol{d}_{s,j}[fS + s] = \boldsymbol{w}_j[f]$
                end for
            else if $j == N$ then
                $\boldsymbol{d}_{s,N} = \mathbf{0}^S$                  ▷ Initialization
                $\boldsymbol{d}_{s,N}[s] = 1$
            end if
        end for
        $\boldsymbol{d}_s = [\boldsymbol{d}_{s,0}; \ldots; \boldsymbol{d}_{s,N}]$
    end for
    return $\boldsymbol{d} = [\boldsymbol{d}_0; \ldots; \boldsymbol{d}_{S-1}]$
end function
```

Appendix A
Security Definition for MIFE

In an indistinguishability-based security game between a challenger and an adversary, the challenger runs the Setup algorithm to generate the public parameters PP, N encryption keys EK_i, and master secret key MSK. The adversary then chooses the set of encryption keys that she wants. Then the adversary chooses two messages $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$ and gives them to the challenger. The challenger chooses a bit β at random and encrypts the message $\boldsymbol{x}^\beta$ using the i^{th} encryption key EK_i to get challenge ciphertext CT_i. The adversary then asks the challenger for the secret keys corresponding to the functions f. At last, the adversary guesses a bit β' and replies to the challenger. The admissible adversary wins if $\beta' = \beta$. In security, we want the probability of the adversary winning the security game to be negligibly close to 1/2.

The adversary is said to be admissible if and only if she sends at least one element of the form $(i, *, *)$ in the message space and she queries the secret key for the function f which satisfies the constraint that $f(\boldsymbol{x}^0) = f(\boldsymbol{x}^1)$. We formally define MIFE security in Definition 2.

Definition 2 (MIFE Security [18])**.** An MIFE scheme is IND-secure if for any stateful *admissible* PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, N \in \mathbb{N}$, the following probability is negligibly close to 1/2 in λ:

$$\Pr\left[\beta' = \beta : \begin{array}{l} \beta \leftarrow \{0,1\} \\ (\mathsf{PP}, \{\mathsf{EK}_i\}_{i\in[N]}, \mathsf{MSK}) \leftarrow \\ \mathsf{Setup}(1^\lambda, 1^N) \\ (\mathcal{CS}, \mathcal{MS}, \mathcal{FS}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{PP}) \text{ s.t.} \\ \mathcal{CS} \subseteq [N] \\ \mathcal{MS} = \{i^\mu, \boldsymbol{x}^{\mu,0}, \boldsymbol{x}^{\mu,1}\}_{\mu\in[q_c]} \\ \mathcal{FS} = \{f^\nu\}_{\nu\in[q_k]} \\ \{\mathsf{CT}_\mu \leftarrow \mathsf{Enc}(\mathsf{EK}_{i^\mu}, \boldsymbol{x}^{\mu,\beta})\}_\mu \\ \{\mathsf{SK}_\nu \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f^\nu)\}_\nu \\ \beta' \leftarrow \mathcal{A}\big(\{\mathsf{EK}_i\}_{i\in\mathcal{CS}}, \{\mathsf{CT}_\mu\}_\mu, \{\mathsf{SK}_\nu\}_\nu\big) \end{array}\right]$$

where the adversary $\mathcal{A}$ is said to be admissible if and only if

- $q_c[i] > 0$ for all $i \in [N]$, where $q_c[i]$ denotes the number of elements of the form $(i, *, *)$ in $\mathcal{MS}$.
- $f(\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_n^0) = f(\boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_n^1)$ for all sequences $(\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_n^0, \boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_n^1, f)$ such that:
  - For all $i \in [n]$, $[(i, \boldsymbol{x}_i^0, \boldsymbol{x}_i^1) \in \mathcal{MS}]$ or $[i \in \mathcal{CS}$ and $\boldsymbol{x}_i^0 = \boldsymbol{x}_i^1]$,
  - $f \in \mathcal{FS}$.

Appendix B
Pseudocode

In this section, we give reference to the algorithms used for constructing the function vectors. The algorithm 2 shows the construction of $\boldsymbol{c}_{i,p}$ and the algorithm 3 shows the construction of vector d.

Appendix C
Extension to Logistic Regression Model

In this section, we extend the protocol to work for logistic regression with the help of Taylor approximation. The prediction function of logistic models is as follows:

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 - e^{-\boldsymbol{x}\boldsymbol{w}}} \tag{11}$$

We use Cross-Entropy as the loss function for logistic regression. The loss function in the vectorized form is

$$L(\boldsymbol{w}) = \frac{1}{S}\left[-\boldsymbol{y}^\top \log(f(\boldsymbol{X}, \boldsymbol{w})) - (\mathbf{1} - \boldsymbol{y})^\top \log(\mathbf{1} - f(\boldsymbol{X}, \boldsymbol{w}))\right]$$

(12)

Here we use Taylor approximation to make the loss function polynomial. In [5], it takes a Taylor Series expansion of $\log(1+e^{-z})$ around $z = 0$.

$$\log(1 + e^{-z}) = \log 2 - \frac{1}{2}z + \frac{1}{8}z^2 - \frac{1}{192}z^4 + O(z^6) \tag{13}$$

We apply (13) to (12) and get the gradients of vector-format expression as given below.

$$g(\boldsymbol{w}) \approx \frac{1}{S}\left(\frac{1}{4}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} + \frac{1}{2}\right)^{\top}\boldsymbol{X} \tag{14}$$

Then we decompose $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}_0\boldsymbol{w}_0 + \boldsymbol{X}_1\boldsymbol{w}_1 + \ldots + \boldsymbol{X}_{N-1}\boldsymbol{w}_{N-1}$ and $\boldsymbol{X} = [\boldsymbol{X}_0||\boldsymbol{X}_1||\ldots||\boldsymbol{X}_{N-1}]$ and substitute the decomposition into (14) to get

$$g(\boldsymbol{w}) \approx \frac{1}{S}\begin{bmatrix} -(\boldsymbol{y} - \frac{1}{2})^{\top}\boldsymbol{X}_0 + \frac{1}{4}\sum_{j=0}^{N-1}\boldsymbol{w}_j^{\top}\boldsymbol{X}_j^{\top}\boldsymbol{X}_0|| \\ \ldots|| \\ -(\boldsymbol{y} - \frac{1}{2})^{\top}\boldsymbol{X}_{N-1} + \frac{1}{4}\sum_{j=0}^{N-1}\boldsymbol{w}_j^{\top}\boldsymbol{X}_j^{\top}\boldsymbol{X}_{N-1} \end{bmatrix} \tag{15}$$

Each term in (15) has the similar format to the term in (2) except for $\boldsymbol{w}_j^{\top}\boldsymbol{X}_j^{\top}\boldsymbol{X}_i$ in (15) has coefficient $\frac{1}{4}$ and $\boldsymbol{y}^{\top}\boldsymbol{X}_i$ in (2) becomes $(\boldsymbol{y}-\frac{1}{2})^{\top}\boldsymbol{X}_i$ in (15). We can modify the protocol to compute the gradients for non-linear models without exposing labels $\boldsymbol{y}$ and $\boldsymbol{X}_i$. First, when the aggregator constructs function vector $\boldsymbol{c}$, instead of using the original weights, we multiply the weights with $\frac{1}{4}$ element-wise and use the modified weights to construct $\boldsymbol{c}$. Moreover, the active party, instead of sending the ciphertext of $\boldsymbol{y}$, now sends the ciphertext of $\boldsymbol{y} - \frac{1}{2}$. After decrypting and concatenating all the elements in (15), the aggregator multiplies the concatenated results with $\frac{1}{S}$ to obtain the gradients. Other procedures remain same as the procedure for the linear regression models.