FEVERLESS
Fast and Secure Vertical Federated Learning based on XGBoost for Decentralized Labels

Wang, Rui; Ersoy, Oguzhan; Zhu, Hangyu; Jin, Yaochu; Liang, Kaitai

DOI
10.1109/TBDATA.2022.3227326

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Big Data

Citation (APA)
Wang, R., Ersoy, O., Zhu, H., Jin, Y., & Liang, K. (2024). FEVERLESS: Fast and Secure Vertical Federated Learning based on XGBoost for Decentralized Labels. *IEEE Transactions on Big Data*, *10*(6), 1001-1015. https://doi.org/10.1109/TBDATA.2022.3227326

FEVERLESS: Fast and Secure Vertical Federated Learning Based on XGBoost for Decentralized Labels

Rui Wang, Oğuzhan Ersoy, Hangyu Zhu, Yaochu Jin, *Fellow, IEEE*, and Kaitai Liang, *Member, IEEE*

Abstract—Vertical Federated Learning (VFL) enables multiple clients to collaboratively train a global model over vertically partitioned data without leaking private local information. Tree-based models, like XGBoost and LightGBM, have been widely used in VFL to enhance the interpretation and efficiency of training. However, there is a fundamental lack of research on how to conduct VFL securely over distributed labels. This work is the first to fill this gap by designing a novel protocol, called FEVERLESS, based on XGBoost. FEVERLESS leverages secure aggregation via information masking technique and global differential privacy provided by a fairly and randomly selected noise leader to prevent private information from being leaked in the training process. Furthermore, it provides label and data privacy against honest-but-curious adversaries even in the case of collusion of $n-2$ out of n clients. We present a comprehensive security and efficiency analysis for our design, and the empirical results from our experiments demonstrate that FEVERLESS is fast and secure. In particular, it outperforms the solution based on additive homomorphic encryption in runtime cost and provides better accuracy than the local differential privacy approach.

Index Terms—Differential privacy, privacy preservation, secure aggregation, vertical federated learning, XGBoost

1 Introduction

Traditional centralized deep learning models, demanding to collect a considerable amount of clients' data to maintain high accuracy, to some degree, may increase the risk of data breaches. Data may not be easily shared among different entities due to privacy regulations and policies. To tackle this "Data Island" problem [1], Google proposed Federated Learning (FL) [2] to allow multiple clients to train a global model without sharing private data. The basic paradigm of FL is that all clients train local models with their own data, and then the information of local models, e.g., gradients, may be exchanged to produce a global model.

- *Rui Wang and Kaitai Liang are with the Department of Intelligent Systems, Delft University of Technology, 2628 XE Delft, The Netherlands. E-mail: {r.wang-8, kaitai.liang}@tudelft.nl.*
- *Oğuzhan Ersoy is with the Institute for Computing and Information Sciences, Radboud University, 6525 EC Nijmegen, The Netherlands, and also wtih the Department of Intelligent Systems, Delft University of Technology, 2628 XE Delft, The Netherlands.*
- *Hangyu Zhu is with the School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi, Jiangsu 214126, China. E-mail: hangyu.zhu@jiangnan.edu.cn.*
- *Yaochu Jin is with the Faculty of Technology, Bielefeld University, 33619 Bielefeld, Germany, and also with the Department of Computer Science, University of Surrey, GU2 7XH Guildford, U.K. E-mail: yaochu.jin@surrey.ac.uk.*

Manuscript received 19 May 2022; revised 2 November 2022; accepted 5 December 2022. Date of publication 7 December 2022; date of current version 13 November 2024.
This work was supported by European Union's Horizon 2020 research and innovation programme under Grants 952697 (ASSURED) and 101021727 (IRIS).
(Corresponding author: Rui Wang.)
Recommended for acceptance by Jesus M. Cortez.
Digital Object Identifier no. 10.1109/TBDATA.2022.3227326

Based on different types of data partition [1], FL can be mainly categorized into Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). The former focuses on training with horizontally partitioned data where clients share the same feature space but differ in data index set. Several research works [3], [4], [5], [6] have found that training data of HFL is still at high risk of leakage although private data is kept locally. Other studies [7], [8], [9], [10], [11] have been dedicated to enhancing the security of HFL. On the contrary, VFL is mainly applied in the scenario of training with vertically partitioned data [12], [13] where clients share the same data index set but differ in feature space. In this article, our principal focus is to achieve privacy-preserving training on VFL.

To the best of our knowledge, many existing studies [12], [13], [14], [15], [16], [17], [18] have proposed innovative approaches to prevent private information breaches in the context of VFL. Specifically, [14] introduced encryption-based privacy-preserving logistic regression to safeguard the information of data indexes. [15] gave a comprehensive discussion on the impact of ID resolution. [17] introduced a scheme without using a coordinator for a limited number of clients. Recently, [16] proposed an asymmetrically VFL scheme for logistic regression tackling privacy concerns on ID alignment.

Unlike the training models used in the aforementioned works, XGBoost [18], which is one of the most popular models applied in VFL, can provide better interpretation, easier parameter tuning, and faster execution than deep learning in tabular data training [19], [20]. These practical features and advantages draw academia and industry's attention to the research on XGBoost, especially in the privacy-preserving context. [12] introduced an approach for tree-based model training through a hybrid method composing homomorphic encryption and secure Multi-Party Computation (MPC) [21], [22]. After that, [13] proposed a similar system to train XGBoost [18] securely over vertically partitioned data by using Additively Homomorphic Encryption (AHE). By applying Differential Privacy (DP) [23], [24] designed a VFL system to train GBDT without the need of encryption/decryption.

However, most of the above solutions based on AHE and MPC do not scale well in terms of efficiency on training XGBoost. Beyond that, all the existing schemes basically assume that training labels are managed and processed by a *sole* client. In practice, *a VFL scheme supporting distributed labels is necessary*. For instance, multiple hospitals, clinics and health centers currently may be set to COVID-19 test spots and aim to train a model, e.g., XGBoost, to predict with good interpretation if citizens (living in various locations) are infected based on their health records and symptoms. In this context, the labels (and their values), e.g., the test results, are likely distributed among different health authorities - even targeting to the same group of patients, and feature space is vertically portioned. For example, a cardiac hospital only maintains heart data for the patients, while a psychiatric center holds the mental records, in which both authorities may collect and manage each of its registered patient's label locally. Another common scenario could be in the financial sector where multiple bank branches and e-commerce companies prefer to build a global model to predict if their customers may pay for some service (e.g., car loan) on time. The banks have part of features about the customers (e.g., account balance, funding in-and-out records), while the companies may obtain other features (e.g., payment preference). Since the customers may get the same service, e.g., loan, from different institutions, it is clear that labels must be distributed rather than centralized. In addition to efficiency and functionality aspects, one may also consider capturing stronger security for VFL. Training an XGBoost usually should involve the computation of first and second-order derivatives of the loss function (note gradients and hessians contain labels' information), and the aggregation of them is required in each round. In the context where the labels (and their values) are held by different clients, if the gradients and hessians are transmitted in the form of plaintexts and the summations of them are known to an aggregator (who could be one of the clients engages in training), inference and differential attacks (Section 3.3) will be easily conducted by the aggregator, resulting in information leakage.

To tackle these problems, we propose a fast and secure VFL protocol, FEVERLESS, to train XGBoost on distributed labels without disclosing both feature and label value. In our design, privacy protection is guaranteed by secure aggregation (based on a masking scheme) and Global Differential Privacy (GDP). We leverage masking instead of heavy-cost multiparty computation and we guarantee a "perfect secrecy" level for the masked data. In GDP, we use Verifiable Random Function (VRF) to select a noise leader per round (who cannot be predicted and pre-compromised in advance) to aggregate noise from "selected" clients, which significantly maintains model accuracy.

Our contributions can be summarized as follows.

(1) We define VFL in a more practical scenario where training labels are distributed over multiple clients. Beyond that, we develop FEVERLESS to train XGBoost securely and efficiently with the elegant combination of secure aggregation technique (based on Diffie-Hellman (DH) key exchange and Key Derivation Function (KDF) and GDP.

(2) We give a comprehensive security analysis to demonstrate that FEVERLESS is able to safeguard label value and feature privacy in the semi-honest setting, but also maintain robustness even for the case where $n-2$ out of n clients commit collusion.

(3) We implement FEVERLESS and perform training time and accuracy evaluation on different real-world datasets. The empirical results show that FEVERLESS can maintain efficiency and accuracy simultaneously, and its performance is comparable to the baseline - a "pure" XGBoost without using any encryption and differential privacy. Specifically, training the credit card and bank marketing datasets just takes 1% and 6.5% more runtime than the baseline and meanwhile, the accuracy is only lower than that of the baseline by 0.9% and 3.21%, respectively.[1]

2 Preliminaries

2.1 Xgboost

XGBoost [18] is a popular tree-based model in tabular data training that can provide better interpretation, easier parameters tuning and faster execution speed than deep learning [19], [20]. It also outperforms other well-known boosting tree systems in terms of accuracy and efficiency, like Spark MLLib [25] and H2O [18], especially for large-scale datasets. Therefore, in this paper, we consider using XGBoost as a building block for classification tasks.

Assume that a training set with m data points composing with feature space $\mathcal{X} = \{x_1, \ldots, x_m\}$ and label space $\mathcal{Y} = \{y_1, \ldots, y_m\}$. Before training starts, every feature will be sorted based on its values, and split candidates will be set for features. XGBoost builds trees based on the determination of defined split candidates and some pruning conditions. Specifically, computing gradients and hessians first according to Eqs. (1) and (2) for each data entry, where $y_i^{(t-1)}$ denotes the prediction of previous tree for i-th data point, and y_i is the label of ith data point:

$$g_i = \frac{1}{1 + e^{-y_i^{(t-1)}}} - y_i = \hat{y}_i - y_i, \tag{1}$$

$$h_i = \frac{e^{-y_i^{(t-1)}}}{(1 + e^{-y_i^{(t-1)}})^2}. \tag{2}$$

For splitting nodes, the XGBoost algorithm determines the best split candidate from all others based on maximum L_{split} in Eq. (3), where λ and γ are regularization parameters:

$$L_{split} = \frac{1}{2}\left[\frac{\sum_{i \in I_L} g_i}{\sum_{i \in I_L} h_i + \lambda} + \frac{\sum_{i \in I_R} g_i}{\sum_{i \in I_R} h_i + \lambda} - \frac{\sum_{i \in I} g_i}{\sum_{i \in I} h_i + \lambda}\right] - \gamma. \tag{3}$$

1. For banknote authentication dataset, FEVERLESS takes 13.96% more training time than the baseline, and the accuracy is 30.4% lower. This is because the model is trained by a small-scale dataset, so that the robustness is seriously affected by noise.

The current node will be the leaf node if the following conditions are fulfilled: reaching the maximum depth of tree, the maximum value of impurity is less than a preset threshold. The calculation of the leaf value follows:

$$w = -\frac{\sum_{i\in I} g_i}{\sum_{i\in I} h_i + \lambda}. \tag{4}$$

2.2 Diffie-Hellman Key Exchange

Based on Decision Diffie-Hellman (DDH) hard problem [26] defined below, Diffie-Hellman key exchange (DH) [27] provides a method used for exchanging keys across public communication channels. Without losing generality and correctness, it consists of a tuple of algorithms (**Param.Gen, Key.Gen, Key.Exc**). The algorithm $(\mathbb{G}, g, q) \leftarrow$ **Param.Gen** (1^{α}) generates public parameters (a group $\mathbb{G}$ with prime order q generated by a generator g) based on secure parameter α. $(sk_i, pk_i) \leftarrow$ **Key.Gen**$(\mathbb{G}, g, q)$ allows client i to generate secret key $(sk_i \xleftarrow{\$} \mathbb{Z}_q)$ and compute public key $(pk_i \leftarrow g^{sk_i})$. Shared key is computed by $(pk_i^{sk_j}, pk_j^{sk_i}) \leftarrow$ **Key.Exc**(sk_i, pk_i, sk_j, pk_j). Inspired by [22], [28], we utilize shared keys as maskings to protect the information of labels against inference attacks during transmission in public channels. The correctness requires $pk_i^{sk_j} = pk_j^{sk_i}$. The security relies on the DDH problem [26], which is defined as:

Definition 2.1 (Decision Diffie-Hellman). *Let $\mathbb{G}$ be a group with prime order q and g be the fixed generator of the group. The Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ is given and g^a and g^b where a and b are randomly chosen. The probability of $\mathcal{A}$ distinguishing (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) for a randomly chosen c is negligible:*

$$\left| \Pr\left[a, b \xleftarrow{\$} \mathbb{Z}_q \colon \mathcal{A}(g, g^a, g^b, g^{ab}) = \text{true}\right] - \Pr\left[a, b, c \xleftarrow{\$} \mathbb{Z}_q \colon \mathcal{A}(g, g^a, g^b, g^c) = \text{true}\right] \right| < negl(\alpha).$$

2.3 Pseudo-Random Generator and Hash Function

Pseudo-Random Generator (PRG) [29] is an algorithm that is able to generate random numbers. The "pseudo-random" here means that the generated number is not truly random but has similar properties to a random number. Generally, the pseudo-random numbers are determined by given initial values a.k.a seeds. In cryptographic applications, a secure PRG requires attackers not knowing seeds can distinguish a truly random number from an output of PRG with a negligible probability. Similar to PRG, the hash function allows mapping arbitrary sizes of data to a fixed bit value. For reducing the communication cost of FEVERLESS, we use `SHAKE-256` [30], one of the hash functions in `SHA-3` [31] family, to generate the customized size of maskings.

2.4 Key Derivation Function

Key Derivation Function (KDF) [32] is a kind of hash function that derives multiple secret keys from the main key by utilizing Pesudo-Random Function (PRF) [33]. In general, KDF algorithm $DK \leftarrow KDF(mainkey, salt, rounds)$ derives keys DK based on the main key, a cryptographic salt and the current round of processing algorithm. The security requires a secure KDF that is robust for brute-force attacks or dictionary attacks. Inspired by [34] where key shares generated by DH key exchange are converted to AES keys, in this paper, we use KDF to generate maskings for every round to reduce communication costs. The main key we use is generated by DH key exchange.

2.5 Verifiable Random Function

Verifiable Random Function (VRF) [35] is a PRF providing verifiable proof of the correctness of outputs. It is a tool widely used in cryptocurrencies, smart contracts and leader selection in distributed systems [36]. Basically, given an input x, a signature scheme and a hash function, a practical leader selection scheme with VRF [36] works as:

$$S_{leader} \leftarrow \mathsf{H}(\text{sign}_{sk_i}(x)) \Rightarrow \tag{5}$$

where sk_i is the secret key for i-th client, and the maximum leader score S_{leader} is used to determine leader. The security and unforgeability of VRF require that the signature scheme has the property of uniqueness, and the hash function is able to map the signature to a random string with a fixed size. The correctness of this S_{leader} is proved by the signature of x.

2.6 Differential Privacy

Differential Privacy (DP) [37], [38] is a data protection system targeting on the publishing of statistical information of datasets while keeping individual data private. The security of DP requires that adversaries cannot distinguish statistical change between two datasets where an arbitrary data point is different.

The most widely used DP mechanism is called (ϵ, δ)-DP requiring less noise injected than originally proposed ϵ-DP but with the same privacy level. The formal definition is given as follows.

Definition 2.2. ((ϵ, δ) ***- Differential Privacy)*** *Given two real positive numbers (ϵ, δ) and a randomized algorithm $\mathcal{A}$: $\mathcal{D}^n \rightarrow \mathcal{Y}$, the algorithm $\mathcal{A}$ provides (ϵ, δ) - differential privacy if for all data sets D, $D' \in \mathcal{D}^n$ differing in only one data sample, and all $\mathcal{S} \subseteq \mathcal{Y}$:*

$$Pr[\mathcal{A}(D) \in \mathcal{S}] \leq exp(\epsilon) \cdot Pr[\mathcal{A}(D') \in \mathcal{S}] + \delta. \tag{6}$$

Note the noise $\mathcal{N} \sim N(0, \Delta^2\sigma^2)$ will be put into the output of the algorithm, where Δ is l_2 - norm sensitivity of D and $\sigma = \sqrt{2\ln(1.25/\delta)}$ [39].

3 Problem Formulation

3.1 System Model

We here make some assumptions on our system. We suppose that a private set intersection [40], [41] has been used to align data IDs before the training starts, so that each client shares the same data index space $\mathcal{I}$. But the names of features are not allowed to share among clients. As for the relationship of label tagging (indexes indicating a label belongs to which client, e.g., the label a is held by client A, B), we will consider that this can be known to the public in advance. But this assumption does not mean that the label

TABLE 1
Notations Summary

Notation	Description
$\mathcal{X}$	feature space
$X_j^{(c)}$	j-th feature owned by c-th client
x_i	i-th data point with d features
$\mathcal{Y}$	label space
$y_i^{(c)}$	the label of i-th data point owned by c-th client
$\mathcal{I}$	data index space
$\mathcal{C}$	clients set
$g_i^{(c)}$	the gradient of i-th data point owned by c-th client
$h_i^{(c)}$	the hessian of i-th data point owned by c-th client
G	summation of gradients
H	summation of hessians
m	number of data entries
n	number of clients
f	number of features
d	the maximum depth of tree
ϵ, δ	parameters of differential privacy
Δ_g	sensitivity of gradients
Δ_h	sensitivity of hessians
L_{split}	impurity score
w	leaf value
pk_c	public key generated by c-th client
sk_c	secret key owned by c-th client
g	generator of multiplicative group
B_z^j	z-th bucket of j-th feature

"value" is leaked. For instance, client C knows that client A, B have a, but it does not know the specific value of a. We also consider that the training is conducted on a dataset with m samples composing with feature space $\mathcal{X} = \{x_1, \ldots, x_m\}$, each containing f features, and label set $\mathcal{Y} = \{y_1, \ldots, y_m\}$. Besides, features $\{X_j^{(c)} \mid j \in \{1, \ldots, f\}\}$ and labels $\{y_i^{(c)} \mid i \in \{1, \ldots, m\}\}$ are held among n clients where each client has at least one feature and one label. $X_j^{(c)}$ and $y_i^{(c)}$ refer to j-th feature and i-th label owned by c-th client, respectively. Note we summarize the main notations in Table 1.

Considering a practical scenario wherein training labels are distributed among clients, we propose a new variant of VFL, named VFL over Distributed Labels (DL-VFL). The concrete definition is given as follows.

Definition 3.1 (DL-VFL). *Given a training set with m data samples consisting of feature space $\mathcal{X}$, label space $\mathcal{Y}$, index space $\mathcal{I}$ and clients set $\mathcal{C}$, we have:*

$$\mathcal{X}^c \cap \mathcal{X}^{c'} = \emptyset, \left|\mathcal{Y}^c \cap \mathcal{Y}^{c'}\right| < m, \mathcal{I}^c = \mathcal{I}^{c'}, \forall c, c' \in \mathcal{C}, c \neq c'.$$

Remarks. Different clients hold the subset of $\mathcal{X}$ sampled from feature space. A client c participating DL-VFL shares the same sample ID space $\mathcal{I}$ with the corresponding labels, where a single label may be tagged to multiply clients (1-to-many case), for example, the label $a \rightarrow$ $client\ A,\ B$. One may easily see the special case where a single label is assigned to only one client (i.e. $\mathcal{Y}^c \cap \mathcal{Y}^{c'} = \emptyset$), 1-to-1 case. Recall that the "tagging" relationship between label and client can be publicly known. Based on this assumption, our experiments are conducted in 1-to-1 cases for simplicity. We state that the designed experiments are also compatible with 1-to-many cases. This is so because the assumption allows the source client (which is defined below) to have knowledge of label holders, so that it can request "distinct" and missing labels from those holders, e.g., requesting a from either client A or B. Note as for 1-to-many, the size of missing labels, $|mIDs|$, could be smaller than 1-to-1, which may require less communication cost and runtime.

We will further require the participation of the source client and noise leader in our design. And they are defined as follows.

Definition 3.2 (Source client). *A source client with split candidates wants to compute the corresponding L_{split} based on Eq. (3). But some labels are missing so that $\sum g_i$ and $\sum h_i$ are unable to derive.*

For the case that a source client does not hold all labels in the current split candidates, we propose a solution based on secure aggregation and global differential privacy to help the source client to compute L_{split} while safeguarding other clients' privacy. Note each client may have a chance to act as a source client because all the labels are distributed, where the *source client* leads the L_{split} computation, and *clients* provide missing label values to the source client.

To achieve GDP, we define a noise leader who is selected fairly and randomly from all clients (except for the source client) - preventing clients from being compromised beforehand.

Definition 3.3 (Noise leader). *By using VRF, a noise leader is responsible for generating the maximum leader score , aggregating differentially private noise from a portion of clients and adding the noise to the gradients and hessians.*

3.2 Threat Model

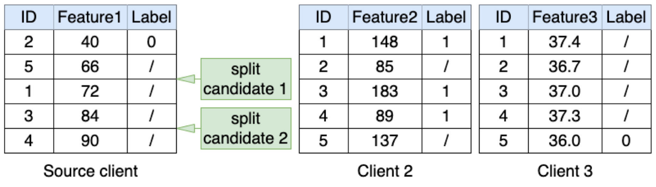
We mainly consider potential threats incurred by participating clients and outside adversaries. We assume that all clients are *honest-but-curious*, which means they strictly follow designed algorithms but try to infer the private information of others from the received messages. Besides, we also consider up to $n-2$ clients' collusion to conduct attacks, and at least one non-colluded client adds noise per round. Through authenticated channels, DH key exchange can be securely executed among clients. Other messages are transmitted by public channels, and outside attackers can eavesdrop on these channels and try to reveal information about clients during the whole DL-VFL process. Note this paper mainly focuses on solving privacy issues in training DL-VFL based on XGBoost. Thus, other attacks, like data poisoning and backdoor attacks deteriorating model performance, are orthogonal to our problem.

3.3 Privacy Concern

Since we assume feature names are not public information for all clients, and the values of features never leave clients, privacy issues are mainly incurred by the leakage of label information.

3.3.1 Inference Attack

During the training process, gradients and hessians are sent to the source client for L_{split} computation. For the classification task, the single gradient is in the range $(-1, 0) \cup (0, 1)$ for binary classification. According to Eq. (1), a label can be



ID	Feature1	Label
2	40	0
5	66	/
1	72	/
3	84	/
4	90	/

Source client

ID	Feature2	Label
1	148	1
2	85	/
3	183	1
4	89	1
5	137	/

Client 2

ID	Feature3	Label
1	37.4	/
2	36.7	/
3	37.0	/
4	37.3	/
5	36.0	0

Client 3

Fig. 1. A differential attack on single node split.

inferred as 1 and 0 if the range is $(-1, 0)$ and (0,1), respectively. Besides, hessian illustrated in Eq. (2) can leak a prediction of the corresponding data sample. With training processing, the prediction is increasingly closer to a true label. The source client and outside attackers can infer the true label with high probability. Gradients and hessians cannot be transmitted in plaintext. We thus use a secure aggregation scheme to protect them from inference attacks.

3.3.2 Differential Attack

The differential attack can happen anytime and many times during the calculation of gradients and hessians. Fig. 1 describes an example of a differential attack taking place in a single node split. After sorting feature1, the semi-honest source client defines 2 split candidates and further computes $G_{\{2,5\}} = g_2 + g_5$ and $G_{\{1,2,3,5\}} = g_2 + g_5 + g_1 + g_3$ for the candidates 1 and 2, respectively. Since the source client holds label 2, even if $G_{\{2,5\}}$ is derived by secure aggregation, the g_5 still can be revealed by $G_{\{2,5\}} - g_2$.

Another example of differential attack is shown in Fig. 2. Assume split candidate 1 is the one for splitting the root node. In the current tree structure, the source client may split the right node by computing L_{split} of split candidate 2. In this case, $G_{\{1,3\}}$ should be aggregated by the source client. And the g_5 can be revealed by $G_{\{1,2,3,5\}} - G_{\{1,3\}} - g_2$, where $G_{\{1,2,3,5\}}$ is computed in the previous node.

4 A Practical Privacy-Preserving Protocol

4.1 FEVERLESS Protocol Description

To prevent a source client from knowing gradients and hessians sent by other clients, one may directly use MPC [42] based on AHE [12], [43]. But this method yields expensive computation costs. Getting rid of the complex mechanism like MPC, we leverage secure aggregation protocol via masking scheme based on DH key exchange [22], [24], [28]. By further using KDF and Hash Function, our masking (for gradients and hessians) can be derived without exchanging keys per training round. Our approach significantly reduces the communication cost but still maintains the robustness up to $n-2$ colluded clients. Meanwhile, the secure aggregation can provide "perfect secrecy" for broadcast messages. After receiving the broadcast messages, the masking will be canceled out at the source client side. But only using the masking is unable to defend against differential attacks. One may consider using Local Differential Privacy (LDP) [44] to make sure that each client may add noise per send-out message, barely consuming any extra computation cost. The accumulated noise, from all clients, may seriously affect the model's accuracy. To tackle this problem, we use a GDP [45] approach with noise leader selection. A hybrid method is finally formed based on a masking scheme and GDP, so that per client's sensitive information can be protected by the "masks" and the aggregated values are secured by the noise which is injected by the chosen clients.

We here briefly introduce our design. Assume each client $c \in [1, n]$ generates respective secret key sk_c and computes gradients $g_i^{(c)}$ and hessians $h_i^{(c)}$ locally, where $\{i \mid y_i \in \mathcal{Y}^c\}$. FEVERLESS works as follows.

1. *Broadcast missing indexes.* The source client broadcasts the $mIDs$= $\{i \mid y_i \notin \mathcal{Y}^c\}$. Regardless of 1-to-1 or 1-to-many cases, the source client will need to send out the missing indexes (with knowledge of tagging relationships).

2. *Key exchange computation.* Each client c computes public key $pk_c = g^{sk_c}$ using secret keys sk_c, sends pk_c to other clients and computes the corresponding shared keys[2] $\{S_{c,c'} = pk_{c'}^{sk_c} = g^{sk_c sk_{c'}} \mid c, c' \in \mathcal{C}, c \neq c'\}$ based on secret key sk_c received public keys $\{pk_{c'} \mid c' \in \mathcal{C}\}$.

3. *Data masking.* Each client c runs the masking generation algorithm to compute the maskings for protecting gradients and hessians. Specifically, based on KDF, clients' indexes and the number of queries, the masking generation algorithm is conducted by $\mathtt{mask}_g^{(c)} \leftarrow \sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot (\mathsf{H}(S_{c,c'} \| 0 \| \mathtt{query}), \mathtt{mask}_h^{(c)} \leftarrow \sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot (\mathsf{H}(S_{c,c'} \| 1 \| \mathtt{query})$[3]. Then the masked gradients $G^{(c)}$ and hessians $H^{(c)}$ are generated by $G^{(c)} = \sum_{i \in mIDs} g_i^{(c)} + \mathtt{mask}_g^{(c)} - r_g^{(c)}, H^{(c)} = \sum_{i \in mIDs} h_i^{(c)} + \mathtt{mask}_h^{(c)} - r_h^{(c)}$.

4. *Noise leader selection.* Each client generates the selection score $selec_c$ using the VRF, $\mathsf{H}(\mathtt{SIGN}_{sk_c}(\mathtt{count}, \mathtt{mIDs}, \mathtt{r}))$, and broadcasts it, where `count` is the number of times clients conduct VRF, `r` is a fresh random number, and `SIGN` is the signature scheme. The client with the maximum score will be the noise leader. For ease of understanding, we assume client n with the largest selection score $select_n^{max}$ is the leader, in Fig. 3.

5. *Noise injection.* a) Noise leader selects k clients adding noise. For the details of the selection, please see Algorithm 5. b) The selected clients send $\{\widetilde{n_g^{(c)}} = N(0, \Delta_g^2 \sigma^2) + r_g^{(c)}, \widetilde{n_h^{(c)}} = N(0, \Delta_h^2 \sigma^2) + r_h^{(c)} | c \in k\}$ to noise leader, in which the $r_g^{(c)}$ and $r_h^{(c)}$ are two random values to mask noise. c) The leader aggregates the noise: $\widetilde{N_g} = k \cdot N(0, \Delta_g^2 \sigma^2) + R_g$ and $\widetilde{N_h} = k \cdot N(0, \Delta_h^2 \sigma^2) + R_h$, and further adds them to $G^{(n)}$ and $H^{(n)}$, respectively.

6. *Aggregation and computation.* All clients send the masked values to the source client. The source client computes $\sum_{c=1}^n G^{(c)} + k \cdot N(0, \Delta_g^2 \sigma^2)$, $\sum_{c=1}^n H^{(c)} + k \cdot N(0, \Delta_h^2 \sigma^2)$ and L_{split}.

7. *Final update.* The source client with maximum L_{split} updates the model following XGBoost [18] and broadcasts the updated model and data indexes in child nodes as step 8.

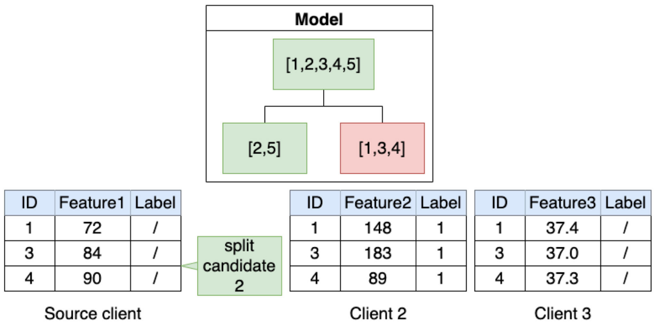
We present an overview of FEVERLESS in Fig. 3. Note the depicted process can be conducted iteratively.

4.2 XGBoost Training Over Distributed Labels

At the initial stage, we allow all clients to agree on a tree structure (maximum depth and the number of trees) and the learning rate for updating prediction. To avoid the

2. Shared keys are only generated once, and the KDF is used to generate the remaining maskings.

3. For simplicity, we do not show the modular computations here. The full description is elaborated on Algorithms 3, 4, and 5.



ID	Feature1	Label
1	72	/
3	84	/
4	90	/

ID	Feature2	Label
1	148	1
3	183	1
4	89	1

ID	Feature3	Label
1	37.4	/
3	37.0	/
4	37.3	/

Fig. 2. A differential attack on multiply node splits.

overfitting problem, we should define regularization parameters. Threshold impurity is also another vital parameter used to identify tree and leaf nodes via the maximum impurity. After that, we should choose ϵ, δ for DP, a hash function for masking generation, and noise leader selection. Besides, we select a multiplicative group $\mathbb{G}$ with order q generated by a generator g and a large prime number p to run DH.

During the initialization process, all clients set parameters and sort their own features based on values. Then, split candidates can be defined, and data samples between two different candidates will be grouped as a bucket. In the end, all entries are assigned initialized values to calculate the derivatives of the loss function. The detailed algorithm is described as follows.

Algorithm 1. Initialization

```
1: Set parameters: all clients agree on the maximum depth of a tree $d$, the number of trees $(NT)$, learning rate $(\eta)$, regularization parameters $(\lambda, \gamma)$, the threshold of $L_{split}$, $\epsilon$, $\delta$, $p$, $g$, selection portion $(p)$ and hash function
2: for $c \in [1, n]$ do
3:   for each feature $j$ owned by $c$ do
4:     sort($X_j^{(c)}$)
5:     define buckets: $B_z^j$
6:   set initialized values: $\hat{y}_i{}^{(c)}$
```

After initialization, all clients can invoke Algorithm 2 to train the model collaboratively. The inputs are from feature space consisting of features $X_j^{(c)}$ and labels $y_i^{(c)}$ distributed on different clients, respectively; while the output is a trained XGBoost model that can be used for prediction. Generally, trees are built one by one. And we see from lines 4-10 in Algorithm 2 that each client can compute gradients and hessians at beginning of a new tree construction.

Following that, clients are to split the current node. Note that XGBoost training in DL-VFL requires each client to calculate G and H. If the labels in some buckets are incomplete, the corresponding gradients and hessians cannot be computed. Thus, each client should first broadcast the missing data index set mID (see lines 15-17 in Algorithm 2). Based on the predefined bucket B_z^j, mID can be defined if labels in B_z^j are not held by clients. In each broadcast, a client sending messages is regarded as a source client. Then others send the corresponding $g_i^{(c')}$ and $h_i^{(c')}$ back to the source client to compute L_{split} through Algorithms 3, 4, and 5. After finding a maximum impurity $L_{split\_max}^c$, the current node will be split into "left" and "right" nodes if $L_{split\_max}^c > threshold\_L_{split}$, in which the value of the split candidate is own by c.

In node splitting, clients should set a given node as "leaf" if current depth reaches the predefined maximum depth or the maximum L_{split} is less than the predefined threshold of L_{split} (see line 12, 24-32 in Algorithm 2). The derivation of leaf value is followed by (4) where G and H are intaken. Since a leaf node is either "left" or "right" split by one of the clients in $\mathcal{C}$ from its parent node, this client knows G and H and leaf value can be derived. Finally, this leaf value will be broadcast, and clients who own the corresponding $g_i^{(c)}$ and $h_i^{(c)}$ can use it to update predictions. The details for the above process are shown in Algorithm 2.

4.3 Secure Aggregation With Global Differential Privacy

In lines 15-19 of Algorithm 2, the source client is able to compute L_{split} from the requested missing data indexes and the aggregation of received messages. To avoid that inference

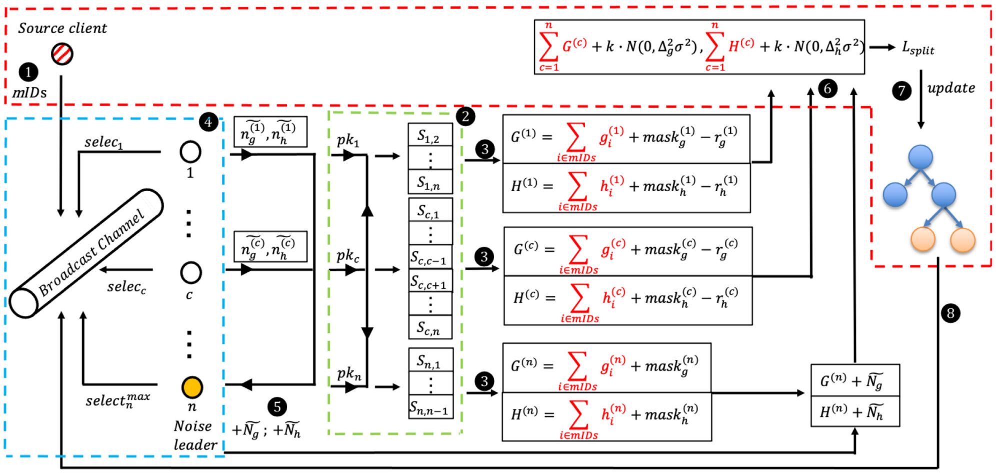

Source client
❶
mIDs
Broadcast Channel
$selec_1$
$selec_c$
$select_n^{max}$
❹
1
c
n
Noise leader
$\widetilde{n_g^{(1)}}, \widetilde{n_h^{(1)}}$
$\widetilde{n_g^{(c)}}, \widetilde{n_h^{(c)}}$
❺
$+\widetilde{N_g}$; $+\widetilde{N_h}$
pk_1
pk_c
pk_n
❷
$S_{1,2}$
$S_{1,n}$
$S_{c,1}$
$S_{c,c-1}$
$S_{c,c+1}$
$S_{c,n}$
$S_{n,1}$
$S_{n,n-1}$
❸
$G^{(1)} = \sum_{i \in mIDs} g_i^{(1)} + mask_g^{(1)} - r_g^{(1)}$
$H^{(1)} = \sum_{i \in mIDs} h_i^{(1)} + mask_h^{(1)} - r_h^{(1)}$
$G^{(c)} = \sum_{i \in mIDs} g_i^{(c)} + mask_g^{(c)} - r_g^{(c)}$
$H^{(c)} = \sum_{i \in mIDs} h_i^{(c)} + mask_h^{(c)} - r_h^{(c)}$
$G^{(n)} = \sum_{i \in mIDs} g_i^{(n)} + mask_g^{(n)}$
$H^{(n)} = \sum_{i \in mIDs} h_i^{(n)} + mask_h^{(n)}$
$G^{(n)} + \widetilde{N_g}$
$H^{(n)} + \widetilde{N_h}$
❻
$\sum_{c=1}^{n} G^{(c)} + k \cdot N(0, \Delta_g^2\sigma^2), \sum_{c=1}^{n} H^{(c)} + k \cdot N(0, \Delta_h^2\sigma^2)$
L_{split}
❼
update
❽

Fig. 3. Overview of FEVERLESS. - - - - : Source client broadcasts missing IDs, aggregates gradients and hessians securely, updates model and broadcasts nodes IDs. - - - - : DH key exchange and maskings generation. - - - - : Noise leader selection. ❶Broadcast missing indexes. ❷Key exchange computation. ❸Data masking. ❹Noise leader selection. ❺Global noise injection. ❻Aggregation and computation. ❼❽ Final update and broadcast updated model. Note sensitive data are in red. The maskings in ❸ protect data from the source client, and the noise in aggregated gradients and hessians prevents the source client from conducting the differential attack.

and differential attacks conducted on labels by source client and outside adversaries, we propose a privacy-preserving approach, shown in Algorithms 3, 4, and 5, to "twist" the DH key exchange, noise leader selection and secure aggregation together. This method represents a viable alternative to train XGBoost securely in DL-VFL without demanding excessive computational resources and affecting model accuracy.

To generate the secure-but-can-be-canceled-out maskings, we adopt DH here. In Algorithm 3, all clients randomly select numbers as their secret keys and generate the corresponding public keys. For any two clients in the set $\mathcal{C}$, they will exchange the public key and compute the corresponding shared keys. For simplicity, we do not describe the signature scheme for DH. We assume DH is conducted on authenticated channels, which means the man-in-the-middle attack [46] should be invalid here.

Algorithm 2. Protocol Overview

```
1:  Input: ${X_j^{(c)} | j \in f, c \in |\mathcal{C}|}$: features, ${y_i^{(c)} | i \in m, c \in |\mathcal{C}|}$: labels
2:  Output: XGBoost model
3:  Building trees:
4:  for $nt \in [1, NT]$ do
5:    for $c \in [1, n]$ do
6:      for each data entry $i$ owned by $c$ do
7:        $g_i^{(c)} \leftarrow \partial_{\hat{y}_i^{(c)}} Loss(\hat{y}_i^{(c)}, y_i^{(c)})$
8:        $h_i^{(c)} \leftarrow \partial^2_{\hat{y}_i^{(c)}} Loss(\hat{y}_i^{(c)}, y_i^{(c)})$
9:      end
10:   end
11:   for each node in the current tree do
12:     while current depth $< d$ do
13:       for $c \in [1, n]$ do
14:         for each feature $j$ owned by $c$ do
15:           for each $B_z^j$ owned by $c$ do
16:             Broadcast $mID = \{i \mid y_i \notin \mathcal{Y}^c\}$
17:           end
18:           aggregate $G$, $H$ by Algorithms 3, 4, and 5
19:           compute $L_{split}$ according to Eq. (3)
20:         end
21:         find the maximum $L_{split}^{(c)}$ and broadcast
22:       end
23:       $L_{split\_max}^{(c)} \leftarrow \max(\{L_{split}^{(c)} \mid c \in [1, n]\})$
24:       if $L_{split\_max}^{(c)} \le threshold\_L_{split}$ then
25:         set current node as leaf node
26:         $c$ computes $w$ and broadcast
27:         Break
28:       else
29:         $c$ splits the current node to the left node and right
            node and broadcasts the data index of them.
30:       end
31:     end
32:     set remaining nodes as leaf nodes
33:     $c$ computes $w$ and broadcast
34:     clients participating in calculation of $w$: update $\hat{y}_i^{(c)}$
35:   end
36: end
```

If the shared keys are used as maskings directly, our system is not robust for clients' collusion unless the amount of communication has been sacrificed as a cost to updating maskings per round. But the communication complexity is exponentially increased with the number of clients for a single node splitting. Considering the structure of trees, the overall communication complexity will be $O(2^d \cdot NT \cdot n^2)$, which may not scale well in practical applications.

Algorithm 3. Diffie-Hellman Key Exchange

```
1: for $c \in [1, n]$ do
2:   $sk_c \leftarrow \mathbb{Z}_p^*$
3: end
4: for $c \in [1, n]$ do
5:   $pk_c = g^{sk_c} \bmod p$
6:   for $c' \in [1, n] \wedge c' \neq c$ do
7:     $S_{c,c'} = pk_c^{sk_{c'}} \bmod p$
8:   end
9: end
```

To tackle this issue, we use KDF to update maskings per round automatically. Specifically, in lines 24-25 of Algorithm 5, shared keys are taken as main keys. 0 and 1 are salt values for gradients and hessians, respectively. Since the query in each round varies, the generated maskings should be dynamic accordingly. Besides, the sign of maskings is determined by the indexes of clients. In this way, we only need to use DH once, and the communication complexity is independent of tree structure.

To enable FEVERLESS to hold against differential attack, we use the GDP approach allowing the chosen one to inject a global noise to aggregated values per round. The approach is quite subtle. If the noise leader is selected by the source client, the system will be vulnerable to collusion. Moreover, a client could be easily identified as a target if we choose it in advance, e.g., by selecting a list of leaders before the training. To avoid these issues and limit the probability of collusion to the greatest extent, we use VRF to iteratively select the leader (see Algorithm 4) to securely inject a global noise. The input of VRF includes $mIDs$ and a fresh random number r (line 4 in Algorithm 4), so that this client will not be predicted and set beforehand - reducing its chance to be corrupted in advance by outsiders and the source client.

Algorithm 4. Noise Leader Selection

```
1:  count = 1
2:  for each time run this algorithm do
3:    for $c \in [1, n] \wedge c \neq$ source client do
4:      $selec_c \leftarrow \mathsf{H}(\mathtt{SIGN}_{\mathtt{sk_c}}(\mathtt{count}, \mathtt{mIDs}, \mathtt{r}))$
5:      Broadcast
6:    end
7:    $selec_c^{max} \leftarrow \max(\{selec_c \mid c \in [1, n]\})$
8:    set $c$ as noise leader
9:    count+=1
10: end
```

All clients can broadcast their scores and then the one who provides the "max value" will become the leader. Then the leader re-generates a selection score as score threshold ($selec_{threshold}$) and sends it to the rest of the clients. (line 2-6 in Algorithm 5). The clients send the masked noise back to the leader if the re-generated score is larger than the threshold (lines 7-13 in Algorithm 5). Subsequently, the leader will select $\hat{k}$ clients, notify them and aggregate this masked noise to generate a global noise with a random number. In this context, even if these selected clients are colluded (note at least one is not) with the noise leader and source client, there is still a noise that cannot be recovered, safeguarding the training differentially private. Note since the noise is masked by the random number, the source client (even colluding with the leader) cannot recover the "pure" global noise to conduct the differential attack. And each client adds a noise with a probability p. If k out of $\hat{k}$ are non-colluded, the probability of collusion is $(1-\frac{k}{n})^h$. To cancel out the randomness, the selected clients will subtract the same randomness from masked messages (line 28-31 in Algorithm 5).

Algorithm 5. Secure Aggregation With Global Differential Privacy

```
1: **Noise injection:**
2: **if** $c = leader$ **then**
3:     $selec_c^{threshold} \leftarrow \mathsf{H}\,(\mathtt{SIGN}_{\mathtt{sk_c}}(\mathtt{count}, \mathtt{mIDs}, \mathtt{r}))$
4:     Broadcast
5:     count+=1
6: **end**
7: **for** $c \in [1, n] \wedge c \neq source\ client \wedge c \neq noise\ leader$ **do**
8:     $selec_c \leftarrow \mathsf{H}\,(\mathtt{SIGN}_{\mathtt{sk_c}}(\mathtt{count}, \mathtt{mIDs}, \mathtt{r}))$
9:     **if** $selec_c > \widetilde{selec_c^{threshold}}$ **then**
10:        send $\widetilde{n_g^{(c)}} = N(0, \Delta_g^2\sigma^2) + r_g^{(c)}$ and $\widetilde{n_h^{(c)}} = N(0, \Delta_h^2\sigma^2) + r_h^{(c)}$ to noise leader
11:        count+=1
12:    **end**
13: **end**
14: **if** $c = leader$ **then**
15:    $c$ selects $k$ clients from clients of sending noise, $k = \lceil |\{\widetilde{n_g^{(c)}}\}| \cdot p \rceil$
16:    **if** $k < 1$ **then**
17:        redo noise injection
18:    **end**
19:    notify $k$ clients
20:    noise aggregation: $\widetilde{N_g} = k \cdot N(0, \Delta_g^2\sigma^2) + R_g$, $\widetilde{N_h} = k \cdot N(0, \Delta_h^2\sigma^2) + R_h$
21: **end**
22: **Secure aggregation:**
23: **for** $c \in [1, n]$ **do**
24:    $\mathtt{mask}_g^{(c)} \leftarrow (\sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot (\mathsf{H}(S_{c,c'}\|0\|\mathtt{query}) \bmod N)) \bmod N$
25:    $\mathtt{mask}_h^{(c)} \leftarrow (\sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot (\mathsf{H}(S_{c,c'}\|1\|\mathtt{query}) \bmod N)) \bmod N$
26:    $G^{(c)} = \sum_{i \in mIDs} g_i^{(c)} + \mathtt{mask}_g^{(c)} \bmod N$
27:    $H^{(c)} = \sum_{i \in mIDs} h_i^{(c)} + \mathtt{mask}_h^{(c)} \bmod N$
28:    **if** $selec_c > selec_c^{threshold} \wedge received\ notification$ **then**
29:        $G^{(c)} = G^{(c)} - r_g^{(c)} \bmod N$
30:        $H^{(c)} = H^{(c)} - r_h^{(c)} \bmod N$
31:    **end**
32:    **if** $c = leader$ **then**
33:        $G^{(c)} = G^{(c)} + \widetilde{N_g} \bmod N$
34:        $H^{(c)} = H^{(c)} + \widetilde{N_h} \bmod N$
35:    **end**
36:    send $\{G^{(c)}, H^{(c)}\}$ to source client
37: **end**
```

Considering that the source client may procrastinate the leader selection and noise injection procedure so as to buy some time for its colluded clients to prepare sufficient large VRF values to participate in the competition of selection and adding noise. One may apply a heartbeat protocol [47] to prevent a newly elected leader from intentionally halting the noise-adding stage for a long period, say 1 min. If there is no response from the leader after for a short while, a new leader will be randomly selected. Furthermore, the heartbeat may help to solve the problem that the leader accidentally drops from the network. We note that the heartbeat protocol is not our main focus in this paper.

Before replying to the source client, we have the clients with labels put maskings on gradients and hessians, and for those without labels, they just generate and later send out maskings, in which the noise leader (i.e. one of the maskings generators) injects the noise. In this way, the maskings, guaranteeing perfect secrecy of the messages, will be canceled out after the aggregation of the values, and the differentially private noise will consolidate indistinguishability of individual data entry.

Note that in lines 24-34 of Algorithm 5, the maskings and masked values are in the range $[0, N-1]$. And N should be sufficiently large to avoid overflow, and the summation of gradients and hessians should not exceed N.

4.4 Theoretical Analysis

Computation cost: We use B and d to denote the number of buckets and the maximum depth respectively, and $f^{(c)}$ here represents the number of features held by a client c. For each client c, the computation cost can be divided into 4 parts: (1) Performing at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times computation of L_{split} and w, taking $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$ time; (2) Creating $n-1$ shared keys and 1 public key, which is $O(n)$; (3) Conducting $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$ time to compute VRF outputs, select noise leader and generate noise; (4) Generating $2f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ maskings, which takes $O(f^{(c)} \cdot B \cdot NT \cdot 2^d \cdot n)$ time. Overall, each client's computation complexity is $O(f^{(c)} \cdot B \cdot NT \cdot 2^d \cdot n)$.

Communication cost: Each client's communication cost can be calculated as (1) Broadcasting at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times of missing indexes mID; (2) Broadcasting 1 public key and receiving $n-1$ public keys from other clients; (3) Broadcasting 1 leader selection score and sending noise to noise leader at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times; (4) Sending source client 2 masked gradients and hessians of size $2\lceil log_2 N \rceil$. Therefore the overall communication cost is $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1) \cdot (\|mID\| \cdot \alpha_I + \alpha_L + \alpha_N + n \cdot \alpha_K 2\lceil log_2 N \rceil)$, where $\alpha_I, \alpha_L, \alpha_N$ and α_K refer to the number of bits of index, leader selection score, noise and public keys, respectively. Thus, we have the communication complexity $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$.

4.5 Security Analysis

We show that FEVERLESS provides label value and data privacy against an adversary controlling at most $n-2$ clients in the *semi-honest* setting [48]. Here, we provide a brief summary of analysis and theorems. The formal proofs, in the random oracle model, are given in Section 1 of Supplementary.

Label Value Privacy: This implies that the value of a label among honest parties should not be leaked to the adversary.

We achieve this by using a secure aggregation mechanism where the masks are created via DH key exchange and KDF. In brief, we show that because of the Decisional DH problem (see Definition 2.1), the adversary cannot distinguish the individual values from randomly chosen ones. That is why the adversary $\mathcal{A}$ cannot learn the owner of the label.

Data Privacy: FEVERLESS provides data privacy, meaning that an adversary $\mathcal{A}$ cannot extract the features of training data and key shares of any honest party. Individual key shares are not separable from random values because of the secure masking. Since the calculations of gradients or hessians are irrelevant with features of training data, the adversary cannot infer features even if gradients are breached. If the source client is not part of the adversary, no data information is leaked. But we require an additional countermeasure for the case where the source client is part of the adversary because it can collect the summation of the data values. We use differential privacy [37], [38] to achieve data privacy. Because of the noise added by differential privacy, the adversary cannot learn the individual data of an honest client. Moreover, we select the noise clients by the VRF which ensures that the noise leader cannot be predicted or compromised in advance.

Theorem 4.1 ($\mathcal{A}$ not including source client). *There exists a Probabilistic Polynomial Time (PPT) simulator* Sim *for all* $|\mathcal{C}| := n \geq 3$, $|\mathcal{X}| := f \geq n$, $|\mathcal{Y}| := m \geq 1$, $\bigcup_{c\in\mathcal{C}}\mathcal{X}^{(c)}$, $\bigcup_{c\in\mathcal{C}}\mathcal{Y}^{(c)}$ *and* $\mathcal{A} \subset \mathcal{C}$ *so that* $|\mathcal{A}| \leq n-2$, *the output of* Sim *is indistinguishable from the output of* REAL: $\mathtt{REAL}_{\mathcal{A}}^{\mathcal{C},\mathcal{X},\mathcal{Y}}(\mathcal{X}^{\mathcal{C}}, \mathcal{Y}^{\mathcal{C}}) \equiv \mathsf{Sim}_{\mathcal{A}}^{\mathcal{C},\mathcal{X},\mathcal{Y}}(\mathcal{X}^{\mathcal{A}}, \mathcal{Y}^{\mathcal{A}})$.

Theorem 4.2 ($\mathcal{A}$ including source client). *There exists a Probabilistic Polynomial Time (PPT) simulator* Sim *for all* $|\mathcal{C}| := n \geq 3, |\mathcal{X}| := f \geq n, |\mathcal{Y}| := m \geq 1, \bigcup_{c\in\mathcal{C}}\mathcal{X}^{(c)}, \bigcup_{c\in\mathcal{C}}\mathcal{Y}^{(c)}$ *and* $\mathcal{A} \subset \mathcal{C}$ *so that* $|\mathcal{A}| \leq n-2$, *the output of* Sim *is indistinguishable from the output of* REAL:$\mathtt{REAL}_{\mathcal{A}}^{\mathcal{C},\mathcal{X},\mathcal{Y}}(\mathcal{X}^{\mathcal{C}}, \mathcal{Y}^{\mathcal{C}}) \equiv \mathsf{Sim}_{\mathcal{A}}^{\mathcal{C},\mathcal{X},\mathcal{Y}}(G, H, \mathcal{X}^{\mathcal{A}}, \mathcal{Y}^{\mathcal{A}})$ *where* $G = \sum_{i\in mIDs} g_i^{(c)} + N(0, (\Delta_g\sigma)^2)$, $H = \sum_{i\in mIDs} h_i^{(c)} + N(0, (\Delta_h\sigma)^2)$.

Theorem 4.3 (Privacy of the Inputs). *No* $\mathcal{A} \subset \mathcal{C}$ *such that* $|\mathcal{A}| \leq n-2$ *can retrieve the individual values of the honest clients with probability* $1 - \sum_{i=0}^{\hat{k}} \binom{h}{i} \binom{n-2-h}{\hat{k}-i} (P_t)^{\hat{k}} (1-P_t)^{(n-\hat{k})} \left(\binom{\hat{k}-i}{k} / \binom{\hat{k}}{k}\right)$, *where* h *and* $\hat{k}$ *refer to the number of non-colluded clients and the number of clients who have selection score larger than threshold, respectively; and* P_t *is the probability of selection score larger than the threshold.*

Note for a concrete example, if we set $n = 10, h = 2, \hat{k} = 5, k = 8, P_t = \frac{1}{2}$, the probability is 0.938. This means the source client cannot remove the noise with 0.938, which is a relatively high probability.

5 Experiment

We perform evaluations on the accuracy, runtime performance and communication cost, and compare our design with two straightforward secure approaches: one is based on LDP (for accuracy), and the other is built on AHE with GDP (for runtime). These approaches are most-commonly-used components for privacy-preserving FL, and they could be the building blocks for complex mechanisms, e.g., MPC. We note the protocol should intuitively outperform those MPC-based solutions, and one may leverage our source code to make further comparisons if interested. In the experiments, the baseline, which is the pure XGBoots algorithm, follows the training process of Fig. 3 without using any privacy-preserving tools (steps ❷ - ❺). And LDP does not conduct DH key exchange but each client injects noise into the aggregation of gradients and hessians, while AHE follows Fig. 3 except executing DH key exchange. In AHE, each client sends (additive) encrypted messages to the source client after step ❺. We here show the performance of the best case where there is only one (non-colluded and randomly selected) client adding noise per round ($k = 1$).

5.1 Experiment Setup

All the experiments are implemented in Python, and conducted on a cluster of machines with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz, with 15 GB RAM in a local area network. As for the cryptographic tools, we set the key size of DH and Paillier as 160 bits and 1024 bits respectively(to save some time in running the experiments). This size can reach a symmetric security level with 80 bits key length. Note one may indeed increase the key size to obtain stronger security,[4] but this will bring a longer experiment time as a side effect. We use 1024-bit MODP Group with 160-bit Prime Order Subgroup from *RFC 5114*[5] for DH Key exchange. `SHAKE-256` [49], a member of `SHA3` [49] family, is used as a hash function in leader selection and secure aggregation.

Intuitively, the smaller ϵ we set, the more secure FEVERLESS will be; but larger noise will be added. We note the above statement can be seen from the experimental results. To present comprehensive results on the accuracy, we set ϵ to be: 10, 5, 2 and 1, and δ is set to 10^{-5}. In terms of accuracy and runtime, we evaluate different situations by varying the number of clients, the number of trees, and the maximum depth of trees (from 2 to 10). Other parameters regarding training follow the suggestions in [18] and the *library*[6] of XGBoost. To deliver fair results, we conduct each test for 20 independent trials and then calculate the average.

Datasets. We run the experiments on three datasets - Credit Card [50], Bank Marketing [51] and Banknote Authentication[7] - for classification tasks. Because the more concentrated the distribution of labels and features, the more like centralized learning. The entire algorithm requires less interaction among clients. This situation is less common in practical applications. To fairly investigate the model performance in DL-VFL, we make the features and labels as sparse as possible, and they are uniformly distributed among clients.

• *Credit Card:* It is a commercial dataset used for predicting whether customers will make payments on time. It provides 30,000 samples, and each sample composes of 23 features.

4. Note a stronger security level will not affect the training accuracy.
5. https://tools.ietf.org/html/rfc5114
6. https://xgboost.readthedocs.io/
7. https://archive.ics.uci.edu/ml/datasets/banknote+authentication

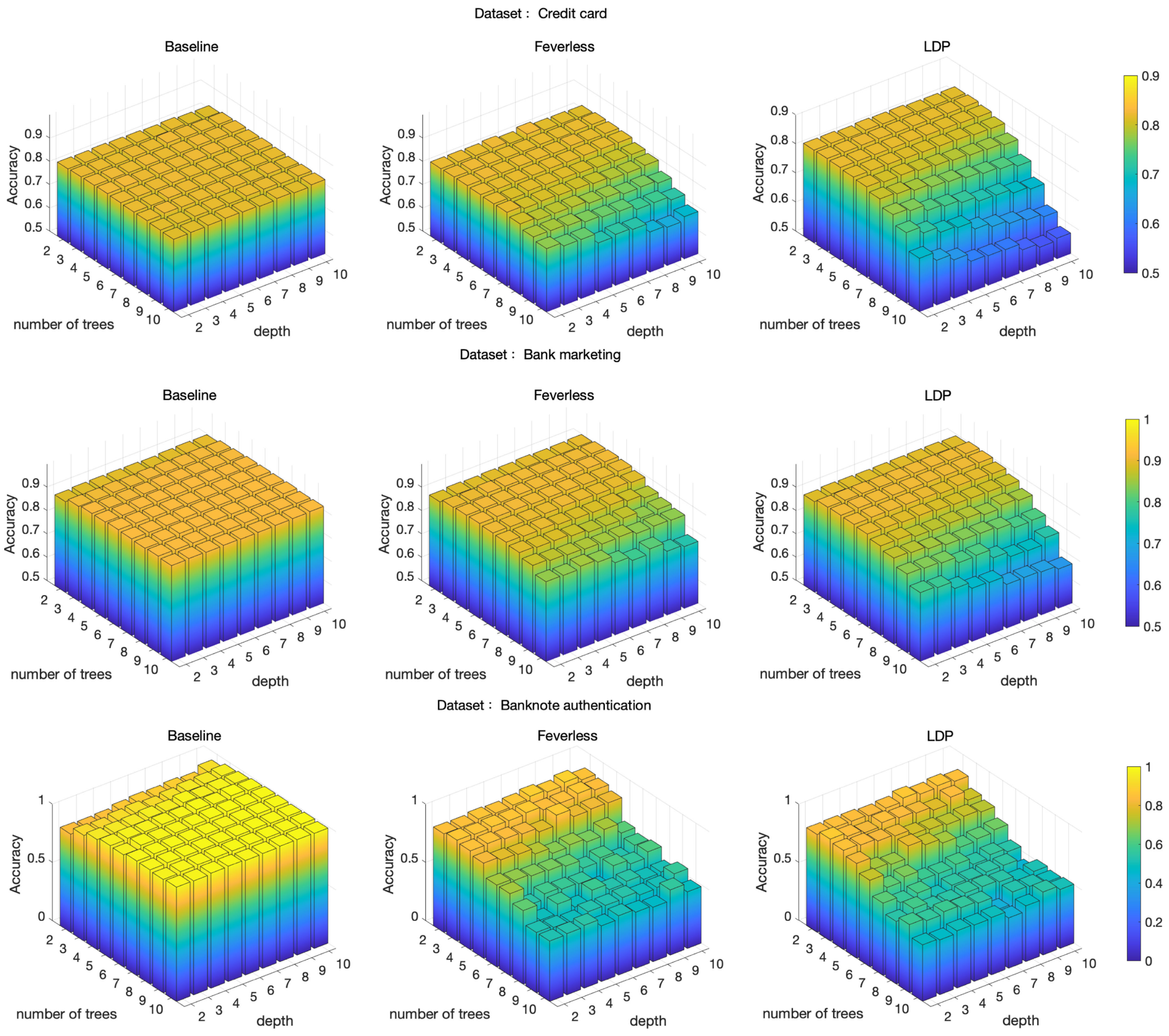


Fig. 4. Comparison among the baseline, FEVERLESS and LDP under $\epsilon = 2$. *Top row:* Credit card dataset, accuracy range: [0.5, 0.9]. *Middle row:* Bank marketing, accuracy range: [0.5, 1]. *Bottom row:* Banknote authentication, accuracy range: [0, 1].

• *Bank marketing:* Consisting of 45,211 data points and 17 features, the goal of bank marketing is to predict if a client will subscribe to a term deposit.

• *Banknote authentication:* Offering 1,372 data points and 4 features, this dataset is used to classify authenticated and unauthenticated banknotes. Note that different from traditional tabular data, features in the dataset are extracted from images that are taken from genuine and forged banknote-like specimens through Wavelet Transform [52]. Using the small-scale dataset, the trained model may not be robust to noise, which brings a negative impact on accuracy.

5.2 Evaluation on Accuracy

In Fig. 4, we present a clear picture of the accuracy performance based on the #tree and the maximum depth in $(2, 10^{-5})-$DP. We merge the #client in one tree structure, which means in one bar, and the value is the mean of accuracy when conducting on different numbers. The accuracy of the baseline in credit card (about 0.82) and bank marketing (nearly 0.9) remains unchanged as the #tree and maximum depth increases, while the accuracy in banknote authentication rises from 0.9 to approximately 1.0. To highlight the differences and ensure all results are displayed clearly, we set the ranges of accuracy as $[0.5, 0.9]$, $[0.5, 1]$ and $[0, 1]$ for the three datasets, respectively.

Compared with the baseline, shown in the top and middle rows of Fig. 4, FEVERLESS and LDP suffer from continuously shrinking accuracy as tree structure becomes complex. This is so because the injected noises are accumulated into the model via the increase of query number. And the accuracy is easily affected by the depth. In the worst case where the #tree and maximum depth are both equal to 10, FEVERLESS decreases 10.37% (resp. 14.98%), and LDP drops 24.78% (resp.24.59%) in credit card (resp. bank marketing). But on average, FEVERLESS' accuracy only shrinks by around 0.9% (resp. 3.21%), while LDP suffers from an

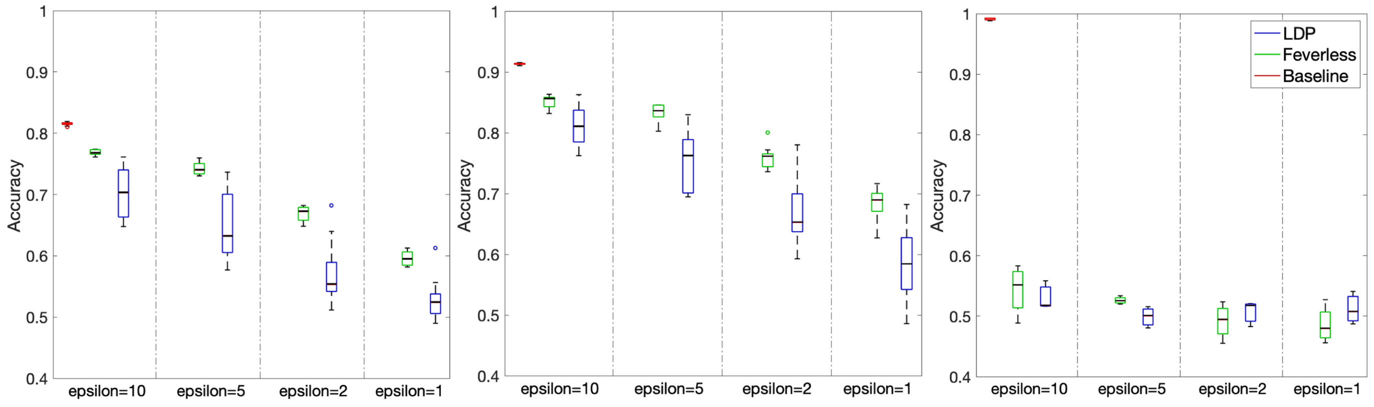


Fig. 5. Comparison of accuracy by varying ϵ in depth =10, the number of trees =10. *Left:* Credit card. *Middle:* Bank marketing. *Right:* Banknote authentication. Accuracy ranges from 0.4 to 1.

estimated 3x (resp. 2x) accuracy loss. The difference in the degree of deterioration mainly comes from how much noise is added for each query. We note the deterioration of FEVERLESS is independent of the #client. Thus, we can maintain great accuracy even in the case where there exists a considerable amount of clients.

Despite the fact that less noise is added in FEVERLESS, we do not predict that the accuracy falls to the same level (around 50%, like randomly guessing in binary classification) as LDP in the bottom row of Fig. 4. This is so because the model is trained by an extremely small-size dataset, which makes it hard to maintain the robustness but relatively sensitive to noise. If setting a larger ϵ, we may see our advantage more clearly.

To distinguish the performance between FEVERLESS and LDP more clearly, Fig. 5 shows the comparison over different ϵ, when #depth and #tree are set to 10. The performance of the model is decayed as the decrease of ϵ. In the left (resp. middle) of Fig. 5, the averaged accuracy of FEVERLESS falls from 0.7686 to 0.5967 (resp. from 0.8517 to 0.6831), while that of LDP also decreases to 0.5299 (resp. 0.5853). We notice that the highest values of LDP stay at the same level as those of FEVERLESS. This is because, in the case of 2-client training, only one client needs to add the noise in LDP (which is identical to our GDP solution). At last, the worse case can be seen on the right of Fig. 5 due to the weak robustness of the model obtained from the banknote authentication. The results are far away from the baseline there. This is because in small-scale datasets, the heterogeneity of data distribution is not large, so the original XGBoost can achieve high accuracy. However, the model trained in this way is less robust, which means it is more sensitive to noise. Therefore, compared with the model trained on a large-scale dataset, it does not perform well under the condition of differential privacy. But even in this case, FEVERLESS still holds a tiny advantage over LDP.

Note that we did not compare the accuracy to systems using AHE. Because the calculation process of homomorphic encryption does not change the precision of the value, training through encryption will not affect the model. Therefore, the accuracy of using AHE is the same as the pure XGBoost.

5.3 Evaluation on Training Time

To highlight the runtime complexity, we average the results varying by client number into one tree structure as well. We further set the ranges of time as [0 s, 9,500s], [0 s, 3,500s] and [0 s, 110s] for the datasets to deliver visible results. Note since the banknote dataset contains the least samples, it does deliver the best training efficiency here. Fig. 6 presents the comparison of the training time by varying maximum depth and the number of trees among the datasets.

The training time increases exponentially and linearly with depth and the number of tree, which is consistent with our analysis given in Section 4.4. In Fig. 6, compared with the baseline, the runtime of FEVERLESS at most increases 110.3 s (resp. 50 s, 4.3 s), while AHE requires around 70x spike (resp. 48x, 21x) in credit card (resp. bank marketing, banknote authentication), where #depth and #trees are equal to 10. For the average case, FEVERLESS consumes Approx. $1\%(resp. 6.5\%, 13.96\%)$ more training time than the baseline, while AHE requires the $351\%(resp. 155.1\%, 674\%)$ extra, w.r.t. the three datasets. Its poor performances are due to the laborious calculations in encryption, in which each client has to conduct an encryption per query. By contrast, the maskings in FEVERLESS avoid these excessive costs. We further investigate the runtime performance on the #client in Section 3 of Supplementary material.

5.4 Evaluation on Communication Cost

In Figs. 7, 8, and 9, we demonstrate the communication cost based on the number of clients, tree and depth. For the convenience of comparison, we set #clients=4, #tree=4 and depth=4 as default. To sum up, we see that the communication cost of FEVERLESS is almost the same as those of the baseline and LDP. But as compared to AHE, FEVERLESS significantly reduces the cost while maintaining privacy.

In each presented figure, we show the results executed on the datasets Credit card (left), Bank Marketing (middle) and Banknote Authentication (right). Note that the comparison among FEVERLESS, LDP, and AHE requires a condition that #client=2; when #client=1, we can only show the results of the baseline. Via the experiments, we elaborate that how the communication cost varies with the increasing number of clients, depth and the number of trees among the baseline, FEVERLESS, AHE and LDP. In general, adding noise has no clear impact on communication costs. The performance of FEVERLESS and LDP is on par with that of the baseline. The AHE approach does harm communication

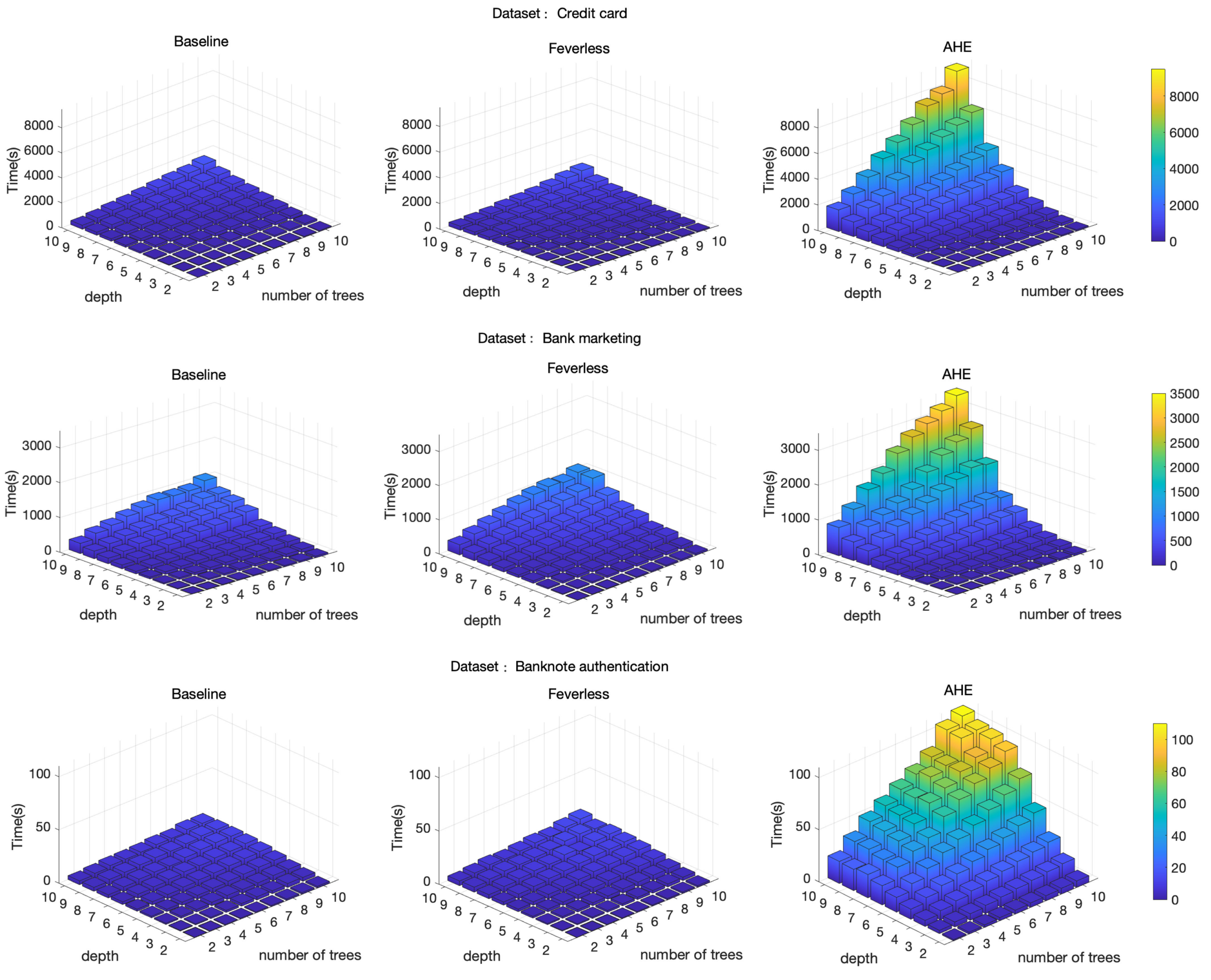


Fig. 6. Comparison of time. *Top row:* Credit card dataset, range: [0 s, 9,500s]. *Middle row:* Bank marketing, range: [0 s, 3,500s]. *Bottom row:* Banknote authentication, range: [0 s, 110s].

costs, which can be seen from the continuously and significantly increasing bars in the figures. Naturally, when more clients engage in the training, more communication costs should be added to the model. Especially, in the number of clients equal to 4, the communication costs of AHE is around 6*1e6 Bytes in Banknote Authentication dataset, which is about 3x than other methods. Similar situations can be observed when training with complex tree structures. In depth (*resp.* the number of trees) equals 10, the communication costs of AHE reaches about 1.3*1e7 Bytes (*resp.* 1.5*1e7 Bytes), which is 2.6x (*resp.* 2.4x) than other methods. AHE generates such a large amount of communication costs because it requires transmitting ciphertexts during interactions among clients.

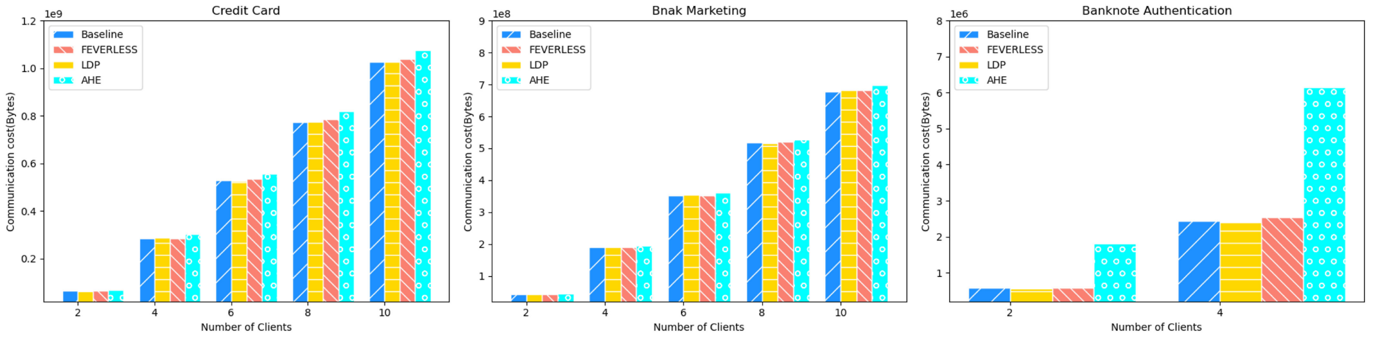


Fig. 7. Comparison of communication cost on the number of clients.

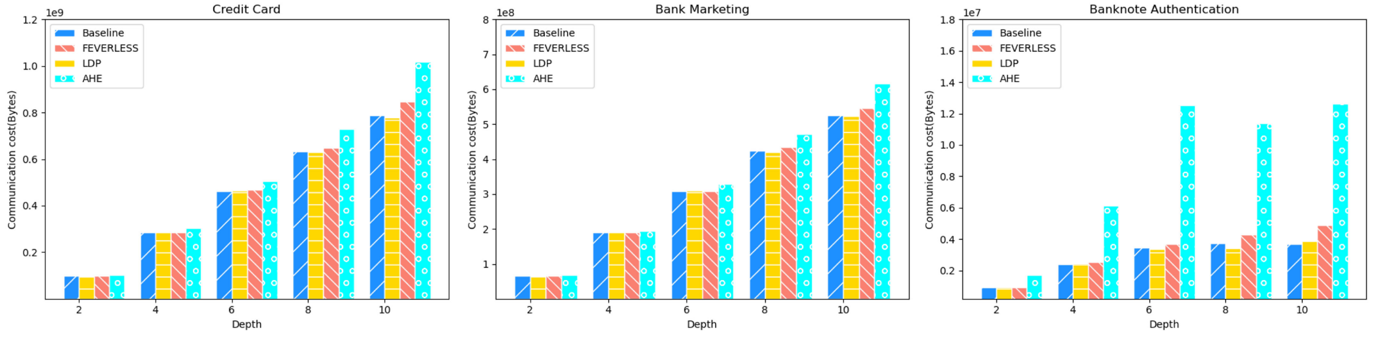


Fig. 8. Comparison of communication cost on depth.

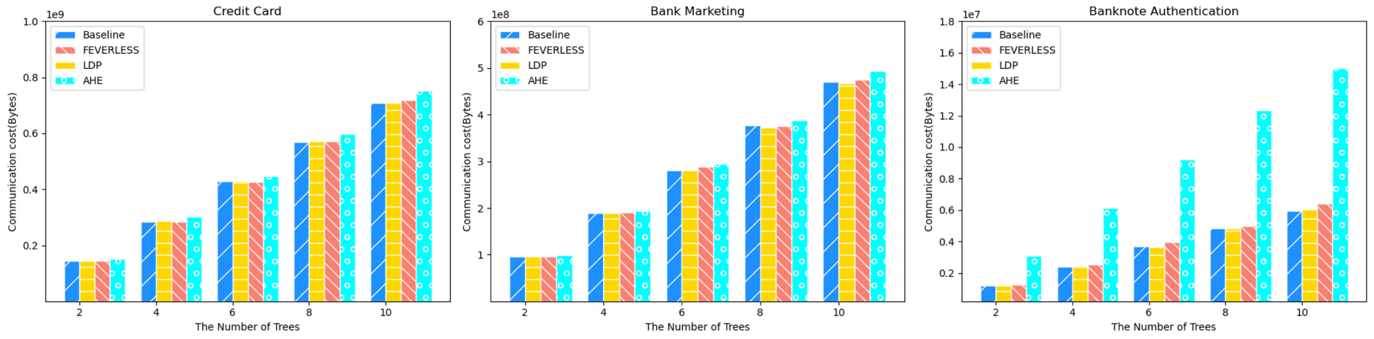


Fig. 9. Comparison of communication cost on the number of trees.

6 Conclusion and Future Work

We consider a practical scenario where labels are distributedly and maintained by different clients for VFL. By leveraging secure aggregation and GDP, we present a novel system, FEVERLESS, to train XGBoost securely. FEVERLESS can achieve perfect secrecy for labels and data, and adversaries cannot learn any information about the data even if the source client is corrupted. With DP against differential attack, the source client knows nothing more than summation. Our design is also robust for the collusion of $n-2$ out of n clients. FEVERLESS is about the same speed and accuracy as the pure XGBoost, taking 1% extra runtime, and sacrificing 0.9% accuracy. In Section 2 of Supplementary material, we discuss how to reduce noise, hide label tagging information and use other security tools. Although our system achieves great performance in terms of security and efficiency, its accuracy still does not work well in small-scale datasets. This remains an open problem. We will also consider secure solutions against malicious adversaries.

Acknowledgments

Y. Jin is funded by an Alexander von Humboldt Professorship for AI endowed by the German Federal Ministry of Education and Reseaech.

References

[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019, Art. no. 12.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[3] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321, doi: 10.1145/2810103.2813687.

[4] T. Orekondy, S. J. Oh, Y. Zhang, B. Schiele, and M. Fritz, “Gradient-leaks: Understanding and controlling deanonymization in federated learning,” 2018, *arXiv: 1805.05838*.

[5] J. Geiping, H.H. BauermeisterDröge, and M. Moeller, “Inverting gradients - how easy is it to break privacy in federated learning?,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 16 937–16 947. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf

[6] H. Li and T. Han, “An end-to-end encrypted neural network for gradient updates transmission in federated learning,” 2019, *arXiv: 1908.08340*.

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

[8] S. Truex et al., “A hybrid approach to privacy-preserving federated learning,” in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 1–11, doi: 10.1145/3338501.3357370.

[9] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, “HybridAlpha: An efficient approach for privacy-preserving federated learning,” in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 13–23, doi: 10.1145/3338501.3357371.

[10] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, “BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning,” in *Proc. USENIX Annu. Tech. Conf. USENIX Assoc.*, 2020, pp. 493–506, [Online]. Available: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang

[11] H. Zhu, R. Wang, Y. Jin, K. Liang, and J. Ning, “Distributed additive encryption and quantization for privacy preserving federated deep learning,” 2020, *arXiv:2011.12623*.

[12] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, “Privacy preserving vertical federated learning for tree-based models,” *Proc. VLDB Endow*, vol. 13, no. 12, pp. 2090–2103, Jul. 2020, doi: 10.14778/3407790.3407811.

[13] K. Cheng et al., “SecureBoost: A lossless federated learning framework,” *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, 2021.

[14] S. Hardy et al., “Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption,” 2017, *arXiv:1711.10677*.

[15] R. Nock et al., “Entity resolution and federated learning get a federated resolution,” 2018, *arXiv:1803.04035*.

[16] Y. Liu, X. Zhang, and L. Wang, “Asymmetrically vertical federated learning,” 2020, *arXiv:2004.07427*.

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

[18] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge, MA, USA: MIT Press Cambridge, 2016.

[20] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[21] O. Goldreich, “Secure multi-party computation,” *Manuscript Preliminary Version*, vol. 78, pp. 639–648, 1998.

[22] K. Bonawitz et al., “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191, doi: 10.1145/3133956.3133982.

[23] C. Dwork, “Differential privacy: A survey of results,” in *Theory and Applications of Models of Computation*, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds., Berlin, Germany: Springer, 2008, pp. 1–19.

[24] Z. Tian, R. Zhang, X. Hou, J. Liu, and K. Ren, “FederBoost: Private federated learning for GBDT,” 2020, *arXiv: 2011.02796*.

[25] X. Meng et al., “MLlib: Machine learning in apache spark,” *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[26] D. Boneh, “The decision diffie-hellman problem,” in *International Algorithmic Number Theory Symposium*. Berlin, Germany: Springer, 1998, pp. 48–63.

[27] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[28] G. Ács and C. Castelluccia, “I have a dream! (differentially private smart metering),” in *Proc. Int. Workshop Inf. Hiding*, 2011, pp. 118–132.

[29] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, “A pseudorandom generator from any one-way function,” *SIAM J. Comput.*, vol. 28, no. 4, pp. 1364–1396, 1999.

[30] N. Sha, “standard: Permutation-based hash and extendable-output functions,” *Federal Inform. Process. Stand. Pub.*, vol. 3, p. 202, 2015.

[31] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, “Sha-3 proposal blake,” *Submission NIST*, vol. 92, 2008.

[32] H. Krawczyk and P. Eronen, “HMAC-based extract-and-expand key derivation function (HKDF),” RFC 5869, May, 2010.

[33] B. Kaliski, *Pseudorandom Function*. Boston, MA, USA: Springer, 2005, pp. 485–485, doi: 10.1007/0-387-23483-7_329.

[34] J. Zdziarski, *Hacking and Securing iOS Applications: Stealing Data, Hijacking Software, and How to Prevent It*. Sebastopol, CA, USA: O’Reilly Media, Inc., 2012.

[35] S. Micali, M. Rabin, and S. Vadhan, “Verifiable random functions,” in *Proc. 40th Annu. Symp. Foundations Comput. Sci.*, 1999, pp. 120–130.

[36] S. Micali, “ALGORAND: The efficient and democratic ledger,” *CoRR*, vol. abs/1607.01341, 2016, *arXiv:1607.01341*

[37] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, “Our data, ourselves: Privacy via distributed noise generation,” in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2006, pp. 486–503.

[38] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 265–284.

[39] M. Abadi et al., “Deep learning with differential privacy,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[40] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, “Practical multi-party private set intersection from symmetric-key techniques,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1257–1272.

[41] B. Pinkas, T. Schneider, and M. Zohner, “Faster private set intersection based on {OT} extension,” in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 797–812.

[42] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Proc. Annu. Cryptol. Conf.*, 2012, pp. 643–662.

[43] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Germany: Springer, 1999.

[44] P. Kairouz, S. Oh, and P. Viswanath, “Extremal mechanisms for local differential privacy,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2879–2887.

[45] K. Wei et al., “Federated learning with differential privacy: Algorithms and performance analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3454–3469, Apr. 2020.

[46] A. S. Khader and D. Lai, “Preventing man-in-the-middle attack in diffie-hellman key exchange protocol,” in *Proc. 22nd Int. Conf. Telecommun.*, 2015, pp. 204–208.

[47] S. Nikoletseas and J. D. Rolim, *Theoretical Aspects of Distributed Computing in Sensor Networks*. Berlin, Germany: Springer, 2011.

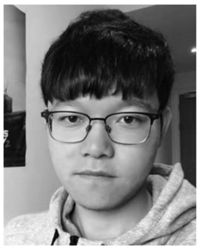
[48] N. P. Smart, *Cryptography Made Simple*. Berlin, Germany: Springer, 2016.

[49] M. J. Dworkin, “SHA-3 standard: Permutation-based hash and extendable-output functions,” Federal Inf. Process. Stds. (NIST FIPS), Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, 2015.

[50] I.-C. Yeh and C.-H. Lien, “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients,” *Expert Syst. Appl.*, vol. 36, no. 2, pp. 2473–2480, 2009.

[51] S. Moro, P. Cortez, and P. Rita, “A data-driven approach to predict the success of bank telemarketing,” *Decis. Support Syst.*, vol. 62, pp. 22–31, 2014.

[52] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, “Image coding using wavelet transform,” *IEEE Trans. image Process.*, vol. 1, no. 2, pp. 205–220, Apr. 1992.



Rui Wang received the BSc degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017, and the MSc degree from the University of Southampton, Southampton, U.K., in 2018. He is currently working toward the PhD degree in focusing on privacy-preserving machine learning with the Department of Intelligent Systems, Delft University of Technology, Delft, the Netherlands.

Oğuzhan Ersoy received the BSc and MSc degrees in electrical and electronics engineering from Boğaziçi University, in 2012 and 2015, respectively, and the PhD degree from the Delft University of Technology, The Netherlands, in 2021, where he was also a postdoctoral researcher. His PhD and postdoctoral study focused on secure, scalable, and incentive-compatible protocols for Bitcoin-like blockchains. Currently, he is a postdoctoral researcher with the Digital Security Group, Radboud University, The Netherlands. His research interests include blockchain technology, machine learning, and applied cryptography.

Hangyu Zhu received the BSc degree from Yangzhou University, Yangzhou, China, in 2015, the MSc degree from RMIT University, Melbourne, VIC, Australia, in 2017, and the PhD degree from the University of Surrey, Guildford, U.K., in 2021. His main research interests include federated learning, privacy-preserving machine learning, and evolutionary federated neural architecture search.

Yaochu Jin (Fellow, IEEE) is an alexander von Humboldt professor of artificial intelligence with the Faculty of Technology, Bielefeld University, Bielefeld, Germany. He is also a distinguished chair professor of computational intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K. He was a finland distinguished professor in Finland, and changjiang distinguished visiting professor, in China. His research interests include evolutionary optimization, evolutionary and multiobjective machine learning, secure and privacy-preserving machine learning, and evolutionary developmental approaches to artificial intelligence. He is currently the editor-in-chief for the *IEEE Transactions on Cognitive and Developmental Systems and Complex and Intelligent Systems*. He was an IEEE Distinguished Lecturer (during 2013, 2015, and 2017–2019) and was the vice president for Technical Activities of the IEEE Computational Intelligence Society (during 2014–2015).He was the recipient of the 2015, 2017, and 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award, and the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award. He was named a Highly Cited Researcher by the Web of Science Group for 2019-2021. He is a member of Academia Europaea.

Kaitai Liang (Member, IEEE) received the PhD degree from the Department of Computer Science, City University of Hong Kong, Hong Kong. He joined the Delft University of Technology, The Delft, the Netherlands, in 2020. Before that he was an Assistant Professor of secure systems with the Department of Computer Science, University of Surrey, Guildford, U.K. His research interests include applied cryptography and information security; in particular, data encryption, blockchain security, postquantum crypto, privacy enhancing technology, and privacy-preserving machine learning.