2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00017

Boosting Privately: Federated Extreme Gradient Boosting for Mobile Crowdsensing

Yang Liu[1], Zhuo Ma*[1], Ximeng Liu*[2], Siqi Ma[3], Surya Nepal[3], Robert.H Deng[4], Kui Ren[5]
[1] Xidian University, [2] Fuzhou University, [3] Data 61, CSIRO, [4] Singapore Management University, [5] Zhejiang University
bcds2018@foxmail.com, mazhuo@mail.xidian.edu.cn, snbnix@gmail.com,
siqi.ma@csiro.au, surya.nepal@csiro.au, robertdeng@smu.edu.sg, kuiren@zju.edu.cn
* Corresponding Author

***Abstract*—Recently, Google and other 24 institutions proposed a series of open challenges towards federated learning (FL), which include application expansion and homomorphic encryption (HE). The former aims to expand the applicable machine learning models of FL. The latter focuses on who holds the secret key when applying HE to FL. For the naive HE scheme, the server is set to master the secret key. Such a setting causes a serious problem that if the server does not conduct aggregation before decryption, a chance is left for the server to access the user's update. Inspired by the two challenges, we propose FEDXGB, a federated extreme gradient boosting (XGBoost) scheme supporting forced aggregation. FEDXGB mainly achieves the following two breakthroughs. First, FEDXGB involves a new HE based secure aggregation scheme for FL. By combining the advantages of secret sharing and homomorphic encryption, the algorithm can solve the second challenge mentioned above, and is robust to the user dropout. Then, FEDXGB extends FL to a new machine learning model by applying the secure aggregation scheme to the classification and regression tree building of XGBoost. Moreover, we conduct a comprehensive theoretical analysis and extensive experiments to evaluate the security, effectiveness, and efficiency of FEDXGB. The results indicate that FEDXGB achieves less than 1% accuracy loss compared with the original XGBoost, and can provide about 23.9% runtime and 33.3% communication reduction for HE based model update aggregation of FL.**

***Index Terms*—Privacy-Preserving, Federated Learning, Extreme Gradient Boosting, Mobile Crowdsensing**

I. INTRODUCTION

Extreme gradient boosting (XGBoost) is a state-of-the-art machine learning model that performs well in processing both classification and regression tasks. Winning 17 out of 29 challenges published by the world-famous Kaggle competition validates that XGBoost is sure to have an impressive prospect for the further development of artificial intelligence [1]. Similar to other machine learning models, the performance of XGBoost depends on how well the training dataset performs. However, creating a large dataset requires lots of human efforts, which is an unaffordable cost for most companies. Hence, mobile crowdsensing is designed to collect data from mobile users who are willing to share data. DroidNet [2] is a sample system to demonstrate how mobile crowdsensing is used in machine learning model training. However, the past crowdsensing architecture usually allows the central server to access to the plaintext user's data, which leaves a chance for user privacy leakage. The incident of Facebook-Cambridge Analytica happened in 2018, is a significant example to demonstrate the consequences of such privacy leakage. The large IT company secretly harvested millions of private user data and use them to control a country's leadership election [3].

To address the privacy leakage problem for mobile crowdsensing, federated learning is proposed by Google and rapidly attract exploded interests of researchers [4]. Federated learning groups mobile users and the central server into a loose federation, and then, proceeds model training without uploading private user data to the central server. Despite its excellent features on security and performance, federated learning is still a developing technique. In 2019, Google, in conjunction with 24 other agencies, proposes a series of open challenges for the future development of federated learning [5]. Besides expanding the applicable machine learning model for federated learning, Google points out that homomorphic encryption (HE) can be a powerful tool in federated learning. However, existing HE based FL schemes [6]–[8] still have the following two unresolved challenges.

- **Forced Aggregation.** The native applications of HE to federated learning is that the server encrypts the parameters with its own public key and sends them to the user [6], [7]. Utilizing the homomorphism of HE, the user can compute the model update without decryption and return the encrypted model update to the server for aggregation. A key challenge here is to force aggregation on the server before decryption, as otherwise, the server may be able to learn a user's model update.
- **User Dropout.** Federated learning is originally designed to run in the open network, in which the user's connectivity is always unstable. To date, most of the existing HE based federated learning schemes cannot resolve the accident user dropout problem, other than abandoning the current round of training [6]–[8]. Such a drawback dramatically reduces the practicality of the schemes in applications.

To resolve the above challenges, we propose FEDXGB, a federated extreme gradient boosting framework for mobile crowdsensing that supports forced aggregation, and is robust against user dropout. FEDXGB is composed of two kinds of entities, a central cloud server and a set of users. FEDXGB proceeds as follows. The central server iteratively invokes a

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00017

suite of secure protocols to build the classification and regression tree (CART) of XGBoost. In the protocols, our newly designed secure aggregation protocol is invoked to aggregate the users' gradients. By combining Bresson's cryptosystem [9] and Shamir's secret sharing [10], FEDXGB makes the central server to operate a forced aggregation on the gradients and can recover the data of the dropout users.

Our contributions can be summarised as follows:

- **Federated Extreme Gradient Boosting.** We propose a federated learning framework to implement privacy-preserving XGBoost training for mobile crowdsensing, called FEDXGB. Using a suite of secure protocols, FEDXGB allows multiple users to cooperatively train an XGBoost model without direct revealing of their private data to the central server.
- **Forced Aggregation.** We design a new secure gradient aggregation algorithm for federated learning, which combines the advantages of both homomorphic encryption and secret sharing. Specifically, through the combination of homomorphic encryption and secret sharing, FEDXGB ensures that the central server cannot get a correct decryption result before operating aggregation, and meanwhile, is robust against user dropout.
- **Practicality for Applications.** We evaluate the effectiveness and efficiency of FEDXGB using two standard datasets. The results indicate that FEDXGB maintains the high performance of XGBoost with less than 1% of accuracy loss and attains about 23.9% runtime and 33.3% communication reduction for gradient aggregation.

Outline. The rest of this paper is organized as follows. In Section II, some background knowledge are briefly introduced. Section III gives an overview of FEDXGB. Section IV presents the technical intuition of our secure aggregation scheme. Section V lists the implementation details of FEDXGB. The security and performance of FEDXGB are discussed and evaluated in Section VI and Section VII. Section VIII discusses the related work. The last section concludes the paper.

II. PRELIMINARY

In this section, we briefly introduce the background knowledge about XGBoost and the cryptographic functions used in FEDXGB. Table I summarizes the frequently-used notations.

TABLE I
NOTATION TABLE

Notation	Description
w_i	the first-order derivative of $l(\cdot)$ for the i_{th} instance.
h_i	the second-order derivative of $l(\cdot)$ for the i_{th} instance.
$\zeta_{u,v}$	the secret share distributed to the user u by the user v.
$\mathbb{F}$	a finite field $\mathbb{F}$, e.g. $\mathbb{F}_p = \mathbb{Z}*_p$ for some large prime p.
$\mathbb{G}$	a cyclic group with a generator g.
$\langle\cdot\rangle_u$	key used for secure aggregation.
$[\![x]\!]$	an encrypted secret x.

A. Extreme Gradient Boosting

One of the goals of FEDXGB is to extend federated learning to the ensemble learning model XGBoost. An XGBoost model is composed of multiple classification and regression trees (CARTs) that are built based on the boosting method [1]. For the k-th iteration, the objective of XGBoost is to generate a CART to minimize the following objective function $\mathcal{L}_k$.

$$\mathcal{L}_k = \sum_{i=1}^{n} l(y_i, \hat{y}_{k-1,i} + f_k(x_i)) + \Omega(f_k), \tag{1}$$

where n is the total number of training samples, i is the index of each sample, and y_i is the label of the i-th sample, $\hat{y}_{k-1,i}$ represents the predicted label of the i-th sample at the $(k-1)$-th iteration, Ω is a regularization item. To grow a CART, XGBoost iteratively adds branches (i.e., splitting the leaf node) to the current tree. Assume I_L and I_R are the instance sets of left and right nodes after a split, and $I = I_L \cup I_R$. The score to evaluate a split is as follows.

$$score = \frac{1}{2} \cdot \left(\frac{(\sum_{i\in I_L} w_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\sum_{i\in I_R} w_i^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} w_i)^2}{\sum_{i\in I} h_i + \lambda}\right), \tag{2}$$

where λ is a constant value, w_i and h_i are the first-order and second-order derivatives of $l(\cdot)$. Each time a branch is added, XGBoost chooses the split with the maximum score from all candidate splits. When a CART structure is fixed, the weight ω_j of a leaf node j is calculated by Eq. 3.

$$\omega_j = -\frac{(\sum_{i\in I} w_i)^2}{\sum_{i\in I} h_i + \lambda}. \tag{3}$$

B. Homomorphic Encryption

Bresson's cryptosystem [9] is a kind of partially homomorphic cryptosystem derived from Paillier's cryptosystem. FEDXGB adopts it to protect the gradient of users. The followings are some definitions of the cryptosystem.

Key Generation. Three inputs are required for the key generation function `Key.Gen`, namely a security parameter ℓ, a big integer $N = q_1 q_2$ and a generator g. q_1 and q_2 are two primes that satisfy $q_1 = 2q_1' + 1$ and $q_2 = 2q_2' + 1$, where q_1' and q_2' are primes and different from q_1 and q_2. $g \in \mathbb{Z}^*_{N_2}$ is a generator of the group $(\mathbb{G}, q_1, q_2, N = q_1 q_2, g)$ with order $ord(\mathbb{G}) = (p-1)(q-1)/2$. With the inputs, `Key.Gen` outputs a private-public key pair $(\langle k_{pri}\rangle, \langle k_{pub}\rangle)$, where $\langle k_{pri}\rangle \in [1, ord(\mathbb{G}))$ and $\langle k_{pub}\rangle = g^{\langle k_{pri}\rangle} \mod N^2$.

Encryption & Decryption. To encrypt a message $m \in \mathbb{Z}^*_N$, a random value $r \in \mathbb{Z}^*_N$ is first chosen. Then, using the public key $\langle k_{pub}\rangle$, we can compute the ciphertext (c_1, c_2) according to Eq. 4

$$c_1 = g^r \mod N^2, c_2 = (1 + mN)\langle k_{pub}\rangle^r \mod N^2. \tag{4}$$

Knowing the private key, we can decrypt the ciphertext as follows.

$$m = \frac{1}{N}(c_2 / c_1^{\langle k_{pri}\rangle} - 1 \mod N^2). \tag{5}$$

The above encryption is semantically secure under the decisional Diffie-Hellman assumption in $\mathbb{Z}^*_{N^2}$ [9].

Key Agreement. FEDXGB invokes the key agreement function `Key.Agr` to generate the shared key used for the shared key encryption and secret masking in the secure aggregation process. Given a user's private key $\langle k_{pri,u} \rangle$ and a public key of another user $\langle k_{pub,v} \rangle$, `Key.Agr` outputs $\langle k_{u,v} \rangle = \langle k_{pub,v} \rangle^{\langle k_{pri,u} \rangle} \mod N^2$. It can be proved that `Key.Agr` is also secure under the decisional Diffie-Hellman assumption in $\mathbb{Z}^*_{N^2}$ [9].

C. Secret Sharing

FEDXGB utilizes Shamir's secret sharing scheme [10] to deal with the user dropout problem. Two functions are involved in the Shamir's secret sharing scheme, share generation `SS.Share` and secret reconstruction `SS.Recon`. Given the secret s, the threshold t and a set of users $\mathcal{U}$, `SS.Share`(s, t, n) outputs a set of shares for each user $\{(u, \zeta_u) | u \in \mathcal{U}\}$. Inversely, given at least t shares, `SS.Recon` recovers the secret s by the Lagrange polynomials. For security, `SS.Share` and `SS.Recon` work on a finite field $\mathbb{F} = \mathbb{Z}^*_p$, where p is a big prime. Note that since FEDXGB uses the above functions to secretly share the private key, p has to be bigger than $ord(\mathbb{G})$, which is the order of the group used in `Key.Gen`.

D. Share-Key Encryption

FEDXGB uses the shared-key encryption to avoid the adversary's eavesdropping while transmitting the secret shares. For security, the encryption function `Enc` is required to be indistinguishable under a chosen plaintext attack (IND-CPA). Given a share key $\langle k_{u,v} \rangle$ and message m, `Enc` outputs a ciphertext $c_{u,v}$ =`Enc`$(\langle k_{u,v} \rangle, m)$, and `Dec`$(\langle k_{u,v} \rangle, c_{u,v})$ outputs the plaintext m.

III. OVERVIEW OF FEDXGB

In this section, we briefly overview the system design of FEDXGB for mobile crowdsensing. To provide a better understanding of FEDXGB, we first list the entities of FEDXGB, illustrated in Fig. 1. Then, we define the security model of FEDXGB. Finally, the workflow of FEDXGB is presented, shown in Fig. 2.

A. Entities of FEDXGB

FEDXGB consists of three types of entities: a set of users $\mathcal{U}$ and a central server $\mathcal{S}$.

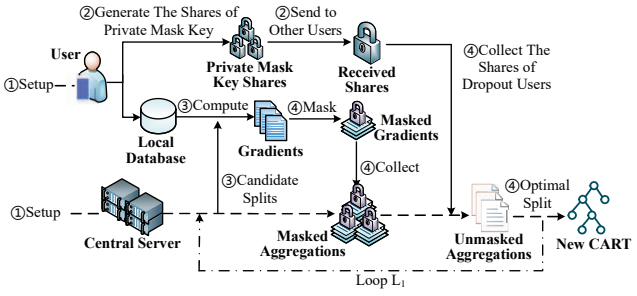


Fig. 1. Entities of FEDXGB

Users. $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. Each $u \in \mathcal{U}$ is a mobile user that volunteers to participate in federated learning and is connected with other users and the central server.

Central Server. $\mathcal{S}$ is owned by a mobile crowdsensing service provider. The aggregation of the model update for FEDXGB proceeds in $\mathcal{S}$, but $\mathcal{S}$ is not trusted by the users.

B. Security Model

In FEDXGB, our security model is based on the *curious-but-honest* model, a standard security model in federated learning [4], [8], [11]. In the model, each entity of the protocol is *curious-but-honest*, defined in *Definition 1*.

Definition 1 [12]. *In a communication protocol, a curious-but-honest entity does not deviate from the defined protocol, but attempts to learn all possible information from the legitimately received messages.*

In addition, we introduce an active adversary into our security model who has the following abilities. $\mathcal{A}$ has the following abilities: 1) simultaneously corrupt less than t legitimate users and the central server; 2) eavesdrop the communication channels; 3) for the corrupted entities, $\mathcal{A}$ can access to all their data in plaintext, e.g., private keys and random seeds; 4) is limited to have polynomial-time computation power. FEDXGB needs to achieve the following security goals.

- **Goal 1: Data Privacy.** $\mathcal{S}$ cannot learn the private data of $u \in \mathcal{U}$, no matter u is active or loses connection in the training process.
- **Goal 2: Forced Aggregation.** Considering the secure gradient aggregation of federated learning, we limit that $\mathcal{S}$ cannot ignore a specific user's uploaded model update without prior notice.

C. Workflow of FEDXGB

Three protocols are involved in FEDXGB, namely secure CART building (SecBoost), secure split finding (SecFind) and secure aggregation (SecAgg). SecFind and SecAgg are two sub-protocols of SecBoost. Here, we briefly overview how the three protocols work in FEDXGB.

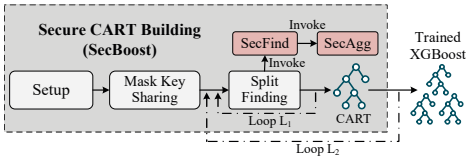


Fig. 2. Workflow of FEDXGB

As shown in Fig 2, FEDXGB trains an XGBoost model by iteratively invokes SecBoost. SecBoost takes three steps, which are setup, mask key sharing and split finding. SecFind is used to complete the split finding step of SecBoost, and SecAgg is invoked by SecFind to securely aggregate gradients.

1) **Setup.** A trusted key generate center setups the cryptographic parameters. $\mathcal{U}$ and $\mathcal{S}$ utilize the parameters to generate their cryptographic keys used in the following steps. We say that such a key generation center is

a typical role in modern networks, e.g., the digital certificate management center.

2) **Mask Key Sharing.** Before uploading the gradients for CART building, each user secretly shares its private mask key. Thus, even the user accidentally drops out, $\mathcal{S}$ can still recover its mask key and continue to get a correct gradient aggregation result.
3) **Split Finding.** As stated in Section II-A, an XGBoost model is composed of multiple CARTs. To build a CART, the key operation is to find the optimal split for the leaf node. SecBoost achieves split finding by invoking SecFind. In SecFind, the user first locally calculates the gradients based on the candidate splits published by $\mathcal{S}$. The gradients include both w_i and h_i, defined in Eq. 2. Then, $\mathcal{S}$ aggregates each user's gradients by SecAgg. Having the aggregation result, $\mathcal{S}$ can derive the scores of all candidate splits according to Eq. 2. Finally, $\mathcal{S}$ chooses the one with the maximum score and use it to add a new branch in the current CART.

By repeating the above steps (Loop L_1 in Fig. 2), $\mathcal{S}$ can obtain a well trained CART f_k. Further, $\mathcal{S}$ publishes the newly trained CART and continues to build the next CART (Loop L_2 in Fig. 2). In the end, $\mathcal{S}$ gets a trained XGBoost model $f(x) = \sum_{\kappa=1}^{K} f_\kappa(x)$, where K is the maximum training round.

IV. Technical Intuition

For federated learning, the most critical operation is the secure gradient aggregation. In this section, we introduce the intuition to design our secure aggregation protocol and present its implementation details.

A. IBM's Homomorphic Aggregation Scheme

To ensure an *honest-but-curious* central server to reliably aggregate data, a popular method is using the homomorphic encryption technique. Recently, IBM proposes such a homomorphic aggregation scheme for federated learning [8] (abbreviated as IBMHOM) as follows.

IBMHOM is based on the t-threshold Paillier cryptosystem [13], that is, the ciphertext must be decrypted with more than t secret keys. Assume that each user u holds a gradient x_u and one of the secret keys. Express the t-threshold encryption algorithm as `ThEnc`. To aggregate all users' gradients, u samples a random value $r_u \in \mathbb{Z}_N^*$ and computes:

$$[\![x_u]\!] = \texttt{ThEnc}(x_u + \epsilon_u, r_u, \langle tk_{pub,\mathcal{S}} \rangle), \tag{6}$$

where $\langle sk_{pub,\mathcal{S}} \rangle$ is the public key and ϵ is the Gaussian noise to implement differential privacy against the inference attack [14]. Then, $[\![x_u]\!]$ is uploaded. Using the homomorphism of the Paillier cryptosystem, the central server $\mathcal{S}$ can get $[\![\sum_{u\in\mathcal{U}} x_u]\!] = \prod_{u\in\mathcal{U}} [\![x_u]\!]$. Next, $\mathcal{S}$ randomly chooses t users and orderly ask them to decrypt $[\![\sum_{u\in\mathcal{U}} x_u]\!]$. In the process, if any user drops out, $\mathcal{S}$ has to ask another user to decrypt. Finally, with the partial decryption results of all the t users, $\mathcal{S}$ can recover the plaintext aggregation result.

B. Our Hybrid Masking Scheme SecAgg

We notice that the secure aggregation scheme above has two disadvantages. First, the aggregation of $\mathcal{S}$ is unforced. If a malicious server does not aggregate a specific user's data, it can still get a correct decrypted aggregation result of the remaining data. In other words, a malicious server can directly ask the users to decrypt a specific user's data, and the users cannot be aware of this. Also, the vulnerability makes it easier to launch an update-leak attack [15]. Second, the user is responsible for sending, encrypting and decrypting the data at the same time, which is too costly for a user. To resolve the problems, we propose a new secure aggregation scheme SecAgg, shown in Protocol 1.

In SecAgg, u first computes the shared mask keys with other users by `Key.Agr` and secretly share its private mask key. Then, u samples a random value r_u and masks x_u using the masking function `SecMask`.

$$\begin{aligned}[\![x_u]\!] =& \texttt{SecMask}(x_u, r_u, \langle sk_{u,v} \rangle, \langle sk_{pub,\mathcal{S}} \rangle) \\ =& [1 + (x_u + \Upsilon_u)N] \cdot \langle sk_{pub,\mathcal{S}} \rangle^{r_u \varphi(g^{r_u})} \mod N^2.\end{aligned} \tag{7}$$

where $\Upsilon_u = \sum_{u<v} \varphi(\langle sk_{u,v} \rangle) - \sum_{u>v} \varphi(\langle sk_{u,v} \rangle)$, $\langle sk_{pub,\mathcal{S}} \rangle$ is the public mask key of the central server $\mathcal{S}$, $\varphi(\cdot)$ is a pseudo-random function with a fixed-length output. $[\![x_u]\!]$ is sent to $\mathcal{S}$. In the process, $\mathcal{S}$ records the received mask values and the senders and then publishes a list of the senders $\mathcal{U}'$. The active users in $\mathcal{U}'$ return g^{r_u} and the shares of the dropout users' private mask keys. Using the shares, $\mathcal{S}$ recovers the private mask key of the dropout users and computes the shared mask keys between the dropout users and the other users. Finally, the aggregation result can be obtained according to line 8, Protocol 1. Notably, it is observed that SecAgg can be simply extended with differential privacy in a similar way of IBMHOM. However, since the inference attack is not the fundamental problem of this paper, we omit the extended implementation.

Correctness. Express the dropout users as $u_0 \in \mathcal{U}/\mathcal{U}'$. Based on our cryptographic definitions, the unmasking process (line 8, Protocol 1) can be expressed as Eq. 8 and Eq. 9.

$$\begin{aligned}\sum_{u\in\mathcal{U}'} x_u =& [1 + N \sum_{u\in\mathcal{U}'} (x_u + \Upsilon_u)] \cdot \langle sk_{pub,\mathcal{S}} \rangle^{\sum_{u\in\mathcal{U}} r_u \varphi(g^{r_u})} \\ & \cdot g^{-\langle sk_{pri,\mathcal{S}} \rangle \sum_{u\in\mathcal{U}} r_u \varphi(g^{r_u})} + \sum_{u\in\mathcal{U}/\mathcal{U}'} \Upsilon_u \mod N^2,\end{aligned} \tag{8}$$

$$\begin{cases} \Upsilon_u = \sum_{u<v} \varphi(\langle sk_{u,v} \rangle) - \sum_{u>v} \varphi(\langle sk_{u,v} \rangle) \mod p \\ \sum_{u\in\mathcal{U}} \Upsilon_u = \sum_{u\in\mathcal{U}'} \Upsilon_u + \sum_{u\in\mathcal{U}/\mathcal{U}'} \Upsilon_u = 0 \end{cases}. \tag{9}$$

Since $\langle sk_{pub,\mathcal{S}} \rangle = g^{\langle sk_{pri,\mathcal{S}} \rangle}$, we can prove that the unmasking result is the correct aggregation result by combining the above two equations.

V. Secure CART building of FedXGB

In this section, we present the details of FEDXGB for federated XGBoost training. In the protocols, the following notions are specified. All users are orderly labeled by a

Protocol 1 Secure Aggregation (SecAgg)

Input: A server $\mathcal{S}$; a user set $\mathcal{U}$; $u \in \mathcal{U}$ holds a secret x_u and a set of other user's private mask shares $\{\zeta_{v,u} | v \in \mathcal{U}/u\}$.
Output: $\mathcal{S}$ obtains the aggregated users' secrets Λ.

1: **for** $u \in \mathcal{U}$ **do**
2: Generate $\langle sk_{u,v} \rangle \leftarrow \texttt{KEY.Agr}(\langle sk_{pri,u} \rangle, \langle sk_{pub,v} \rangle)$ for $v \in \mathcal{U}/u$ and selects a random value $r_u \in \mathbb{Z}_N$.
3: Compute $[\![x_u]\!] \leftarrow \texttt{SecMask}(x_u, r_u, \langle sk_{u,v} \rangle, \langle sk_{pub,\mathcal{S}} \rangle)$ and send $[\![x_u]\!]$ to $\mathcal{S}$.
4: **end for**
5: $\mathcal{S}$ checks the dropout users in the above iteration and publishes the active user list $\mathcal{U}'$.
6: Each user checks whether it is in $\mathcal{U}'$. If yes, send g^{r_u} and $\{\zeta_{v,u} | v \in \mathcal{U}/\mathcal{U}'\}$ to $\mathcal{S}$, otherwise, wait for the next invocation.
7: $\mathcal{S}$ computes $\mathcal{R} \leftarrow \prod_{u \in \mathcal{U}'} g^{r_u \varphi(g^{r_u})}$, and for $u_0 \in \mathcal{U}/\mathcal{U}'$, recovers $\langle sk_{u_0}^{pri} \rangle \leftarrow \texttt{SS.Recon}(\{\zeta_{u_0,v} | v \in \mathcal{U}'\}, t)$ to compute $\{\langle sk_{u_0,v} \rangle | v \in \mathcal{U}'\}$ and $\Upsilon \leftarrow \sum_{u_0 \in \mathcal{U}/\mathcal{U}'} (\sum_{u_0 < v, v \in \mathcal{U}} \varphi(\langle sk_{u_0,v} \rangle) - \sum_{u_0 < v, v \in \mathcal{U}} \varphi(\langle sk_{u_0,v} \rangle))$.
8: $\mathcal{S}$ obtains $\sum_{u \in \mathcal{U}'} x_u \leftarrow \frac{1}{N} [(\prod_{u \in \mathcal{U}'} [\![x_u]\!]) \cdot \mathcal{R}^{-\langle sk_{pri,\mathcal{S}} \rangle} - 1 + \Upsilon] \mod N^2$.

sequence of indexes $(1, 2, ..., n)$ to represent their identities. Each user is deployed with a small local dataset D_u.

A. Secure CART Building SecBoost

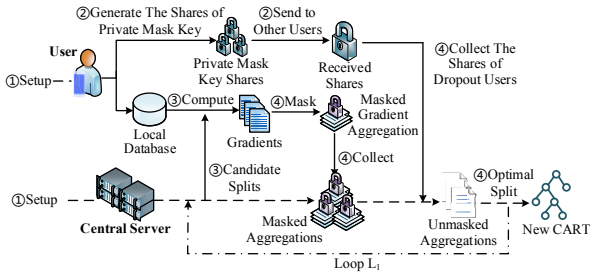


Fig. 3. Detailed Overview of SecBoost

As mentioned before, an XGBoost model is composed of multiple CARTs. FEDXGB implements secure CART building by invoking SecBoost, shown in Protocol 2. Referring to the overview illustrated in Fig. 3, we introduce the detailed steps of SecBoost.

Step 1 - Setup: In the step, $\mathcal{U}$ and $\mathcal{S}$ setup the cryptographic keys. To achieve this, a trusted key generation center samples the parameters for key generation and secret sharing, including a cyclic group $(\mathbb{G}, q_1, q_2, N = q_1 q_2, g)$ and a finite field $\mathbb{Z}_p^*$. Then, the public cryptographic parameters $(\mathbb{G}, g, N)$ and $\mathbb{Z}_p^*$ are published to both $\mathcal{U}$ and $\mathcal{S}$. Using the public parameters, each $u \in \mathcal{U}$ invokes $\texttt{Key.Gen}$ to generate two pairs of keys, $(\langle ek_{pri,u} \rangle, \langle ek_{pub,u} \rangle)$ and $(\langle sk_{pri,u} \rangle, \langle sk_{pub,u} \rangle)$, which are used for shared key encryption and secret masking in secure aggregation, respectively. Meanwhile, $\mathcal{S}$ generates a pair of masking keys, $(\langle sk_{pri,\mathcal{S}} \rangle, \langle sk_{pub,\mathcal{S}} \rangle)$ and determines the secret sharing threshold t. Finally, $\mathcal{U}$ and $\mathcal{S}$ exchange their public keys.

Step 2 - Mask Key Sharing: To deal with the user's accident dropout, each user previously generates random shares of its private mask keys. For a specific user u, it computes $\{(u, \zeta_{u,v}) | v \in \mathcal{U}\} \leftarrow \texttt{SS.Share}(\langle sk_{pri,u} \rangle, t, n)$. For each selected user $v \in \mathcal{U}$, u encrypts one of the shares $c_{u,v} \leftarrow \texttt{Enc}(\langle ek_{u,v} \rangle, u||v||\zeta_{u,v})$ and sends the encryption result to it, where $\langle ek_{u,v} \rangle \leftarrow \texttt{Key.Agr}(\langle ek_{pri,u} \rangle, \langle ek_{pub,v} \rangle)$. The user v decrypts $c_{u,v}$ and extracts $\zeta_{u,v}$. $\zeta_{u,v}$ is stored and used to recover the private mask key if u drops out.

Step 3 - Split Finding: Assume that the feature set of the user data is $\mathcal{Q} = \{\alpha_1, \alpha_2, ..., \alpha_q\}$. According to the boosting method defined in XGBoost [1], $\mathcal{S}$ randomly selects a subsample of all features $\mathcal{Q}' \subset \mathcal{Q}$ and inputs it to SecFind to find the optimal split. The detailed split finding method and optimization criteria are stated in the next section. To build a new CART with an optimal structure, $\mathcal{S}$ successively operates the above steps until the current tree depth reaches the maximum depth or other termination conditions are met [1]. Finally, SecBoost outputs a well trained CART f_κ.

B. Secure Split Finding SecFind

The most important operation of the CART building in XGBoost is to find the optimal split from all candidate splits to branch the leaf node. The candidate splits are evaluated with the split scores computed by Eq. 2. The optimal split is the split with the maximum score. In FEDXGB, split finding is implemented by SecFind, presented in Protocol 3. Details of SecFind are as follows.

First, $u \in \mathcal{U}$ computes the gradients of the local training samples (h_i and w_i, defined in Eq. 2). Then, $\mathcal{S}$ invokes SecAgg twice to get the aggregation of the two kinds of gradients H and G. Next, for each given candidate feature $\alpha \in \mathcal{Q}'$, $\mathcal{S}$ enumerates all possible candidate splits and publishes them to $\mathcal{U}$. Similar to the above aggregation process for H and G, $\mathcal{S}$ collects the left-child gradient aggregation results for each candidate split. The aggregation results are used to compute the score for each candidate split. When the iteration is terminated, SecFind returns the split with the maximum score and its corresponding feature. Intuitively, with the optimal split, $\mathcal{S}$ can add a new branch in current CART by splitting an old leaf node into two new leaf nodes. Moreover, if the termination condition is reached after the splitting, $\mathcal{S}$ extra computes the weights of the leaf nodes with the aggregated gradients by Eq. 3.

C. Robustness against User Dropout.

Two possible cases of user dropout in FEDXGB are discussed as follows.

Protocol 2 Secure Extreme Gradient Boosting Based Tree Building (SecBoost)

Input: A central server $\mathcal{S}$, a user set $\mathcal{U} = \{u_1, ..., u_n\}$ and a key generation center $\mathcal{T}$.
Output: A well-trained CART.

1: **Step 1 - Setup:**
2: Given the security parameter ℓ, $\mathcal{T}$ randomly selects three strong primes p, q_1 and q_2, and samples a cyclic group $(\mathbb{G}, q_1, q_2, N = q_1 q_2, g)$, where g is a generator with order $ord(g) = \frac{(q_1-1)(q_2-1)}{2}$ and $p > ord(g)$. $(\mathbb{G}, g, N)$ and p are published to both $u \in \mathcal{U}$ and $\mathcal{S}$.
3: Each $u \in \mathcal{U}$ compute $(\langle ek_{pri,u}\rangle, \langle ek_{pub,u}\rangle) \leftarrow$`Key.Gen`$(g, N, \ell)$ and $(\langle sk_{pri,u}\rangle, \langle sk_{pub,u}\rangle) \leftarrow$`Key.Gen`$(g, N, \ell)$.
4: $\mathcal{S}$ generates $(\langle sk_{pri,\mathcal{S}}\rangle, \langle sk_{pub,\mathcal{S}}\rangle) \leftarrow$`Key.Gen`$(g, N, \ell)$ and determines the secret sharing threshold t.
5: $\mathcal{S}$ and $\mathcal{U}$ publish their public keys.
6: **Step 2 - Mask Key Sharing:**
7: $u \in \mathcal{U}$ computes the shares of its private mask key $\langle sk_{pri,u}\rangle$ by $\{(u, \zeta_{u,v})|v \in \mathcal{U}\} \leftarrow$ `SS.Share`$(\langle sk_{pri,u}\rangle, t, n)$.
8: For $v \in \mathcal{U}$, u sends $c_{u,v} \leftarrow$`Enc`$(\langle ek_{u,v}\rangle, u||v||\zeta_{u,v})$, where $\langle ek_{u,v}\rangle \leftarrow$`Key.Agr`$(\langle ek_{pri,u}\rangle, \langle ek_{pub,v}\rangle)$.
9: $v \in \mathcal{U}$ decrypts $\zeta_{u,v} \leftarrow$`Dec`$(\langle ek_{u,v}\rangle, c_{u,v})$, and stores $(u, \zeta_{u,v})$.
10: **Step 3 - Split Finding:**
11: $\mathcal{S}$ randomly selects a feature sub-sample $\mathcal{Q}'$ from the full feature set $\mathcal{Q}$.
12: $\mathcal{S}$ invokes SecFind($\mathcal{Q}'$,$\mathcal{U}$) to determine the current optimal split.
13: Repeat **Step 3** until reaching the termination condition.

Protocol 3 Secure Split Finding (SecFind)

Input: The sub-sampled feature set $\mathcal{Q}'$; the user set $\mathcal{U}$; $u \in \mathcal{U}$ holds a set of shares about other user's private mask keys $\{\zeta_{v,u}|v \in \mathcal{U}/u\}$

1: $u \in \mathcal{U}$ computes $h_u \leftarrow \sum_{i=1}^{|D_u|} h_i$ and $w_u \leftarrow \sum_{i=1}^{|D_u|} w_i$.
2: $\mathcal{S}$ invokes $H \leftarrow$SecAgg($\mathcal{S}$,$\mathcal{U}$, $\{h_u|u \in \mathcal{U}\}$, $\{\zeta_{v,u}|u \in \mathcal{U}, v \in \mathcal{U}/u\}$) and $W \leftarrow$SecAgg($\mathcal{S}$,$\mathcal{U}$, $\{w_u|u \in \mathcal{U}\}$, $\{\zeta_{v,u}|u \in \mathcal{U}, v \in \mathcal{U}/u\}$).
3: **for** $1 \le q \le \delta$ **do**
4: $\mathcal{S}$ enumerates every possible candidate split $A_q = \{a_1, a_2, ..., a_m\}$ for feature $\alpha_q \in \mathcal{Q}'$ and publishes them to $\mathcal{U}$. For each $a_r \in A_q$, take the following steps.
5: Based on the candidate splits, $u \in \mathcal{U}$ computes $h_{u,L} \leftarrow \sum_{i=1}^{|D_L|} h_i$ and $w_{u,L} \leftarrow \sum_{i=1}^{|D_L|} w_i$.
6: $\mathcal{S}$ invokes $H_L \leftarrow$SecAgg($\mathcal{S}$,$\mathcal{U}$, $\{h_{u,L}|u \in \mathcal{U}\}$, $\{\zeta_{v,u}|u \in \mathcal{U}, v \in \mathcal{U}/u\}$) and $W_L \leftarrow$SecAgg($\mathcal{S}$,$\mathcal{U}$, $\{w_{u,L}|u \in \mathcal{U}\}$, $\{\zeta_{v,u}|u \in \mathcal{U}, v \in \mathcal{U}/u\}$).
7: $\mathcal{S}$ computes $H_R \leftarrow H - H_L$ and $W_R \leftarrow W - W_L$.
8: $score \leftarrow max(score, \frac{W_L^2}{H_L+\lambda} + \frac{W_R^2}{H_R+\lambda} - \frac{W^2}{H+\lambda})$.
9: **end for**
10: **return** The optimal split with maximum $score$.

Case 1: A user u_0 drops out at the first or second step of SecBoost. In such a case, the user becomes illegal. $\mathcal{S}$ refuses u_0 to be involved in the current round of training and replaces the user by another active user if possible.

Case 2: A user u_0 drops out during the secure aggregation process of split finding. $\mathcal{S}$ recovers the private mask key of u_0 and removes u_0 from $\mathcal{U}$, that is, the active user list becomes $\mathcal{U}' \subseteq \mathcal{U}$ and $u_0 \in (\mathcal{U} \setminus \mathcal{U}')$. To recover the private mask key of u_0, $\mathcal{S}$ collects the shares of its private mask key from at least t users, i.e., $\{\zeta_{u_0,v}|v \in \mathcal{U}'\}$ and $|\mathcal{U}'| > t$. Then, $\mathcal{S}$ recovers the private mask key of u_0 through $\langle sk_{pri,u_0}\rangle \leftarrow$`SS.Recon`$(\{\zeta_{u_0,v}|v \in \mathcal{U}'\}, t)$. Using $\langle sk_{pri,u_0}\rangle$, $\mathcal{S}$ computes the shared mask keys that u_0 uses to mask the gradients, $\{\langle sk_{u_0,v}\rangle|v \in \mathcal{U}\}$, where $\langle sk_{u_0,v}\rangle \leftarrow$`KEY.Agr`$(\langle sk_{pri,u_0}\rangle, \langle sk_{pub,v}\rangle)$. Finally, $\mathcal{S}$ adds the shared mask keys to the aggregated result, shown in line 8, Protocol 1. In this way, $\mathcal{S}$ can still get the correct aggregation result of remaining active users' gradients, whose correctness has been discussed in Section IV.

VI. Security Analysis

In this section, we discuss the security of FEDXGB for secure aggregation and XGBoost training.

A. Security of SecAgg

For SecAgg, we first present how it achieves our security goals, and then, give a more formal security proof in the next sub-section.

SecAgg achieves forced aggregation because from the analysis of correctness, ignoring any user's data makes $\mathcal{S}$ unable to get a meaningful result. Then, consider a single user's masked value $[\![x_u]\!]$ derived by Eq. 7. For an eavesdropper or a malicious user, $[\![x_u]\!]$ is a ciphertext encrypted with the server's public key, which is semantically secure based on the security of Bresson's cryptosystem [9]. If a malicious central server is included, there are two conditions required to be discussed: a) when the user does not drop out, the adversary can decrypt $[\![x_u]\!]$ but cannot obtain x_u that is masked by Υ_u. b) when the user drops out, the adversary can access both Υ_u and $\langle sk_{pri,\mathcal{S}}\rangle$. However, since g^{r_u} is unknown, $[\![x_u]\!]$ is still undecipherable. Further, assume that the central server is a more active attacker (out of the scope of our security model). In such a case, the central server can cheat the user to upload g^{r_u} by sending a forged active user list. We can defend the attack by letting the user sign the active user list $\mathcal{U}'$ and exchanging it with other users. Thus, by checking the consistency of $\mathcal{U}'$, the user can defend the active attack.

B. Security of FEDXGB

The security of FEDXGB for XGBoost training is determined by three protocols, SecBoost, SecFind and SecAgg. To prove the protocols' security, we adopt the standard formal definition of security *Definition 2* [16].

Definition 2. *We say that a protocol π is secure if there exists a probabilistic polynomial-time (PPT) simulator ξ that can generate a view for the adversary $\mathcal{A}$ in the real world and the view is computationally indistinguishable from its real view.*

Moreover, our security still needs the following lemma.

Lemma 1 [17]. A protocol is perfectly simulatable if all its sub-protocols are perfectly simulatable.

Interested readers can refer to [17] for the detailed proof of *Lemma 1*. According to *Lemma 1* and *Definition 2*, to prove the security of FEDXGB, we just have to prove that all of its protocols are simulatable for a PPT simulator. Since SecAgg is a sub-protocol that is frequently invoked by SecFind, we merge the proof of SecAgg into SecFind. The security proofs of SecFind and SecBoost are given below.

Theorem 1. *For SecFind, there exists a PPT simulator ξ that can simulate an ideal view which is computationally indistinguishable from the real view of $\mathcal{A}$.*

Proof. Denote the views of the user $u \in \mathcal{U}$ as $\mathcal{V}_u = \{view_{u_1}, ..., view_{u_n}\}$. From SecFind, we can derive that $view_{u_i} = \{h_{u_i}, w_{u_i}, r_{u_i}, g^{r_{u_i}}, [\![h_{u_i}]\!], [\![w_{u_i}]\!], \langle sk_{u,v}\rangle, \langle sk_{pri,u_i}\rangle, \langle sk_{pub,v}\rangle, \langle sk_{pub,\mathcal{S}}\rangle\}$ and $view_{\mathcal{S}} = \{H, W, H_j, W_j, A, H_L, H_R, W_L, W_R, score, \mathcal{Y}, \mathcal{R}, \mathcal{K}, \langle sk_{pri,\mathcal{S}}\rangle\}$, where $u \in \mathcal{U}$, $v \in \mathcal{U}$, $\mathcal{R} = \{[\![h_{u_i}]\!], [\![w_{u_i}]\!] | u \in \mathcal{U}'\}$, $\mathcal{R} = \{g^{r_{u_i}} | u \in \mathcal{U}'\}$, $\mathcal{K} = \{\langle sk_{pri,u_i}\rangle | u \in \mathcal{U}/\mathcal{U}'\}$. $[\![h_{u_i}]\!]$, $[\![w_{u_i}]\!]$, $\mathcal{Y}$, $\mathcal{R}$ and $\mathcal{K}$ are the variables used in SecAgg.

We prove the security of SecFind according to the universal composition theorem (UC) [18]. Assume that there is an ideal functionality $\mathcal{F}$ that can be called by a simulator ξ. $\mathcal{F}$ has the ability to ideally generate uniformly random values and operate the cryptographic functions of FEDXGB. We say that with $\mathcal{F}$, there exists such a simulator that can simulate both the honest entity and the corrupted (curious-but-honest) entity in SecFind as follows. For the corrupted entity, ξ can access all its local data, including the private keys, training samples and etc., based on our security model. Therefore, ξ can simply use the corrupted data to simulate the corrupted entity. For the honest entity, the simulation is a little complicated. To simulate an honest user, ξ first asks $\mathcal{F}$ to generates random values as dummy function inputs h_{u_i}, w_{u_i}, r_{u_i} and $\langle sk_{pri,\mathcal{S}}\rangle$. Then, use the dummy inputs to derive other variables to ask $\mathcal{F}$ to complete the protocol steps. Similarly, ξ can use the same way to simulate an honest server. It is observed that the elements in $View_{u_i}$ or $View_{\mathcal{S}}$ are either the ciphertext in the Bresson's cryptosystem or random values (h_{u_i} and w_{u_i} are private, which can be seen as random values). Since the Bresson's cryptosystem is semantically secure [9], the dummy function outputs are computationally indistinguishable from the real ones. Consequently, there exists a simulator that can generate the simulated views Sim_{u_i} and $Sim_{\mathcal{S}}$ that are indistinguishable from $View_{u_i}$ and $View_{\mathcal{S}}$. Based on *Definition 2*, *Theorem 1* holds and SecFind is secure. □

Theorem 2. *For SecBoost, there exists a PPT simulator ξ that can simulate an ideal view which is computationally indistinguishable from the real view of $\mathcal{A}$.*

Proof. Denote the views of the user and the central server for SecBoost as $\mathcal{V}_u = \{view_{u_1}, ..., view_{u_n}\}$ and $\mathcal{V}_{\mathcal{S}}$. The view of the user is $view_{u_i} = \{\langle sk_{pri,u_i}\rangle, \langle sk_{pub,u_i}\rangle, \langle ek_{pri,u_i}\rangle, \langle ek_{pub,u_i}\rangle, c_{v,u}, \{\zeta_{v,u} | v \in \mathcal{U}'\}, view'_{u_i}\}$. For the central server, we have $view_{\mathcal{S}} = \{\langle sk_{pub,\mathcal{S}}\rangle, \langle sk_{pri,\mathcal{S}}\rangle view'_{\mathcal{S}}\}$, $view'_{u_i}$ and $view'_{\mathcal{S}}$ are the views generated by SecFind. Except for the encryption keys randomly selected from $\mathbb{Z}_N^*$, the remaining elements of $view_{u_i}$ and $view_{\mathcal{S}}$ are random shares or ciphertexts encrypted with the shared key encryption algorithm. According to Shamir's secret sharing theory [19], the shares can be regarded as random values uniformly selected from $\mathbb{Z}_p^*$. To simulate the ciphertexts and random shares, a simulator ξ can use the ideal functionality $\mathcal{F}$ defined in the proof of SecFind to generate random values as dumpy cryptographic function's inputs. Since the shared key encryption algorithm is assumed to be indistinguishable under a chosen plaintext attack, the corresponding dumpy outputs cannot be computationally distinguished. Moreover, $view'_{u_i}$ and $view'_{\mathcal{S}}$ have been proved to be simulatable in the proof of *Theorem 2*. Thus, there exists a simulator ξ that can simulate a corrupted entity or an honest entity of SecBoost and the simulated view is computationally indistinguishable from the real view. Based on *Definition 2*, *Theorem 2* holds and SecBoost is secure. □

According to *Lemma 1* and the above proofs, it is concluded that FEDXGB is a simulatable system. Based on the formal definition of security given in *Definition 2*, FEDXGB is secure.

VII. PERFORMANCE EVALUATION

In this section, we conduct extensive experiments to evaluate the effectiveness and efficiency of FEDXGB.

A. Experiment Configuration

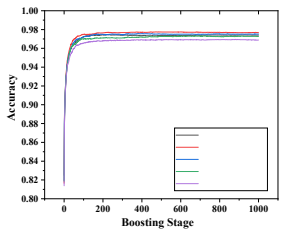
To evaluate FEDXGB, we ran single-threaded simulations on a Windows desktop with an Intel Core i7-8565U CPU @1.8Ghz and 16G RAM. The programs are implemented in Python and C++. Two standard datasets are used in the experiments, ADULT[1] and MNIST[2], both of which are commonly used to evaluate the performance of federated learning schemes [4], [11], [20]. The Bresson's cryptosystem is conducted with a key size of 512 bits.

The shared-key encryption is operated by 128-bit AES-GCM [21]. Given each dataset, the instances are averagely and randomly assigned to each user with no overlap. User dropout is assumed to occur every 10 rounds of training in our experiment. That is, 0%, 10%, 20%, 30% of users are randomly selected to be disconnected at each 10^{th} round of training.

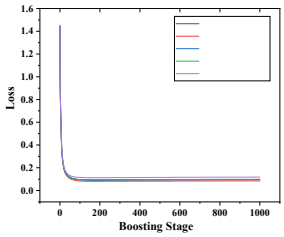
[1]ADULT: https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets
[2]MNIST: http://yann.lecun.com/exdb/mnist/

B. *Evaluation of* FEDXGB

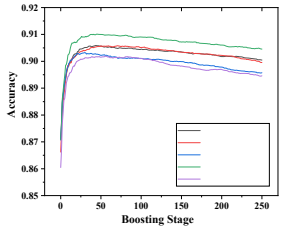
To assess the performance of FEDXGB, we first evaluate its effectiveness with ADULT and MNIST. Then, we experiment with its runtime to find an optimal split to test its efficiency.



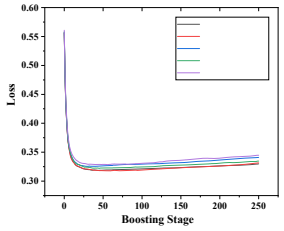
(a) Accuracy with different user dropout rates for ADULT.



(b) Loss with different user dropout rates for ADULT.



(c) Accuracy with different user dropout rates for MNIST.



(d) Loss with different user dropout rates for MNIST.

Fig. 4. Accuracy and loss for each round of training with MINST and ADULT. Different lines show different dropout rates.

The effectiveness of FEDXGB is assessed with two indicators that are commonly used to evaluate a machine learning model, namely classification accuracy and loss. Fig. 4 presents the accuracy and loss for each round of training in FEDXGB. More specific, Fig. 4(a) and Fig. 4(b) describe the accuracy and loss of ADULT, and Fig. 4(c) and Fig. 4(d) show the result of MNIST. For ADULT, the accuracy peaks after around 100 rounds. For MINST, the convergence speed is faster, peaking at around the 20^{th} rounds. Compared with the non-federated XGBoost, FEDXGB only introduces the accuracy loss with less than 1%. Consider the user dropout rate increased from 0% to 30%, FEDXGB is robust against the user changes. The performance decrease is mainly due to the loss of data caused by the user dropout. In Table II, we list the runtime and communication cost of different stages to execute SecBoost to find an optimal split in FEDXGB without user dropout. The processed data in the experiment is ADULT. The user number is set to 500. The system setup cost is ignored. The result indicates that the main overhead in FEDXGB is caused by the split finding step, because numerous secure aggregation protocols are invoked. Therefore, in the next section, we comprehensively analyze the efficiency of SecAgg.

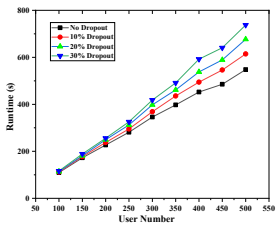
C. *Efficiency Analysis of SecAgg*

To further assess the efficiency of FEDXGB, we simulate the runtime and communication costs of SecAgg under different numbers of users, input sizes and dropout rates.

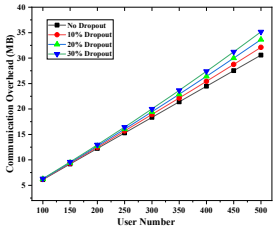
TABLE II
SECBOOST RUNTIME IN DIFFERENT STAGES WITHOUT USER DROPOUT

Stage	RunTime (s)		Communication (MB)	
	$\mathcal{U}$	$\mathcal{S}$	$\mathcal{U}$	$\mathcal{S}$
Mask Key Sharing	0.89	N.A.	0.06	N.A.
Split Finding	1.22	249.97	26.86	46.25
Total Cost	2.11	249.97	0.05	13.57

Theoretical Analysis. Suppose the transmitted data is a vector with m entries. The length of N is $\mathcal{N} = \lfloor \log_2 N \rfloor$. The user number is n. For communication, each user sends one random seed, m ciphertexts and $\mathcal{O}(n)$ shares for private mask key and dropout users, whose complexity is $\mathcal{O}(m+n)$. The server receives the user's ciphertexts, the random seeds and key shares of dropout users, whose overhead is $\mathcal{O}(nm+n^2)$. For computation, suppose the modular exponentiation costs $1.5\mathcal{N}$ multiplications [22]. Each user computes $2m$ times modular exponentiation and $\mathcal{O}(n^2)$ multiplications for sharing the private mask key, which takes $\mathcal{O}(m\mathcal{N}+n^2)$ time. The server conducts mn times modular exponentiation to decryption and $\mathcal{O}(n^2)$ multiplications for data recovering of the dropout user, which takes $\mathcal{O}(mn\mathcal{N}+n^2)$ time.

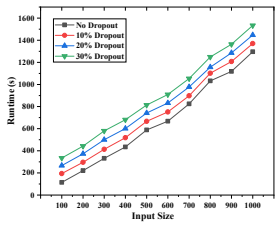


(a) Runtime as the user number increases.

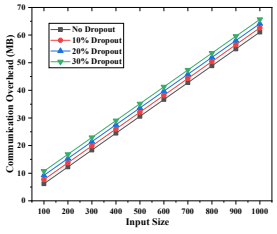


(b) Communication overhead as the user number increases

Fig. 5. Efficiency evaluation of the central server with fixed input size 500. Different lines show different dropout rates.



(a) Runtime as the input size increases.



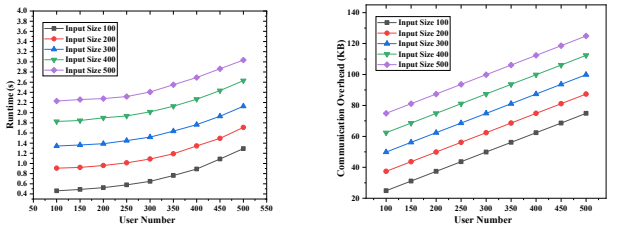
(b) Communication overhead as the input size increases.

Fig. 6. Efficiency evaluation of the central server with fixed user number 500. Different lines show different dropout rates.

Impact of Users. The runtime and communication overhead of the central server and the user with different user numbers are plotted in Fig. 5, Fig. 7, respectively. The input size is fixed to 500. The steps of setup and the shared mask keys

generation can be previously completed, therefore, they are not included in the evaluations. We omit the plot of the runtime and communication overhead of the user with different dropout rates, as the user just has to send the private mask key shares of the dropout users, which has little impact on the metrics.

As shown in Fig. 5(a) and Fig. 7(a), it is shown that the runtime for the central server and the user grows with the increasing user number. For the central server, the runtime is mostly spent on the modular exponentiation to decrypt the masked aggregation result. For the user, most computational costs are also focused on the modular exponentiation but for masking the gradient. The user dropout rate has a significant influence on the runtime of the central server because the operation for recovering the shared mask key of the dropout users involves the costly modular exponentiation. Fig. 5(b) and Fig. 7(b) illustrate the communication overhead for the user and the central server. The communication overhead for both the user and the central server also linearly increases as the user number increases. As the dropout rate grows, the communication overhead of the central server is barely influenced because only a little overhead increment is caused by collecting the private mask key shares for the dropout user.



(a) Runtime per user as the user number increases.

(b) Communication overhead per user as the user number increases.

Fig. 7. Efficiency evaluation of the user without user dropout.

Impact of Input Size. The runtime and communication overhead of the user and the central server with different input sizes are plotted in Fig. 6 and Fig. 7, respectively. In XGBoost, the input size is equal to the multiplication of the sub-sampled feature number and the enumerated candidate split number. In the experiments, the number of users is fixed to 500. When the input size increases from 100 to 1000, the runtime cost for each user increases because of the masking operation, illustrated in Fig. 7(a). The growth of the central server's runtime is mainly caused by the more masked aggregation result required to be decrypted. As a larger scale of inputs is involved, the communication overhead of the central server is expanded, shown in Fig. 6(b). For the user, Fig. 7(b) shows that the communication overhead is linearly influenced by the input size. Compared with the number of users, the input size has a less obvious influence on the runtime and communication overhead, which is consistent to our theoretical analysis result given in Section IV.

Comparison. Besides IBMHOM, we compare the efficiency of FEDXGB with the functional Paillier encryption based secure aggregation scheme (abbreviated as FPE) [23].

TABLE III
EFFICIENCY COMPARISON WITH FIXED USER NUMBER 500, INPUT SIZE 500 AND NO USER DROPOUT

		FEDXGB	IBMHOM [8]	FPE [23]
Rt.	Server	557.97	596.03	952.07
	User	3.03	4.67	2.55
Comm.	Server	30.57	67.13	30.58
	User	0.12	0.18	0.08

Rt. $\rightarrow$ Runtime (s); Comm.$\rightarrow$ Communication overhead (MB)

We implement all the three methods in our desktop, referring to the python library for Paillier's cryptosystem[3]. Moreover, for fairness, we cancel the noise addition operation of IBMHOM. Considering the secure aggregation, the cancellation does not influence the security of IBMHOM. The threshold of IBMHOM is set to $0.6 \times n$. Table III summarizes the comparison result. It is observed that compared with IBMHOM, FEDXGB outperforms it in all indicators. Note that during the experiments, we find that the server in IBMHOM requires very little time to decrypt the aggregation result when the user number is small, e.g., the user number n is less than 100. This is because, for the HE algorithm used in IBMHOM, the complexity of the exponent length involved in the decryption is $\mathcal{O}(n \log n)$, not fixed $\mathcal{O}(\log N)$ in FEDXGB. When the user scale is small, IBMHOM is faster. However, in real-world applications, such a small scale of users is almost impossible. For FPE, although its communication overhead and user runtime are less than FEDXGB, its server has to conduct double times modular exponentiation to unmask the encrypted aggregation result, which greatly increases its runtime. Therefore, when applied to federated learning, FEDXGB is still more practical than FPE.

VIII. RELATED WORK

Google's federated learning is a kind of privacy-preserving machine learning framework originally proposed for the mobile crowdsensing scenario [4]. Due to the high performance on security and efficiency, federated learning attracts a lot of attention as soon as being proposed. Up to now, most of the existing federated learning schemes are designed towards the stochastic gradient descent (SGD) based neural networks. For example, Wang *et al.* [24] provided an edge computing based federated learning scheme for the convolutional neural network (CNN) in the Internet of Things (IoT) environment. Mcmahan *et al.* [20] applied federated learning to the long-short term memory network (LSTM) based language model and gain better performance than the traditional centralized machine learning method. Smith *et al.* [25] proposed a general federated learning framework for the neural network to simultaneously process multi-tasks, which solves the stragglers and fault tolerance problems in the real-world network and significantly improved the efficiency of the original federated learning

[3]https://github.com/data61/python-paillier

framework. Nonetheless, there is none of the existing work that gives a systematical federated learning scheme for XGBoost, a special tree structure machine learning model.

To avoid the adversary to analyze the hidden information about private user data from the uploaded gradient values [14], almost all of the current federated learning schemes introduce the secure aggregation mechanism. The existing secure aggregation schemes for federated learning mainly depend on three types of cryptographic tools. The first and most popular tool is differential privacy (DP). Mcmahan *et al.* [20] is one of the outstanding works using DP to protect the gradient's security. Nonetheless, the introduction of noises for DP is pointed to be able to lead non-erasable accuracy loss to the trained model [26]. The second tool is secret sharing (SS), especially the Shamir's secret sharing scheme. In [11], Bonawitz proposed a novel SS based secure aggregation scheme against user dropout. However, since having to operate data reconstruction for all users, no matter the user is dropout or not, the communication cost of [11] explodes with the increasing of the user number. The last tool is homomorphic encryption (HE). For HE, the most commonly used HE method is the Pailliar cryptosystem [27] and its variants [9], [28]. Although many HE schemes are proposed [6], [8], [29], none of them solve the forced aggregation problem while preserving the robustness against user dropout.

IX. Conclusion

In this paper, we proposed a privacy-preserving federated extreme gradient boosting scheme (FEDXGB) for mobile crowdsensing. In FEDXGB, a new hybrid secure aggregation scheme is first presented by combining homomorphic encryption and secret sharing, which can force the central server to conduct the aggregation operation, and is robust against user dropout. Then, using the newly designed secure aggregation scheme, we designed a suite of secure protocols to implement the classification and regression tree building of XGBoost. Comprehensive experiments were conducted to evaluate the effectiveness and efficiency of FEDXGB. Experiment results showed that FEDXGB made it possible to train an XGBoost with negligible performance loss, and attained computation and communication cost reduction for secure aggregation.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 61872283, 61702105, U1764263, U1804263), the China 111 Project (No. B16037).

References

[1] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.

[2] P. Rustgi and C. Fung, "Droidnet-an android permission control recommendation system based on crowdsourcing," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 737–738.

[3] D. Garcia, Y. M. Kassa, A. Cuevas, M. Cebrian, E. Moro, I. Rahwan, and R. Cuevas, "Analyzing gender inequality through large-scale facebook advertising data," *Proceedings of the National Academy of Sciences*, vol. 115, no. 27, pp. 6958–6963, 2018.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[5] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[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 preprint arXiv:1912.00513*, 2019.

[7] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, "Secureboost: A lossless federated learning framework," *arXiv preprint arXiv:1901.08755*, 2019.

[8] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, and R. Zhang, "A hybrid approach to privacy-preserving federated learning," *arXiv preprint arXiv:1812.03224*, 2018.

[9] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2003, pp. 37–54.

[10] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[12] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," *Tech. Rep.*, 2014.

[13] I. Damgard and M. Jurik, "A generalisation, a simplification, and some applications of paillier's probabilistic public-key system, presented at the 4th international workshop on practice and theory in public key cryptosystems, cheju island," *Korea*, 2001.

[14] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019 Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.

[15] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-leak: Data set inference and reconstruction attacks in online learning," *arXiv preprint arXiv:1904.01067*, 2019.

[16] Z. Ma, Y. Liu, X. Liu, J. Ma, and K. Ren, "Lightweight privacy-preserving ensemble classification for face recognition," *IEEE Internet of Things Journal*, 2019.

[17] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[18] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 19–38.

[19] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981.

[20] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[21] M. Bellare and B. Tackmann, "The multi-user security of authenticated encryption: Aes-gcm in tls 1.3," in *Annual International Cryptology Conference*. Springer, 2016, pp. 247–276.

[22] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[23] M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu, "Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings," in *Annual International Cryptology Conference*. Springer, 2018, pp. 597–627.

[24] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[25] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[26] C.-L. Chen, R. Pal, and L. Golubchik, "Oblivious mechanisms in differential privacy: experiments, conjectures, and open questions," in

2016 IEEE Security and Privacy Workshops (SPW). IEEE, 2016, pp. 41–48.

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[28] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, 2001, pp. 119–136.

[29] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 13–23.