mathematics

MDPI

Article

Scalable Multi-Party Privacy-Preserving Gradient Tree Boosting over Vertically Partitioned Dataset with Outsourced Computations

Kennedy Edemacu [1] and Jong Wook Kim [2,*]

[1] Department of Computer Science and Electrical Engineering, Muni University, Arua P.O. Box 725, Uganda; k.edemacu@muni.ac.ug
[2] Department of Computer Science, Sangmyung University, Seoul 03016, Korea
* Correspondence: jkim@smu.ac.kr

Abstract: Due to privacy concerns, multi-party gradient tree boosting algorithms have become widely popular amongst machine learning researchers and practitioners. However, limited existing works have focused on vertically partitioned datasets, and the few existing works are either not scalable or tend to leak information. Thus, in this work, we propose SSXGB, which is a scalable and acceptably secure multi-party gradient tree boosting framework for vertically partitioned datasets with partially outsourced computations. Specifically, we employ an additive homomorphic encryption (HE) scheme for security. We design two sub-protocols based on the HE scheme to perform non-linear operations associated with gradient tree boosting algorithms. Next, we propose secure training and prediction algorithms under the SSXGB framework. Then, we provide theoretical security and communication analysis for the proposed framework. Finally, we evaluate the performance of the framework with experiments using two real-world datasets.

Keywords: gradient tree boosting; multi-party machine learning; privacy-preservation; homomorphic encryption; vertically partitioned dataset

MSC: 68T07

Citation: Edemacu, K.; Kim, J.W. Scalable Multi-Party Privacy-Preserving Gradient Tree Boosting over Vertically Partitioned Dataset with Outsourced Computations. *Mathematics* **2022**, *10*, 2185. https://doi.org/10.3390/math10132185

Academic Editors: Zibin Zheng, Ruoxi Jia, Dan Li, Yuxun Zhou and Liang Xu

Received: 1 June 2022
Accepted: 20 June 2022
Published: 23 June 2022

1. Introduction

The privacy-preserving multi-party machine learning paradigm has shown promising potential in encouraging collaboration between organizations while preserving the privacy of their data [1]. The basic idea of the privacy-preserving multi-party machine learning is that each collaborating party holds a private dataset and trains a local model using the dataset. The local models from the participating parties are then aggregated to create a single and more powerful model. Hence, different organizations can jointly train a machine learning model without sharing their private datasets. Usually, privacy-preserving mechanisms such as multi-party computation, homomorphic encryption, and differential privacy are employed to improve the security of privacy-preserving multi-party machine learning frameworks.

Although the privacy-preserving multi-party machine learning has attracted a lot of attention recently, the majority of the existing works focus on linear regression [2,3], logistic regression [4,5] and neural networks [6,7] over vertically and horizontally partitioned datasets (For horizontally partitioned datasets, participants hold subsets of the samples with the same features. For example, two regional banks may have different user groups from their regions, with the intersection between the user groups being small (different samples). However, the two banks have similar businesses and hence, similar features. While for vertically partitioned datasets, participants hold the same samples with different features. For example, two companies, a bank and an e-commerce company located in the same city. The two companies are likely to have the same users, but since their businesses are different, they will have different features [8]).

Similar to the above methods, gradient tree boosting [9], which is one of the most popular machine learning methods, has also received considerable attention due to its effectiveness in a wide range of application areas such as fraud detection [10], feature selection [11] and product recommendation [12]. Efforts to address the privacy concerns for gradient tree boosting in multi-party settings are presented in [13–18]. The datasets in [13–15] are horizontally partitioned while the datasets are vertically partitioned in [16–18].

In this work, we focus on the latter dataset partitioning. The current privacy preservation efforts proposed for the multi-party gradient tree boosting method with vertically partitioned datasets have a number of limitations. In [18], the proposed scheme is not scalable; it is limited to only two collaborating parties. In [16,17], intermediate information is revealed during the model training. Thus, designing a scalable and yet secure gradient tree boosting scheme has remained open for investigation, and hence, we intend to answer the question of how to construct a scalable (in terms of collaborating parties) and secure XGBoost [19] over vertically partitioned datasets in this work.

Apart from the high memory usage challenge, the secure XGBoost model training requires complicated computation primitives such as *division* and *argmax* [18]. To address these challenges and build a scalable but secure XGBoost over vertically partitioned datasets, we propose the SSXGB framework that securely outsources and performs the complicated computations in encrypted form. The key idea is to allow the participants to jointly train a model by sharing their encrypted information with a server that, in turn, collaborates with a second server to securely perform further computations to complete the generation of the model. Specifically, we present an additive homomorphic encryption (HE) scheme that provides the addition (`Add`) and subtraction (`Sub`) primitives. We also present sub-protocols designed to provide additional primitives such as the multiplication (`Mult`) and comparisons. Next, we propose new sub-protocols based on the HE scheme for the division (`Div`) and `argmax` primitives. We employ the secure computation primitives to build the scalable and secure XGBoost model. Then, we present a secure prediction algorithm for predictions based on the trained model. We present the analysis of our framework and its implementation using real-world datasets. A summary of our contributions is presented as follows:

- We propose sub-protocols based on an additive HE scheme used to perform primitive secure operations during a machine learning task. The sub-protocols are collaboratively executed by two non-colluding servers.
- We design a novel scalable and privacy-preserving multi-party XGBoost training algorithm and a corresponding prediction algorithm. The algorithms are constructed under the semi-honest security assumption and there is no limit on the number of participants involved.
- We conduct experiments using real-world datasets to demonstrate the effectiveness and efficiency of our proposed framework.

The rest of the paper is organized as follows. In Section 2, we present the related works. Section 3 contains the preliminary concepts. In Section 4, we present our proposed HE sub-protocols for non-linear operations. We present the overview of the proposed SSXGB framework in Section 5. Sections 6 and 7 present the secure training and prediction algorithms of the SSXGB framework. In Section 8, we present theoretical security and communication analysis. Performance evaluation is presented in Section 9, and Section 10 concludes the paper.

2. Related Work

Recently, efforts devoted to multi-party machine learning research have shown a huge potential in addressing the training data scarcity problem while preserving data privacy [1,5,20,21]. However, the majority of the works focus on linear machine learning models. Little effort has been invested in researching multi-party gradient tree boosting models. Currently, multi-party gradient tree boosting frameworks can be categorized as

horizontal, *vertical* and *generic* frameworks, depending on how the datasets are partitioned amongst the collaborating participants.

2.1. Horizontal Multi-Party Gradient Tree Boosting Frameworks

In horizontal multi-party gradient tree boosting frameworks, sets of samples are shared amongst the collaborating participants. Several works have adopted this approach. In [14], Ong et al. designed a multi-party gradient tree boosting framework in which the participants exchange adaptive histogram representations of their data during model learning. Liu et al. combined secret sharing with homomorphic encryption to prevent participants from dropping out and securely aggregate their gradients during XGBoost training [15]. Reference [22] employed an oblivious algorithm to prevent privacy violations at hardware enclaves during learning of a multi-party gradient tree boosting model. In [23], Yang et al. designed a multi-party tree boosting framework with anomaly detection from extremely unbalanced datasets. Reference [24] designed secure training and prediction frameworks for multi-party gradient tree boosting. A secret sharing scheme is employed for the secure training, while a key agreement scheme and an identity-based encryption and signature scheme are employed for the secure prediction framework. Unlike the above frameworks, our work focuses on vertically partitioned datasets.

2.2. Vertical Multi-Party Gradient Tree Boosting Frameworks

In vertical multi-party gradient tree boosting frameworks, dataset features are shared amongst the collaborating participants. Several existing efforts have focused on addressing concerns in this setting. In [16], Cheng et al. designed a lossless privacy-preserving multi-party gradient tree boosting framework using a homomorphic encryption scheme. The framework achieves the same accuracy as the non-federated gradient tree boosting frameworks. However, it reveals the intermediate parameters during the training process, which can lead to privacy violations. In [17,18], the authors propose secure training and prediction frameworks for privacy-preserving multi-party gradient tree boosting. However, their schemes have limited scalability, i.e., they are limited to two parties. In contrast, in our work, we propose a privacy-preserving multi-party gradient tree boosting framework that is scalable and does not expose the intermediate parameters.

2.3. Generic Multi-Party Gradient Tree Boosting Frameworks

Unlike the horizontal and vertical categories, the generic multi-party gradient tree boosting frameworks support training for generically split datasets (vertical and horizontal split datasets or their combinations). A recent work [25] proposed by Deforth et al. focuses on this category. By minimizing the number of oblivious permutation evaluations using optimization techniques and employing the Manticore multi-party computation framework [26], the authors are able to achieve a scalable and secure gradient tree boosting framework. Our work focuses on using HE instead of the multi-party computation techniques employed in this work.

3. Preliminaries

This section summarizes the gradient tree boosting framework, XGBoost, and the cryptographic foundations used to construct our proposed privacy-preserving multi-party gradient tree boosting framework.

3.1. XGBoost

XGBoost is an implementation of gradient tree boosting. It trains an additive ensemble model in a sequence of iterations [19]. At each iteration, a weaker model is learned and added to the ensemble model. Thus, the ensemble model is the sum of all the weaker models. To train a model with n samples, let the feature set and label for sample i be (X_i, y_i),

and $\hat{y}$ be the model prediction. The goal is to build a model that minimizes the following objective loss function at iteration t [16,18,19].

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(X_i)) + \sum_{k=1}^{t} \Omega(f_k) \quad (1)$$

where, $\hat{y}_i{}^{(t-1)}$ denotes the predicted value of the ensemble model at iteration $t-1$, $f_t(X_i)$ denotes the predicted value of the weaker model built at the iteration t and Ω is the regularization term associated with the leaf nodes' number and weight values.

In XGBoost, the objective loss function is approximated as follows:

$$L = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(X_i) + \frac{1}{2} h_i f_t^2(X_i)] + \sum_{k=1}^{t} \Omega(f_k) \quad (2)$$

where g_i and h_i are the first- and second-order gradients of the loss function for iteration $t-1$, which are defined as follows:

$$g_i = \partial_{\hat{y}_i{}^{(t-1)}} l(y_i, \hat{y}_i{}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}_i{}^{(t-1)}} l(y_i, \hat{y}_i{}^{(t-1)}) \quad (3)$$

The sum of g_i and h_i for a node's instance set I_j can be computed as:

$$G_j = \sum_{i \in I_j} g_i \text{ and } H_j = \sum_{i \in I_j} h_i. \quad (4)$$

The optimal weight w_j^* for the leaf node is obtained as:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \quad (5)$$

where λ is the regularizer for the leaf weight.

At each iteration, i.e., during the construction of each tree, Equation (6) is used for split decisions at each intermediate node for every possible split. The split with the highest value of L_{split} is chosen.

$$L_{split} = \frac{1}{2} [\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}] - \gamma \quad (6)$$

where γ is the regularizer for the leaf number, and I_L and I_R make up the instance set for the left and right child nodes. Thus, $G_L = \sum_{i \in I_L} g_i$ and $H_L = \sum_{i \in I_L} h_i$ denote the sum of g_i and h_i for the left child node instance space, and $G_R = \sum_{i \in I_R} g_i$ and $H_R = \sum_{i \in I_R} h_i$ denote the sum of g_i and h_i for the right child node instance space.

3.2. Homomorphic Encryption (HE)

In this work, we adopt the BCP Scheme [27,28], which is an additive homomorphic encryption scheme. The scheme comprises the following algorithms:

(1) Setup (κ): For a given security parameter κ and two large primes p and q of length κ bits, the algorithm generates the public parameters (pp) and the master key (mk) as follows. First, it computes $N = pq$. It then randomly chooses $g \in \mathbb{Z}^*_{N^2}$ of order $pp'qq'$ (s.t. $g^{p'q'} = 1 + kN$ for $k \in [1, N-1]$), where $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$. The algorithm outputs pp as (N, g, k) and mk as (p', q').

(2) KeyGen (pp): Generates the public-secret key pairs for users. To generate the key pair for a user i, the algorithm randomly picks $a_i \in \mathbb{Z}_{N^2}$, and outputs the public key (pk_i) as $h_i = g^{a_i} \bmod N^2$ and the secret key (sk_i) as a_i.

(3) Enc (pp, pk_i, m)**:** Encrypts the message $m \in \mathbb{Z}_N$ under the public key pk_i. To encrypt m, the algorithm randomly chooses $r \in \mathbb{Z}_{N^2}$ and outputs the ciphertext (CT) as (A, B), where

$$A = g^r \bmod N^2 \text{ and } B = h_i^r(1 + mN) \bmod N^2.$$

(4) Dec (pp, sk_i, CT)**:** Recovers the message m from $CT = (A, B)$ using the corresponding $sk_i = a_i$. The recovery is performed as follows:

$$m = \frac{B/(A^{a_i}) - 1 \bmod N^2}{N}. \tag{7}$$

Note that the above recovery is successful only for the pk_i-sk_i pair.

(5) mDec (pp, pk_i, mk, CT)**:** Recovers any properly created ciphertext using the master key, i.e., the algorithm can decrypt CT encrypted under any users' public key pk_i (so long as pk_i is legitimate). For the decryption to proceed, first $a \bmod N$ and $r \bmod N$ are computed as:

$$a \bmod N = \frac{h^{p'q'} - 1 \bmod N^2}{N} \cdot k^{-1} \bmod N \tag{8}$$

and

$$r \bmod N = \frac{A^{p'q'} - 1 \bmod N^2}{N} \cdot k^{-1} \bmod N, \tag{9}$$

where k^{-1} is the inverse of $k \bmod N$. m is recovered from CT as:

$$m = \frac{(B/(g^{\gamma}))^{p'q'} - 1 \bmod N^2}{N} \cdot \delta \bmod N, \tag{10}$$

where δ is the inverse of $p'q' \bmod N$ and $\gamma := ar \bmod N$.

3.3. BCP Sub-Protocols

To perform arithmetic operations homomorphically, Refs. [27–29] proposed the following sub-protocols.

(a) `KeyProd`: The sub-protocol transforms the encryptions under the different user public keys $pk_1, \ldots, pk_n$ to encryptions under a joint public key $pk_\Sigma := \prod_{i=1}^{n} pk_i$. The transformation is an interactive process that involves two non-colluding servers (see [28] for the details). The encryption under the joint public key pk_Σ can only be decrypted using the sum of all the user secret keys $sk_\Sigma = \sum_1^n sk_i$ or the master key.

(b) `Add`: Returns the encrypted sum of two encrypted messages. Suppose two messages m_1 and m_2 are encrypted as $[\![m_1]\!]$ and $[\![m_2]\!]$, respectively. $[\![.]\!]$ denotes an encryption operation under the joint public key in our case. The `Add` sub-protocol sums the two ciphertexts as $[\![m_1 + m_2]\!]$. The fact that the BCP HE scheme achieves its additive nature under the same public key straightaway simplifies the `Add` sub-protocol. The encryptions under different user public keys can first be transformed to be under the same public key using the `KeyProd` sub-protocol, then followed by their addition. Thus, the sum of two encryptions (A, B) and (A', B') under the same public key can be computed as (see [28] for the details):

$$(\bar{A}, \bar{B}) \leftarrow (A \cdot A' \bmod N^2, B \cdot B' \bmod N^2).$$

(c) `Mult`: The `Mult` sub-protocol returns the encrypted product of two ciphertexts. The process involves an interaction between two non-colluding servers (see [28] for the details). Thus, given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$, the `Mult` sub-protocol returns $[\![m_1 \times m_2]\!]$.

(d) `TransDec`: The `TransDec` sub-protocol does the opposite of the `KeyProd` sub-protocol. It transforms the encryptions under the joint public key pk_Σ to encryptions under the user public keys $pk_1, \ldots, pk_n$. The transformation process involves interactions between two non-colluding servers (see [28] for the details).

(e) `Neg`: The `Neg` sub-protocol negates an encrypted message. For example, given an encryption of a message m as $[\![m]\!]$, the `Neg` sub-protocol transforms it to $[\![-m]\!]$ as (see [27,29] for the details):

$$\left(g^{(N-1)\cdot r} \bmod N^2, h^{(N-1)\cdot r}(1+m\cdot N)^{N-1} \bmod N^2\right) = [\![-m]\!].$$

(f) `Exp`: Using the same principles as in (e) above, the `Exp` sub-protocol returns the product of an encrypted message $[\![m]\!]$ and a constant κ as $[\![\kappa m]\!]$. It can be computed as below:

$$\left(g^{(N+\kappa)\cdot r} \bmod N^2, h^{(N+\kappa)\cdot r}(1+m\cdot N)^{N+\kappa} \bmod N^2\right) = [\![\kappa m]\!].$$

The correctness is similar to the `Neg` sub-protocol.

(g) `Sub`: The `Sub` sub-protocol returns the difference between two ciphertexts. For example, given m_1 and m_2 encrypted as $[\![m_1]\!]$ and $[\![m_2]\!]$, respectively, the `Sub` sub-protocol returns $[\![m_1 - m_2]\!]$. We describe the sub-protocol as follows: First, $[\![m_2]\!]$ is negated using the `Neg` sub-protocol. Then, the `Add` is used to complete the process. Thus,

$$[\![m_1 - m_2]\!] = \texttt{Add}([\![m_1]\!], \texttt{Neg}([\![m_2]\!])).$$

(h) `LGT`: The less than or greater than (LGT) sub-protocol shows the relationship between two ciphertexts, i.e., $[\![m_1]\!] \geq [\![m_2]\!]$ or $[\![m_1]\!] < [\![m_2]\!]$. The sub-protocol returns 1 if $[\![m_1]\!] < [\![m_2]\!]$, it returns 0 otherwise. It is an adaptation of the SLT protocol in [29]. A detailed description is presented in Appendix A.

4. Proposed Computation Primitives

Building gradient tree boosting algorithms requires more complicated computation primitives such the *division* and *argmax*. The division sub-protocol based on BCP scheme proposed in [29] is inefficient and unsuitable for our setting. Thus, we propose two sub-protocols `Div` and `Sargmax` based on the BCP HE scheme to perform the *division* and *argmax* operations, respectively.

4.1. `Div`

The `Div` sub-protocol outputs the encrypted division of two ciphertexts. Given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$, the `Div` sub-protocol returns $[\![m_1/m_2]\!]$, m_1 being the nominator and m_2 the denominator. The protocol is run interactively between two non-colluding servers, say server C and server S, as illustrated in Figure 1.

Using the `Exp` sub-protocol, the server C first masks the ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$ as $[\![\tau_1 m_1]\!]$ and $[\![\tau_2 m_2]\!]$, respectively, where $\tau_1, \tau_2 \in \mathbb{Z}_N$. The server C then sends $[\![\tau_1 m_1 + \tau_2 m_2]\!]$ and $[\![\tau_1 m_2]\!]$ to the server S. The two ciphertexts are decrypted by the server S. In plaintext, the server S performs $\frac{1}{\tau_1 m_2} \times (\tau_1 m_1 + \tau_2 m_2)$ and encrypts the result and sends it back to the server C. The server C extracts $[\![\frac{m_1}{m_2}]\!]$ by subtracting $[\![\frac{\tau_2}{\tau_1}]\!]$ out of the result received from the server S. See Appendix B for the proof of correctness.

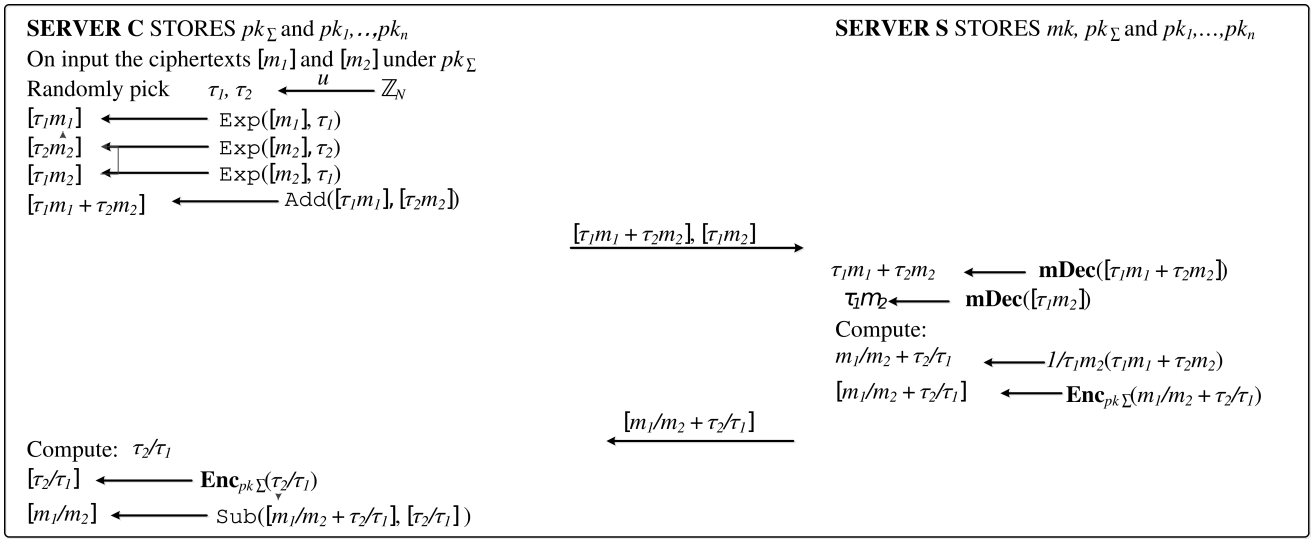


Figure 1. Illustration of the `Div` sub-protocol.

4.2. `Sargmax`

The `Sargmax` sub-protocol returns the arguments of the maximum value. In our case, the maximum value returned is in encrypted form. The maximum encrypted value is obtained using the `LGT` sub-protocol. Once the maximum value is obtained, its associated arguments are returned as the `Sargmax` sub-protocol's result. The details are shown in Algorithm 1.

Algorithm 1: Secure `argmax` Computation Algorithm

```
function Sargmax(dict)
    //Input: dict -is a dictionary of encrypted values
    max=None
    for key in dict.keys() do
        if max==None then
            max = dict[key]
        else
            //Securely compare the encrypted values
            LGT(max, dict[key])
            if max>dict[key] then
                max=max
            else
                max=dict[key]
            end
        end
    end
    return key
end
```

5. Overview of Our Proposed Framework

In this section, we first describe the involved entities, followed by a security assumption and the workflow of our proposed framework.

5.1. Entities of the Framework

Our proposed framework comprises three types of entities: a set of participants, and two servers S and C, as illustrated in Figure 2.

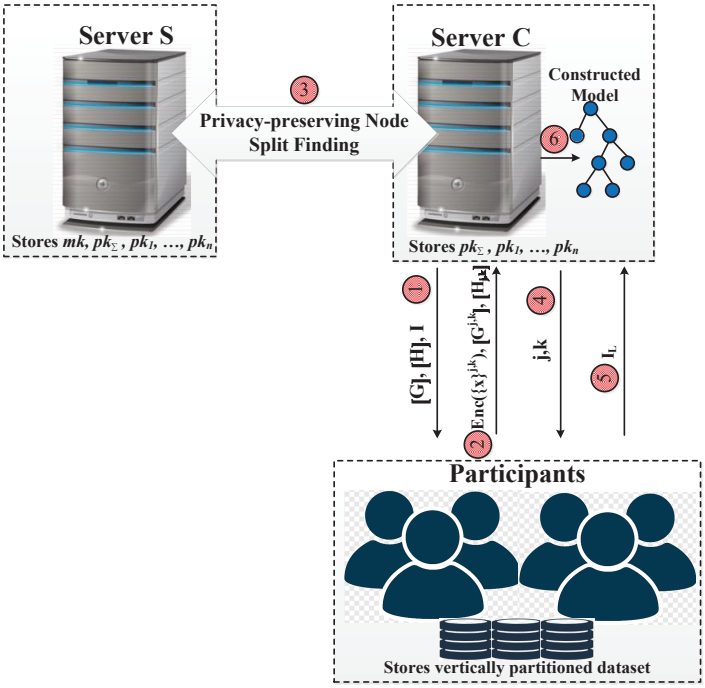


Figure 2. Entities of our proposed framework.

Participants: Participants are volunteers willing to take part in multi-party gradient tree boosting model learning. In this work, each participant holds a portion of a vertically partitioned dataset. We refer to the participant holding the label feature as Label Bearing Participant (LBP). There is only one LBP. Each participant only interacts with Server C. Additionally, each participant holds a public-private key pair.

Server S: S holds pk_Σ, mk and $pk_1, \ldots, pk_n$. Thus, S can decrypt any legitimately encrypted message. However, S only communicates directly with server C, i.e., it does not directly access the participants' data. It mainly helps with decrypting masked data from server C. We assume that S is *honest-but-curious* [30–32].

Server C: C holds pk_Σ and $pk_1, \ldots, pk_n$. C directly communicates with all the other entities. It has access to the instance space and encrypted intermediate parameters received from the participants. C and S then collaboratively perform computations on the received parameters to build the model. All the data from C to S are masked to prevent S from observing the actual contents of the data. We also assume that C is *honest-but-curious*. An example of C is a cloud server.

5.2. Security Assumptions

We assume a semi-honest and non-colluding security model. In other words, all the entities follow the protocols but are curious about the other entities' inputs and parameters. Furthermore, the entities do not collude with each other. We also assume that the communication channels between the entities are secure, i.e., no information is revealed during its transmission.

5.3. Workflow of Our Proposed Scheme

The general workflow of our proposed SSXGB training is shown in Algorithm 2. As shown in Algorithm 2, three major protocols namely: LbpXgbTrain, SBuildTree and SPredTree are invoked during the model learning. To prevent inference attacks on label information, we adopt the second proposal of [16], where the first tree is built by the LBP. The LbpXgbTrain protocol is thus executed by the LBP for the above purpose.

Algorithm 2: Scalable and Secure XGBoost Training

function SxgbTrain(X, Y)
  //Input: X:$\{X^i\}_{i=1}^n$ is an aggregation of X^i from n participants.
  //Input: Y is the label borne by the LBP
  for $t = 1, \ldots, T$ **do**
    if *TreeList*==*Empty* **then**
      LbpXgbTrain(X^{lbp}, Y)
    else
      Compute: $[\![G_{t-1}]\!]$: $\sum_i([\![g_{t-1}]\!])$ and $[\![H_{t-1}]\!]$: $\sum_i([\![h_{t-1}]\!])$
      //Construct a tree using $[\![G_{t-1}]\!]$ and $[\![H_{t-1}]\!]$
      F_t = SBuildTree($[\![G_{t-1}]\!], [\![H_{t-1}]\!]$)
      //Predict using the current tree
      $[\![\hat{Y}_t]\!]$ = SPredTree(F_t, X)
      TreeList.append(F_t)
      $[\![\hat{Y}]\!]$= $[\![\hat{Y}]\!]$+ $[\![\hat{Y}_t]\!]$
    end
  end
  return TreeList
end

Once the LbpXgbTrain protocol is executed, the returned parameters are encrypted and sent to server C for the rest of the participants to join the process. The two protocols SBuildTree and SPredTree are then iteratively executed by all the participants and the servers to complete the model learning process. At each iteration t, the SBuildTree is invoked to securely build a tree, while the SPredTree is invoked to make predictions using the built tree at t. Finally, the protocol returns a trained model TreeList. The details are presented in the subsequent sections.

6. Scalable and Secure Multi-Party XGBoost Building

This section presents the building of our proposed SSXGB model over the vertically partitioned dataset. We specify that all participants bear distinct sets of data features. The LBP bears the label feature. We also assume that the participants have the same samples. We emphasize that server C operates only on encrypted data and does not have direct access to the participants' data, including the label.

6.1. The First Secure Tree Building by LBP

As stated in the previous section, to prevent participants from inferring on the label information, we adopt the proposal of [16] in which the LBP builds the first tree. The LbpXgbTrain function shown in Algorithm 3 is invoked for that purpose. The LbpXgbTrain function takes as input the dataset (X^{lbp}, Y), where X^{lbp} is the feature matrix of the LBP and Y is the label information. In other words, the LBP does not require the other participants' data to build the first tree. Since there is no collaboration in building the first tree, the sub-routines: ComputeBaseScore, BuildTree and PredTree for computing the base score, building the first tree and making the initial predictions, respectively, are consistent with the mechanisms of XGBoost [19].

Algorithm 3: The First Tree Building by the LBP

function LBPXGBTRAIN(X^{lbp}, Y)
  //Input: X^{lbp}:$\{x^{i,j}_{M\times\mathcal{N}_{lbp}}\}$ where $\mathcal{N}_{lbp}$ is the number of features borne by the LBP
  //Input: Y:$\{y\}_{i=1}^{M}$ is the label
  //Compute the base score
  F_0 = COMPUTEBASESCORE(Y)
  TREELIST = []
  //Initial prediction
  $\hat{Y} = F_0$
  Compute: G_0:$\sum_i g_0$ and H_0:$\sum_i h_0$
  //Construct a tree using G_0 and H_0
  F_1 = BUILDTREE(G_0, H_0)
  //Predict using the tree F_1
  $\hat{Y}_1$ = PREDTREE(F_1, X^{lbp})
  $\hat{Y} = \hat{Y} + \hat{Y}_1$
  //Encrypt the base score and the tree node values, and update the model list
  TREELIST.append(Enc(F_0)$_{pk_{lbp}}$, Enc(F_1)$_{pk_{lbp}}$)
  //Encrypt the label information and the updated prediction matrix
  Enc(Y)$_{pk_{lbp}}$, Enc($\hat{Y}$)$_{pk_{lbp}}$
  return TREELIST, Enc(Y)$_{pk_{lbp}}$, Enc($\hat{Y}$)$_{pk_{lbp}}$
end

Once the first tree is constructed, LBP encrypts the base score and the node values of the tree with its public key (pk_{lbp}). Next, it updates the model with the encrypted base score and the tree. It also encrypts the label information and the prediction matrix with its public key. Finally, it returns TREELIST, Enc(Y)$_{pk_{lbp}}$ and Enc($\hat{Y}$)$_{pk_{lbp}}$. The returned parameters are sent to server C to continue with the model training.

6.2. Secure BUILDTREE and PREDTREE

Once the first tree is built by the LBP and the results are returned to server C, the rest of the participants can join to continue with the model training. First, the servers C and S collaboratively transform the parameters from the LBP encrypted under the public key pk_{lbp} to be under the joint public key pk_{Σ} using the `KeyProd` sub-protocol discussed in Section 3. For example, the encrypted label information Enc(Y)$_{pk_{lbp}}$ is transformed to $[\![Y]\!]$, the encrypted prediction matrix Enc($\hat{Y}$)$_{pk_{lbp}}$ is transformed to $[\![\hat{Y}]\!]$, etc. Then server C computes the first- and second-order derivatives using the encrypted parameters for all the instances. Next, the joint model building resumes using the SBUILDSPREDTREE protocol shown in Algorithm 4.

The SBUILDSPREDTREE protocol is executed by multiple entities. The algorithm takes in as input the encrypted first- and second-order derivatives computed by server C. The execution begins in server C, which assigns the current node as the root node if there exists no root node. Next, each participant, including the LBP, proposes split candidate values for each of their features as in [19]. For ease of understanding, we shall refer to all the participants, including the LBP as participant i. For each split candidate, each participant i computes $[\![G_{t-1}]\!]$, $[\![H_{t-1}]\!]$, which are associated with the left branch according to [19]. Each participant i also creates a lookup table to record the split candidate information. Then, for each split candidate, each participant i sends the parameter tuple $\mathcal{T} = ((j,k), [\![G^{j,k}_{t-1}]\!], [\![H^{j,k}_{t-1}]\!])$ to the server C.

Algorithm 4: Secure BUILD and PRED TREE

function SBUILDSPREDTREE($[\![\mathrm{G}_{t-1}]\!]$, $[\![\mathrm{H}_{t-1}]\!]$)
  //Input:- $[\![\mathrm{G}_{t-1}]\!]$:$\{[\![g_{t-1}]\!]\}_{i=1}^{M}$, $[\![\mathrm{H}_{t-1}]\!]$:$\{[\![h_{t-1}]\!]\}_{i=1}^{M}$
  /* *Computed by* SERVER *C* */
  if *ROOTNODE==None* **then**
    //Register the current node as the root node
    ROOTNODE=CURRENTNODE
  end
  /* *Computed at each* PARTICIPANT *i* */
  foreach *feature j* **do**
    Propose split candidates $\{x\}^{j,0}, \ldots, \{x\}^{j,K}$
  end
  foreach $\{x\}^{j,k}$ **do**
    Compute $[\![\mathrm{G}_{t-1}^{j,k}]\!]$ and $[\![\mathrm{H}_{t-1}^{j,k}]\!]$
    Create a lookup table and record j, k and $\{x\}^{j,k}$ in the table
    Create a tuple (j, k)
  end
  Send the tuple $((j, k)$, $[\![\mathrm{G}_{t-1}^{j,k}]\!]$, $[\![\mathrm{H}_{t-1}^{j,k}]\!])$ to SERVER C
  /* *Computed by* SERVER *C* */
  j_{opt}, k_{opt} = SSPLITNODE($[\![\mathrm{G}_{t-1}^{j,k}]\!]$, $[\![\mathrm{H}_{t-1}^{j,k}]\!]$, $\mathcal{T}$)
  /* *Computed at the optimal* PARTICIPANT */
  Receive the optimal j_{opt}, k_{opt} from SERVER C
  Check the lookup for $\{x\}^{j,k}$ associated with j_{opt}, k_{opt}
  Partition I based on $\{x\}^{j,k}$
  Record the instance space I_L with SERVER C
end

Server C then securely identifies the optimal split feature and value using the SSPLITNODE algorithm and sends the result to the participant bearing the feature (optimal participant). The optimal participant then decrypts the optimal value for the optimal feature and splits the current node's instance space accordingly. Finally, the optimal participant registers the left branch instance space I_L with server C after the partition.

6.3. Secure Node Split Decision

From Equation (6), it can be observed that the optimal split can be obtained if the values of G_L and H_L, and G_R and H_R can be obtained. Hence, the secure split finding algorithm SSPLITNODE shown in Algorithm 5 takes as input the first and second encrypted derivatives $[\![G_{t-1}]\!]$ and $[\![H_{t-1}]\!]$ and the parameter tuple $\mathcal{T}$. In this context, we simply use $[\![G]\!]$ and $[\![H]\!]$ for $[\![G_{t-1}]\!]$ and $[\![H_{t-1}]\!]$, respectively.

First, the algorithm returns and stores the encrypted prediction matrix if the current node is a leaf node (shown in lines 4–9 of Algorithm 5). Otherwise, the algorithm proceeds to securely identify the optimal score. It enumerates all the participants, their features and the proposed encrypted split candidates for each of the features. For each proposed split candidate, the algorithm computes an encrypted gain (shown in lines 21–30 of Algorithm 5). The encrypted gains for all the proposed split candidates are stored in a dictionary. The algorithm then executes the *Sargmax* primitive algorithm to identify the optimal feature j_{opt} and the threshold value k_{opt}. Next, server C sends the optimal parameters j_{opt} and k_{opt} to the optimal participant bearing the pair $j_{opt} - k_{opt}$. Then server C receives I_L from the optimal participant and uses it to split its instance space into I_L and I_R. Server C then stores the current node and associates it with the optimal participant.

Algorithm 5: Securely Finding Node Split

function SSplitNode($[\![G_{t-1}]\!]$, $[\![H_{t-1}]\!]$, $\mathcal{T}$)
  //Input:- $[\![\mathrm{G}_{t-1}]\!]$:$\{[\![g_{t-1}]\!]\}_{i=1}^{M}$, $[\![\mathrm{H}_{t-1}]\!]$:$\{[\![h_{t-1}]\!]\}_{i=1}^{M}$, and $\mathcal{T}$.
  /*Collaboratively Computed by Server S and Server C*/
  if *CurrentNode*==*LeafNode* **then**
    //Compute the weight of the leaf node
    $[\![w^*]\!] = \mathtt{Div}(\Sigma_i\{[\![g]\!]\}^i, (\Sigma_i\{[\![h]\!]\}^i + [\![\lambda]\!]))$
    $\{[\![\hat{y}]\!]\}^i = [\![w^*]\!]$
    return $[\![\hat{\mathrm{Y}}]\!]$
  end
  //Compute the CurrentNode's gain (*cgain*)
  $[\![G]\!] = \Sigma_i\{[\![g]\!]\}^i$
  $[\![H]\!] = \Sigma_i\{[\![h]\!]\}^i$
  $[\![cgain]\!] = \mathtt{Div}(\mathtt{Mult}([\![G]\!], [\![G]\!]), ([\![H]\!] + [\![\lambda]\!]))$
  //Initialize the gain dictionary
  $gainDict = \{\}$
  //Enumerate all the Participants
  for $p = 0, \ldots, P$ **do**
    //Enumerate all the features of a Participant
    for $j = 0, \ldots, J$ **do**
      //Enumerate all the proposed thresholds
      for $k = 0, \ldots, K$ **do**
        Receive $[\![G_L]\!]$ and $[\![H_L]\!]$ from a Participant
        //Compute the first derivative for the right branch
        $[\![G_R]\!] = \mathtt{Sub}([\![G]\!], [\![G_L]\!])$
        //Compute the second derivatives for the right branch
        $[\![H_R]\!] = \mathtt{Sub}([\![H]\!], [\![H_L]\!])$
        //Compute gains
        $[\![lgain]\!] = \mathtt{Div}(([\![G_L]\!])^2, ([\![H_L]\!] + [\![\lambda]\!]))$
        $[\![rgain]\!] = \mathtt{Div}(([\![G_R]\!])^2, ([\![H_R]\!] + [\![\lambda]\!]))$
        $[\![gain]\!] = lgain + (\mathtt{Sub}(rgain, cgain))$
        //Update the gain dictionary
        $gainDict[(p, j, k)] = [\![gain]\!]$
      end
    end
  end
  $\mathrm{j}_{optimal}, \mathrm{k}_{optimal} = \underset{j,k}{\mathrm{Sargmax}}(gainDict)$
  return $\mathrm{j}_{opt}, \mathrm{k}_{opt}$ to optimal participant p
  Server C receives I_L from optimal participant p
  Server C partitions its instance space into I_L and I_R
  Server C associates the CurrentNode with the optimal participant p as [p : $Node^{j,k}$]
end

7. Secure Prediction

Our proposed secure prediction algorithm is collaboratively executed by all the entities, as shown in Algorithm 6. The SPredict algorithm takes as input the trained model TreeList and a record to be predicted $x_{1\times\mathcal{N}}^{i,j}$ with N number of features. Suppose $x_{1\times\mathcal{N}}^{i,j}$ is held by a *client c* with a public key pk_c. To make predictions on the record, the client first encrypts the record with his public key as $\mathrm{Enc}(x_{1\times\mathcal{N}}^{i,j})_{pk_c}$ and sends it to server C. This preserves the privacy of the record from server C. Next, server C compares and passes the record

down the tree, starting from the root node. At each node, server C identifies the participant p holding the node, and the j, k pair associated with the node. Server C then transforms the record value for the feature j of the participant p's to be under the public key of the participant p as $\text{Enc}(x^{i,j})_{pk_p}$ (shown in lines 12–17 of Algorithm 6). Server C then sends $\text{Enc}(x^{i,j})_{pk_p}$ to the participant p. Next, p decrypts the $\text{Enc}(x^{i,j})_{pk_p}$ using his secret key and compares the value with the node's threshold in plaintext. Depending on the comparison result, the participant decides on whether to follow the left or the right child nodes of the current node and sends the decision to server C. Server C repeats the process until a leaf node is reached. Once a leaf node is reached, the algorithm returns the encrypted weight $[\![w]\!]$ of the leaf node stored in server C as its result. The final prediction result is obtained by cumulating the predictions of all the trees in TreeList.

Algorithm 6: Secure Prediction Algorithm

1: **function** SPredict(TreeList, $x^{i,j}_{1\times\mathcal{N}}$)
2: //Input:-TreeList and $x^{i,j}_{1\times\mathcal{N}}$, where $\mathcal{N}$ is the number of features for the record
3: /* Computed by all the entities */
4: //Client c encrypt the values of the record
5: $\text{Enc}(x^{i,j}_{1\times\mathcal{N}})_{pk_c}$
6: Send the encrypted record to the Server C
7: **while** *True* **do**
8: //Start from the RootNode
9: **if** *CurrentNode* ! = *LeafNode* **then**
10: Server C identifies feature j for the split at CurrentNode
11: Server C identifies the Participant p bearing the feature j
12: //Transform the encryption to be under the public key of the Participant p
13: $\text{Enc}(x^{i,j}+r)_{pk_c} \leftarrow \text{Enc}(x^{i,j})_{pk_c} + \text{Enc}(\text{r})_{pk_c}$ // $r \leftarrow \mathbb{Z}_N$
14: Server C send $\text{Enc}(x^{i,j}+r)_{pk_c}$ to Server S
15: Server S decrypts $\text{Enc}(x^{i,j}+r)_{pk_c}$ using mDec
16: Server S re-encrypts as $\text{Enc}(x^{i,j}+r)_{pk_p}$ and sends to Server C
17: Server C extracts $\text{Enc}(x^{i,j})_{pk_p}$ as: $\text{Enc}(x^{i,j})_{pk_p} \leftarrow \text{Enc}(x^{i,j}+r)_{pk_p} - \text{Enc}(\text{r})_{pk_p}$
18: //Collaboration with the participant bearing the optimal feature for the current node
19: Server C sends $\text{Enc}(x^{i,j})_{pk_p}$ to the Participant p bearing the feature j
20: Participant p decrypts $\text{Enc}(x^{i,j})_{pk_p}$ and compares it with the threshold value at the node
21: Based on whether $x^{i,j}$ is greater or less than the threshold, Participant p decides on the tree branch to follow
22: Participant p forwards the decision to Server C to continue with the process at NextNode
23: //Update the CurrentNode
24: CurrentNode = NextNode
25: **else**
26: Return the encrypted weight $[\![w]\!]$ of the LeafNode
27: **end**
28: **end**

8. Analysis

8.1. Security Analysis

We consider the semi-honest (and non-colluding) model in our security analysis, i.e., we consider the scenario where all the entities adhere to the protocols but try to gather information about the other entities' input and intermediate parameters as much as they can.

8.1.1. Security of BCP Sub-Protocols

First, we present the security analysis of all the sub-protocols used in this work. The security of the `KeyProd`, `Add`, `Mult`, `TransDec`, `Sub`, `Exp` and `Neg` sub-protocols under the semi-honest model have already been proven in [27,28].

`Div`: Similar to the other sub-protocols, the security of the `Div` sub-protocol is based on blinding or masking the plaintext. Given the ciphertexts (numerator and denominator), we employ the properties of the homomorphic encryption to blind the ciphertexts with random elements. These random elements serve as keys. When server S decrypts the ciphertexts, without knowing the random blinding elements, it cannot obtain any information about the nominator and the denominator. They look random. On the other hand, since server C does not have access to the decryption key, it also does not obtain any information about the ciphertexts. Note that we assume the two servers do not collude. Thus, the `Div` sub-protocol is secure in the semi-honest and non-colluding security model.

`LGT` and `Sargmax`: The security of the `LGT` sub-protocol is based on the fact that server S only computes the difference between two data values. Thus, server S obtains no information about the actual data. However, server C can learn the relational information of the comparison process without learning the actual data. This is a partial information leakage and makes the sub-protocol partially secure in the semi-honest and non-colluding model. Therefore, our proposed `Sargmax` sub-protocol, which relies on the `LGT` is also partially secure in the semi-honest and non-colluding model.

8.1.2. Security of SSXGB

The security analysis of the SSXGB can be split into three parts: server S part, server C part and participant part.

Server S part: Server S does not have access to the sample and feature space. It only collaborates with Server C in performing computations. However, the computations performed by server S are on masked values. Thus, no information is leaked to server S.

Server C part: Server C has access to the sample and feature space, and it stores leaf nodes. It also knows which participant holds which feature. However, the intermediate values it has access to are encrypted, and it only performs computations on the encrypted values. Although it stores the leaf nodes, the leaf values are kept in encrypted form. Thus, no information on the actual leaf values is leaked to server C. During the node split process, illustrated by Algorithm 5, comparison results leak to server C. However, server C does not learn the actual values being compared, thus, making it difficult to re-construct the feature values from the leaked information. We, therefore, believe that the benefits of this framework outweigh this partial leakage.

Participant part: Each participant has access to the sample space for each split. Each participant knows the intermediate nodes it holds. However, each participant does not know the actual values of the intermediate parameters apart from the LBP immediately after the construction of the first tree. The participants also do not have access to the leaf nodes. Thus, no information leaks to the participants.

Since there is no information leakage in the involved entities, the proposed SSXGB is secure in the semi-honest and non-colluding model.

8.2. Communication Overhead Analysis

We analyze the communication overhead in terms of analyzing the communication costs associated with a single split. Here, we look at *server C-participant* and *server C-server S* communication costs.

server C-participant communication cost: The *server C-participant* communication cost is similar to that of [16]. Given ζ as the ciphertext size, n as the number of samples for the current node, q as the number of samples in a bucket and d as the number of features held by a participant, the communication cost can be computed as $2 \times \zeta \times d \times (n/q)$.

server C-server S communication cost: During each split, the *server C-server S* communication cost can be computed as $12 \times \zeta + (3 \times \zeta \times (n/q) \times D)$, where D is the total number of features. Our proposed SSXGB experiences fairly heavy communication overhead during the collaborative computation of gains by the two servers.

9. Performance Evaluation

This section presents experiments to demonstrate the effectiveness and efficiency of the proposed SSXGB.

9.1. Experimental Setup

9.1.1. Hardware and Software

All the experiments were performed using a desktop computer with Intel Core i5-6500 CPU with 3.20 GHz $\times$ 4 speed and 16 GB RAM running an Ubuntu 20.04 operating system (Canonical Ltd., London, UK). We used Python 3.8.5 and gmpy2 library for the implementation. We also used Cython 0.29.23 to speed up sections of the code. The participant and server functionalities were all executed on the same computer. Thus, latency is ignored in the experiments.

9.1.2. Datasets

We experimented using two datasets: Iris [33] and MNIST [34]. The Iris dataset comprises three classes of iris plants, each with 50 instances. Each instance bears the features of sepal length, sepal width, petal length and petal width. Thus, the dataset contains 150 instances with 4 features (150 $\times$ 4).

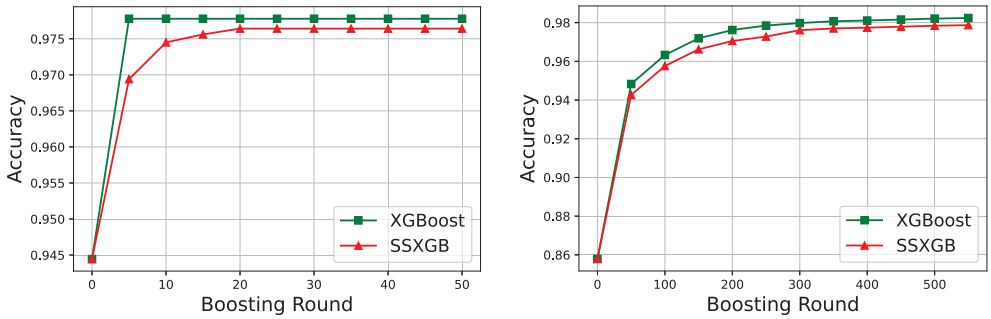
The MNIST dataset consists of 70,000 samples of gray-scale handwritten images of digits (0–9). The training set contains 60,000 samples, while the test set contains 10,000 samples. Each gray-scale sample has 28 $\times$ 28 (784) pixels. Thus, the training set is (60,000 $\times$ 784) and the test set is (10,000 $\times$ 784).

9.2. Evaluation of SSXGB

First, we evaluate the regression effectiveness of the proposed SSXGB with the two datasets. The effectiveness is measured in terms of regression accuracy and loss.

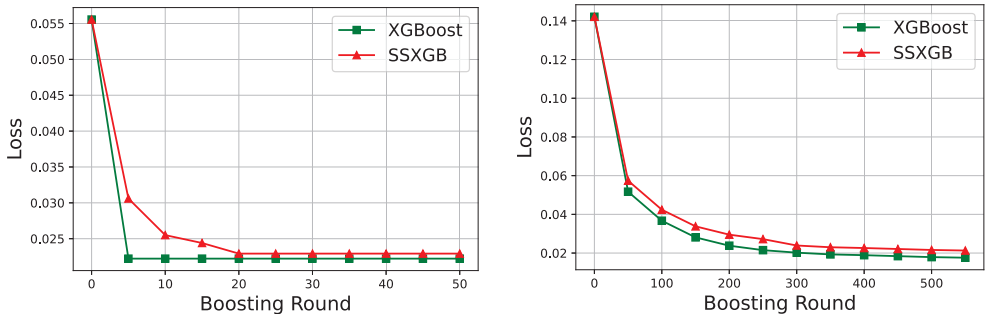
Setup Configuration: We simulated the two servers, four (4) participants for the Iris dataset and sixteen (16) participants for the MNIST dataset. In each case, one of the participants serves as an LBP. We partitioned the datasets vertically and shared the features between the participants, i.e., for the Iris dataset, each participant held one (1) feature and for the MNIST dataset, each participant held forty-nine (49) features (Each image sample of MNIST contains $28 \times 28 = 784$ pixels, and we consider each pixel to be a feature. Thus, each image sample contains 784 features. When partitioned equally among 16 participants, each participant ends with 49 features). Since we performed regression analysis, we considered only two classes. Thus, we take "Iris-setosa" as positive for the Iris dataset and the rest of the classes as negative. To make the dataset balanced, we synthetically generated 50 more instances of the "Iris-setosa" class. Meanwhile, for the MNIST dataset, we take the digits 0–4 as positive and 5–9 as negative. During each simulation, we set the learning rate as 0.08, and the sampling rate as 0.8 for both samples and features. We set maximum depths as 6 and 4 for the MNIST and Iris datasets experiments, respectively. We employed the approximated sigmoid function in [35].

Results: Figure 3 presents the accuracy and loss against the training round with the Iris and MNIST datasets. We can observe that the proposed SSXGB converges slowly in comparison with the XGBoost scheme, and it is slightly less effective as compared to the XGBoost, i.e., the accuracy of 0.9789 vs. 0.9826 for the MNIST dataset and 0.97712 vs. 0.97778 for the Iris dataset. The accuracy loss in SSXGB can be attributed to the use of the approximated sigmoid function and the restrictive nature of training the first tree. However, the slight loss in accuracy is totally acceptable.



(**a**) Accuracy with Iris

(**b**) Accuracy with MNIST



(**c**) Loss with Iris

(**d**) Loss with MNIST

Figure 3. Training convergence with the Iris and MNIST datasets.

9.3. Privacy-Preservation Computation Overhead

Since HE is crucial for privacy preservation in our proposed SSXGB, identifying a suitable configuration for the BCP scheme was paramount. Thus, we implemented the BCP scheme and measured the computation times of its algorithms and sub-protocols under different key sizes. For each operation, the experiment was repeated 20 times and we obtained the average computation times.

Results: The results are shown in Table 1. We can see that the computation times increase with the increase in key size for all the algorithms and sub-protocols. Amongst the BCP algorithms, the **mDec** is computationally the most demanding, while encryption is computationally the least demanding. Meanwhile, amongst the sub-protocols, the `Mult` is computationally the most demanding while `KeyProd` is computationally the least demanding. For the `KeyProd`, we only considered a joint public key from two users and the joint public key generation does not involve encryption of data as in [28]. Furthermore, for the `TransDec`, we considered only two users. Based on the shown computation times and security requirements, we considered the key size of 1024 during the implementation of the SSXGB.

Much as the HE provides the required security and utility, it significantly increases the computation overhead. Adopting a differential privacy mechanism [36] can be considered a possible solution, so long as the right balance between utility and security is identified, which can be challenging.

Table 1. Computation times of BCP algorithms and sub-protocols.

Operation	Key Size (Bits)		
	512	**1024**	**2048**
Enc	10.61 ms	12.02 ms	16.98 ms
Dec	18.27 ms	29.13 ms	38.64 ms
mDec	29.34 ms	40.96 ms	63.72 ms
`KeyProd`	0.02 ms	0.05 ms	0.11 ms
`Add`	8.86 ms	10.23 ms	12.61 ms
`Mult`	180.74 ms	259.97 ms	310.03 ms
`TransDec`	161.20 ms	196.69 ms	283.74 ms
`Exp`	2.04 ms	2.32 ms	4.01 ms
`Sub`	8.93 ms	12.06 ms	13.02 ms
`LGT`	120.17 ms	143.72 ms	171.05 ms
`Div`	140.29 ms	183.18 ms	251.67 ms

9.4. Efficiency of SSXGB

Finally, we investigate the efficiency of the proposed SSXGB by examining its running time. We performed the investigation under a varying number of participants. We considered the running time for training using the two datasets. Figure 4 shows the running time of training using the MNIST and Iris datasets against the varying number of participants. Since we intended to examine the running time comparatively for the datasets, and the Iris dataset has only 4 features, we limited the number of participants to only 4. In Figure 4, it can be observed that there is no significant change in running time against the varying number of participants. We can attribute this to the number of features and samples remaining almost the same under the vertically partitioned datasets; hence, the number of feature and sample operations remain fairly constant. However, the running time is higher for the MNIST dataset compared to the Iris dataset. This is mainly because the MNIST dataset has more features and samples as compared to the Iris dataset.

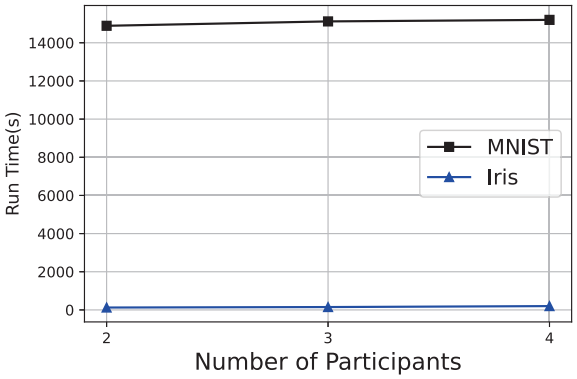


Figure 4. Running time of the SSXGB with the Iris and MNIST datasets.

10. Conclusions

In this work, we proposed the SSXGB framework for scalable and privacy-preserving multi-party gradient tree boosting over vertically partitioned datasets with outsourced computations. We adopted BCP HE for secure computations and proposed sub-protocols based on the BCP HE for two non-linear operations of gradient tree boosting. Analysis of the framework shows that only minimal information is leaked to the entities under the semi-honest and non-colluding security model. However, the benefits of the framework outweigh this weakness. We also implemented the secure training algorithm of the SSXGB framework. We performed experiments using two real-world datasets. The results show that the SSXGB is scalable and reasonably effective. In the future, we shall aim to minimize the communication overhead of the proposed framework and perform more complicated experiments on more real-world datasets.

Author Contributions: Conceptualization, K.E.; methodology, K.E. and J.W.K.; software, J.W.K.; validation, K.E. and J.W.K.; formal analysis, K.E.; investigation, K.E. and J.W.K.; resources, J.W.K.; data curation, K.E. and J.W.K.; writing—original draft preparation, K.E.; writing—review and editing, J.W.K.; visualization, K.E.; supervision, J.W.K.; project administration, J.W.K.; funding acquisition, J.W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF2020R1F1A1072622).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Details of the LGT Sub-Protocol

The LGT sub-protocol is an adaptation of the SLT sub-protocol in [29]. The only difference is that we intentionally reveal the result of the comparison to the server C. An illustration of the sub-protocol is shown in Figure A1. Given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$ encrypted under the joint public key pk_Σ. To determine if $[\![m_1]\!] \geq [\![m_2]\!]$ or $[\![m_1]\!] < [\![m_2]\!]$, the following procedures are collaboratively performed by servers C and S.

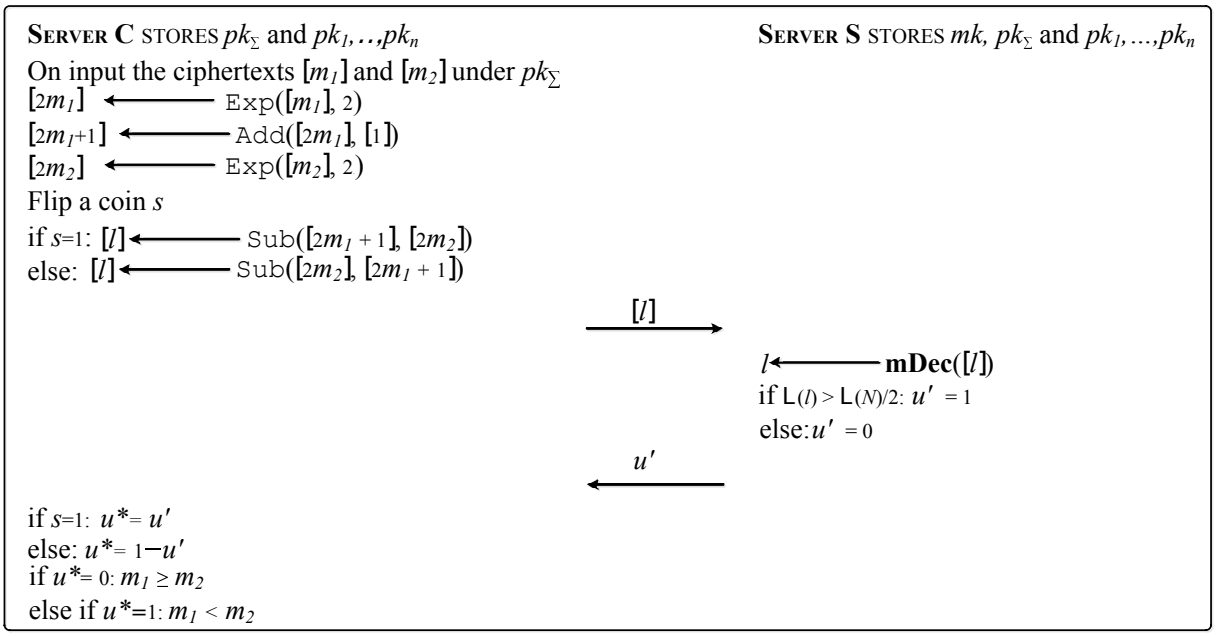


Figure A1. The LGT sub-protocol.

As in [29], first server C uses the Exp sub-protocol to multiply the ciphertexts by two as $[\![2m_1]\!]$ and $[\![2m_2]\!]$. Server C also encrypts 1 using the **Enc** algorithm as $[\![1]\!]$, and adds the result to $[\![2m_1]\!]$ as $[\![2m_1 + 1]\!]$.

Server C then flips a coin s. If $s = 1$, server C uses the Sub sub-protocol and computes $[\![l]\!]$ as $([\![2m_1 + 1]\!] - [\![2m_2]\!])$. Otherwise, $[\![l]\!]$ is computed as $([\![2m_2]\!] - [\![2m_1 + 1]\!])$. Server C then sends $[\![l]\!]$ to server S.

Next, server S uses the **mDec** algorithm to decrypt $[\![l]\!]$ as l. $\mathcal{L}(l)$ is then computed by server S as in [29] ($\mathcal{L}(x)$ denotes the bit length of x. See [29] for the details). If $\mathcal{L}(l) > N/2$, server S sets $u' = 1$, otherwise $u' = 0$. Server S then returns u' to server C.

Next, server C checks for s, and if $s = 1$, server C sets $u^* = u'$. Otherwise, $u^* = 1 - u'$. If $u^* = 0$, $m_1 \geq m_2$, otherwise $m_1 < m_2$.

Appendix B. Proof of Correctness for the Div Sub-protocol

Consider the division of $[\![m_1]\!]$ by $[\![m_2]\!]$ ($[\![m_1 \div m_2]\!]$) using the Div sub-protocol. For the purpose of this proof, we assume $[\![m_1]\!]$ and $[\![m_2]\!]$ are encrypted under the same public key.

First, the server C randomly selects $\tau_1, \tau_2 \in \mathbb{Z}_N$. Using the Exp sub-protocol, the server C generates $[\![\tau_1 m_1]\!]$, $[\![\tau_2 m_2]\!]$ and $[\![\tau_1 m_2]\!]$ as follows:

$$\begin{aligned}
[\![\tau_1 m_1]\!] &= \left(g^{\tau_1 \cdot r_1} \bmod N^2, h^{\tau_1 \cdot r_1}(1 + \tau_1 \cdot m_1 \cdot N) \bmod N^2\right) \\
[\![\tau_2 m_2]\!] &= \left(g^{\tau_2 \cdot r_2} \bmod N^2, h^{\tau_2 \cdot r_2}(1 + \tau_2 \cdot m_2 \cdot N) \bmod N^2\right) \\
[\![\tau_1 m_2]\!] &= \left(g^{\tau_1 \cdot r_3} \bmod N^2, h^{\tau_1 \cdot r_3}(1 + \tau_1 \cdot m_2 \cdot N) \bmod N^2\right)
\end{aligned} \tag{A1}$$

Using the Add sub-protocol, the server C computes $[\![\tau_1 m_1 + \tau_2 m_2]\!]$ as:

$$\begin{aligned}
[\![\tau_1 m_1 + \tau_2 m_2]\!] = \big(&g^{(\tau_1 r_1 + \tau_2 r_2)} \bmod N^2, \\
&h^{(\tau_1 r_1 + \tau_2 r_2)}(1 + (\tau_1 m_1 + \tau_2 m_2)N) \bmod N^2\big)
\end{aligned} \tag{A2}$$

The server C then sends $[\![\tau_1 m_2]\!]$ and $[\![\tau_1 m_1 + \tau_2 m_2]\!]$ to the server S. Using the **mDec** algorithm, the server S decrypts $[\![\tau_1 m_2]\!]$ and $[\![\tau_1 m_1 + \tau_2 m_2]\!]$ as $\tau_1 m_2$ and $\tau_1 m_1 + \tau_2 m_2$, respectively. In plaintext, the server S then computes $(\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1})$ as:

$$(\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}) = \frac{1}{\tau_1 m_2}(\tau_1 m_1 + \tau_2 m_2) \tag{A3}$$

Next, the server S encrypts $(\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1})$ to $[\![\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}]\!]$ as:

$$[\![\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}]\!] = \left(g^r \bmod N^2, h^r(1 + (\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}).N) \bmod N^2\right) \tag{A4}$$

and sends the result to server C.

After receiving $[\![\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}]\!]$, the server C computes $\frac{\tau_2}{\tau_1}$ in plaintext and encrypts it as $[\![\frac{\tau_2}{\tau_1}]\!]$. The server C extracts $[\![\frac{m_1}{m_2}]\!]$ using the Sub sub-protocol as:

$$[\![\frac{m_1}{m_2}]\!] = \mathtt{Sub}([\![\frac{m_1}{m_2} + \frac{\tau_2}{\tau_1}]\!], [\![\frac{\tau_2}{\tau_1}]\!]). \tag{A5}$$

References

1. Gong, M.; Feng, J.; Xie, Y. Privacy-enhanced multi-party deep learning. *Neural Netw.* **2020**, *121*, 484–496. [CrossRef] [PubMed]
2. Cock, M.d.; Dowsley, R.; Nascimento, A.C.; Newman, S.C. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, Denver, CO, USA, 12–16 October 2015; pp. 3–14.
3. Hall, R.; Fienberg, S.E.; Nardi, Y. Secure multiple linear regression based on homomorphic encryption. *J. Off. Stat.* **2011**, *27*, 669.
4. Kim, M.; Song, Y.; Wang, S.; Xia, Y.; Jiang, X. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR Med. Inform.* **2018**, *6*, e19. [CrossRef] [PubMed]
5. Aono, Y.; Hayashi, T.; Phong, L.T.; Wang, L. Privacy-preserving logistic regression with distributed data sources via homomorphic encryption. *IEICE Trans. Inf. Syst.* **2016**, *99*, 2079–2089. [CrossRef]
6. Zheng, L.; Chen, C.; Liu, Y.; Wu, B.; Wu, X.; Wang, L.; Wang, L.; Zhou, J.; Yang, S. Industrial scale privacy preserving deep neural network. *arXiv* **2020**, arXiv:2003.05198.
7. Phong, L.T. and Phuong, T.T. Privacy-preserving deep learning via weight transmission. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 3003–3015. [CrossRef]
8. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 1–19. [CrossRef]
9. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
10. Minastireanu, E.A.; Mesnita, G. Light gbm machine learning algorithm to online click fraud detection. *J. Inform. Assur. Cybersecur.* **2019**, *2019*, 263928. [CrossRef]
11. Punmiya, R.; Choe, S. Energy theft detection using gradient boosting theft detector with feature engineering-based preprocessing. *IEEE Trans. Smart Grid* **2019**, *10*, 2326–2329. [CrossRef]
12. Wang, Y.; Feng, D.; Li, D.; Chen, X.; Zhao, Y.; Niu, X. A mobile recommendation system based on logistic regression and gradient boosting decision trees. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 1896–1902.

13. Li, Q.; Wen, Z.; He, B. Practical federated gradient boosting decision trees. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 4642–4649.
14. Ong, Y.J.; Zhou, Y.; Baracaldo, N.; Ludwig, H. Adaptive Histogram-Based Gradient Boosted Trees for Federated Learning. *arXiv* **2020**, arXiv:2012.06670.
15. Liu, Y.; Ma, Z.; Liu, X.; Ma, S.; Nepal, S.; Deng, R. Boosting privately: Privacy-preserving federated extreme boosting for mobile crowdsensing. *arXiv* **2019**, arXiv:1907.10218.
16. Cheng, K.; Fan, T.; Jin, Y.; Liu, Y.; Chen, T.; Yang, Q. Secureboost: A lossless federated learning framework. *arXiv* **2019**, arXiv:1901.08755.
17. Feng, Z.; Xiong, H.; Song, C.; Yang, S.; Zhao, B.; Wang, L.; Chen, Z.; Yang, S.; Liu, L.; Huan, J. Securegbm: Secure multi-party gradient boosting. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 1312–1321.
18. Fang, W.; Chen, C.; Tan, J.; Yu, C.; Lu, Y.; Wang, L.; Wang, L.; Zhou, J.; Liu, A.X. A hybrid-domain framework for secure gradient tree boosting. *arXiv* **2020**, arXiv:2005.08479.
19. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
20. Phong, L.T.; Aono, Y.; Hayashi, T.; Wang, L.; Moriai, S. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 1333–1345. [CrossRef]
21. Shokri, R.; Shmatikov, V. Privacy-preserving deep learning. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 1310–1321.
22. Leung, C. Towards Privacy-Preserving Collaborative Gradient Boosted Decision Tree Learning. 2020. Available online: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-100.pdf (accessed on 4 August 2021).
23. Yang, M.; Song, L.; Xu, J.; Li, C.; Tan, G. The tradeoff between privacy and accuracy in anomaly detection using federated XGBoost. *arXiv* **2019**, arXiv:1907.07157.
24. Wang, Z.; Yang, Y.; Liu, Y.; Liu, X.; Gupta, B.B.; Ma, J. Cloud-based federated boosting for mobile crowdsensing. *arXiv* **2020**, arXiv:2005.05304.
25. Deforth, K.; Desgroseilliers, M.; Gama, N.; Georgieva, M.; Jetchev, D.; Vuille, M. XORBoost: Tree boosting in the multiparty computation setting. *Cryptol. ePrint Arch. Report 2021/432* **2021**. Available online: https://eprint.iacr.org/2021/432 (accessed on 15 June 2022).
26. Carpov, S.; Deforth, K.; Gama, N.; Georgieva, M.; Jetchev, D.; Katz, J.; Leontiadis, I.; Mohammadi, M.; Sae-Tang, A.; Vuille, M. Manticore: Efficient framework for scalable secure multiparty computation protocols. *Cryptol. ePrint Arch. Paper 2021/200* **2021**. Available online: https://eprint.iacr.org/2021/200 (accessed on 15 June 2022) .
27. Bresson, E.; Catalano, D.; Pointcheval, D. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 30 November–4 December 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 37–54.
28. Peter, A.; Tews, E.; Katzenbeisser, S. Efficiently outsourcing multiparty computation under multiple keys. *IEEE Trans. Inf. Forensics Secur.* **2013**, *8*, 2046–2058. [CrossRef]
29. Liu, X.; Deng, R.H.; Choo, K.R.; Weng, J. An Efficient Privacy-Preserving Outsourced Calculation Toolkit with Multiple Keys. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 2401–2414. [CrossRef]
30. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282. Available online: http://proceedings.mlr.press/v54/mcmahan17a?ref=https://githubhelp.com (accessed on 20 August 2021).
31. Truex, S.; Baracaldo, N.; Anwar, A.; Steinke, T.; Ludwig, H.; Zhang, R.; Zhou, Y. A hybrid approach to privacy-preserving federated learning. In Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, London, UK, 15 November 2019; pp. 1–11.
32. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191.
33. Fisher, R.A. Iris Data Set. 1936. Available online: https://archive.ics.uci.edu/ml/datasets/iris (accessed on 7 June 2021).
34. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
35. Chen, T.; Zhong, S. Privacy-preserving backpropagation neural network learning. *IEEE Trans. Neural Netw.* **2009**, *20*, 1554–1564. [CrossRef] [PubMed]
36. Kim, J.W.; Edemacu, K.; Kim, J.S.; Chung, Y.D.; Jang, B. A Survey Of differential privacy-based techniques and their applicability to location-Based services. *Comput. Secur.* **2021**, *111*, 102464. [CrossRef]