Towards End-to-End Secure and Efficient Federated Learning for XGBoost

Chao Jin[⋆], Jun Wang[‡], Sin G. Teo[⋆], Le Zhang[±], C.S. Chan[§], Qibin Hou[♯], Khin Mi Mi Aung[⋆]

[⋆]**Institute for Infocomm Research,** [‡]**OPPO Research Institute,**
[±]**University of Electronic Science and Technology of China,** [§]**University of Malaya,** [♯]**Nankai University**

[⋆]**{jin_chao, teosg, mi_mi_aung}@i2r.a-star.edu.sg**

Abstract

Federated learning refers to the distributed and privacy-preserving collaborative machine learning paradigm, in which multiple independent data owners jointly train on certain machine learning models without revealing their private data information to each other. In this paper, we study federated learning on XGBoost models in vertical data partition settings where the data owners share a common set of training samples, and each data owner possesses a disjoint subset of the features. We propose CryptoBoost, a federated XGBoost system based on multi-party homomorphic encryption techniques. CryptoBoost outperforms previous works in majorly three aspects. 1) CryptoBoost is end-to-end secure that the models are trained and stored in a completely encrypted and private manner. 2) CryptoBoost eliminates any central or privileged node that knows or controls more information than the other nodes, and the federated learning and inference processes are done in a fully decentralized way. 3) We propose a set of new secure computation algorithms and protocols for CryptoBoost, which achieve improved performance and communication efficiency compared with existing approaches.

1 Introduction

Due to privacy concerns and law regulations such as GDPR (Voigt and Von dem Bussche 2017), organizations are usually reluctant to disclose or share their data. However, data sharing and collaboration across multiple data owners could potentially bring large benefits to machine learning tasks. In order to solve this dilemma, Federated Learning (FL) (Kairouz et al. 2019) aims to enable multiple clients to jointly train or inference on certain models while protecting the private data of each client from being leaked to other clients. Apart from the distributed machine learning in nature, one of the major focused areas in FL is its privacy feature. Generally, FL can employ various techniques such as Multi-Party Computation (MPC), Homomorphic Encryption (HE), Differential Privacy (DP), and Trusted Execution Environment (TEE) to the distributed machine learning systems to protect data privacy for each of the participants.

The most common setups of FL are horizontal FL and vertical FL, categorized by how the data samples and sample features are distributed among the clients. For horizontal FL, each client holds a different set of samples, while each sample contains the same set of features. This is the most straightforward setup for FL, and the joint model is usually trained through secure aggregation such as FedAvg or FedSGD (McMahan et al. 2017) on the client models. On the other hand, for vertical FL all the clients share a common set of samples, but the samples in different clients contain different sets of features. Vertical FL is usually more complicated to implement than horizontal FL, but it has very broad application scenarios in the real world industry, such as the joint risk control for customers between banks and fintech companies, and joint recommendation systems among different retail and internet companies.

Most FL research efforts in the literature are focusing on horizontal data partition settings (Bonawitz et al. 2017; McMahan et al. 2018; Melis et al. 2019), while fewer are on vertical settings. However, existing vertical FL designs are usually insufficient in terms of privacy and efficiency. Some of them (Hu et al. 2019) assume the sample labels are shared across the all the clients, which may not be feasible in many practical scenarios. Some (Vaidya et al. 2013; Cheng et al. 2021) reveal certain intermediate data as plaintext, which may be exploited by a malicious client to infer the private information of other clients' data. Some (Wu et al. 2020; Cheng et al. 2021) assume there is a super client (usually the one holding the sample labels) who gathers all the necessary intermediate information in plaintext and orchestrates the FL process. Nevertheless, a malicious super client could exploit his privilege and break the data privacy of other clients.

In this work, we look into the privacy-preserving FL design for gradient tree boosting models, which have wide application areas like fraud detection, recommender systems, and online advertisements. In particular, we focus on XGBoost (Chen and Guestrin 2016) which provides an optimized design and implementation for gradient tree boosting models. We build CryptoBoost, a federated XGBoost system. CryptoBoost utilizes the Multi-Party Homomorhpic Encryption cryptography techniques for protecting each client's data privacy in the federate XGBoost training and inference processes. CryptoBoost is designed to be end-to-end secure that all the intermediate data and the models are kept in encrypted form during the entire FL process, minimizing the risk of data privacy leakage for each of the clients. Fur-

thermore, the CryptoBoost architecture is fully decentralized. Every client has a equal role in the system, and there is no so-called super client who collects more information and controls the entire FL process.

So far, most existing FL solutions are built for models like logistic regression and neural networks, and there are much fewer solutions for tree-based machine learning models like XGBoost. While the former mostly require just secure addition and multiplication primitives in the processes, the latter are more challenging as they require secure non-linear primitives like division and comparison, which are harder to implement and often deemed to be the performance bottleneck of the system. To solve the problem, we propose a novel design for these primitives that can securely evaluate division and comparison functions in a faster and more accurate way. Our design not only can be used to optimize federated XGBoost or tree-based models in general, but also can be beneficial to other applications that require such secure computations among multiple parties.

The rest of the paper is organized as follows. The next section introduces preliminaries and related work. Section 3 describes the detailed design of CryptoBoost. Section 3.4 analyzes the security of CryptoBoost. Section 4 evaluates the CryptoBoost performance and finally we conclude the paper in Section 5.

2 Preliminaries and Related Work

2.1 Multi-Party Computation

Multi-Party Computation (MPC) is the secure computation paradigm that multiple parties jointly evaluate a function, each with his private input that is not disclosed to others in the computation process. Depending on the number of parties, MPC can be divided into two-Party computation (2PC) and generic MPC with 3 or more parties. Yao's Garbled Circuit (Yao 1982) was the first secure 2PC solution, based on encrypted Boolean Circuits. GMW (Goldreich, Micali, and Wigderson 2019) and BGW (Ben-Or, Goldwasser, and Wigderson 2019) protocols were subsequently proposed for 2PC and also extendable to generic MPC. Generic MPC solutions normally rely on applying secret sharing such as additive sharing or Shamir's threshold sharing (Shamir 1979) to the input data. These secret sharing schemes support homomorphism on addition operations, meaning that adding two secretly shared data can simply be done by adding their corresponding shares respectively. However, multiplying two secretly shared data is more complicated, and the most widely implemented protocol is Beaver's Triple-based protocol (Beaver 1991). The main idea of Beaver's protocol is to divide the multiplication into an offline pre-computation phase and online computation phase. The offline phase is to generate random triples $(a, b, c = a \times b)$ from an appropriate field in the secretly shared form. Subsequently, each online multiplication operation will *consume* one such triple. Multiple different approaches can be used to generate Beaver's Triple in the offline phase, such as the Homomorphic Encryption based approach (Damgård et al. 2012), or Oblivious Transfer based approach (Keller, Orsini, and Scholl 2016).

2.2 Multi-Party Homomorphic Encryption

Before introducing Multi-Party Homomorphic Encryption (MHE), we first give a brief overview of the traditional single-key HE. HE is a special kind of encryption schemes that enables computation on encrypted data (ciphertexts) without requiring the decryption key. The computational result, after decryption, is equivalent to the result computed on the unencrypted counterpart through normal arithmetic operations. Depending on the homomorphic arithmetic operations supported, HE can be divided into Partial HE (PHE), which supports either homomorphic addition (Paillier 1999) or homomorphic multiplication (Rivest, Shamir, and Adleman 1978) between two ciphertexts, and Fully HE (FHE) (Gentry 2009), which supports both homomorphic addition and multiplication between the ciphertexts. The most common FHE schemes, including BGV (Brakerski, Gentry, and Vaikuntanathan 2014), BFV (Fan and Vercauteren 2012), and CKKS (Cheon et al. 2017), are based on the Ring Learning-with-Errors (RLWE) hard problem, which conceal plaintext messages with noises that can be identified and removed with the secret (decryption) key. Every time a ciphertext is computed and updated, the noise magnitude inside that ciphertext will grow. In order to prevent the noises from growing too big that may eventually corrupt the encrypted plaintext message, FHE also includes a primitive called bootstrapping, although could be slow in performance but can reduce the noises inside the ciphertexts accumulated along the computation, thus enables arbitrary computational depth on the ciphertexts in theory.

MHE extends the single-key HE to the multiple-party scenario. Generally, a MHE scheme can be built from the corresponding single-key HE scheme, by distributing its secret key among N parties according to certain secret sharing form. Subsequently, the operations in the original single-key scheme that depend on the secret key can be implemented through special-purpose secure multi-party computations for the multi-party scheme. In (Mouchet et al. 2021), a fully decentralized MHE scheme is proposed that does not require a trusted third-party to generate and distribute the secret key shares. Instead, each party samples his own share of the secret key in a non-interactive way. After the secret key is generated in the secret shared form, the parties can further execute the following protocols. The underlying secret key is not revealed to any party in these processes.

- **Encryption Key Generation.** All the parities work collectively to generate the common public key, using each of their secret key shares. The public key can be used by any party independently for encryption, similar like the single-key HE scheme.
- **Relinearizaiton Key Generation.** All the parties work collectively to generate the relinearization key used in the homomorphic multiplication operations. This enables the homomorphic multiplications can be evaluated by each party in a non-interactive way, similar like the single-key HE scheme.
- **Key Switching.** All the parties work collectively to switch the public or secret key associated with a ciphertext to another public or secret key. After key switching,

the ciphertext is equivalent to be encrypted under the new public key, or can be decrypted using the new secret key. Decryption of a ciphertext can be implemented by a special key switching operation, where the new secret key is zero.

- **Bridging between Encryption and Secret Sharing.** Based on encryption and collective decryption, MHE further provides two protocols to switch a message between encrypted form (i.e., ciphertext) and secret shared form among the parties. The combination of the protocols can also be utilized to implement a collective bootstrapping procedure that refreshes the noise inside a ciphertext.

2.3 Related Work

The literature has shown several research on applying privacy preserving FL to gradient tree boosting models. In (Li, Wen, and He 2020) and (Liu et al. 2020), the authors proposed privacy preserving FL on gradient boosting decision trees (GBDT) and XGBoost respectively. Both of them are under horizontal FL settings which essentially rely on secure aggregation of the sample gradients from different parties. The work more related to ours are federated GBDT or XGBoost under vertical FL settings, such as (Cheng et al. 2021; Le et al. 2021; Feng et al. 2019; Tian et al. 2020; Fang et al. 2021; Wu et al. 2020). (Cheng et al. 2021; Le et al. 2021; Feng et al. 2019) cannot achieve provable security as they reveal intermediate information in plaintext during the model training procedure, which may be exploited by a malicious client to infer private data information of other clients. (Tian et al. 2020) reveals the order instead of values of the features to determine the best split threshold, and adds hash buckets and differential privacy (DP) to mitigate the information leakage. Nevertheless, adding DP noises will trade off data utility for privacy. (Fang et al. 2021) provides an end-to-end encryption framework for federated XGBoost, but it depends heavily on secret sharing technique and incurs communication and computation inefficiency. (Wu et al. 2020) uses Threshold Paillier Homomorphic Encryption to build federated decision trees and GBDT models, aiming to improve efficiency by combining MPC and Paillier additive HE techniques. However, it fails to achieve end-to-end encryption and still shows computational inefficiency in ciphertext multiplications due to limitations of Paillier HE.

Our work employs Multi-Party FHE to build end-to-end secure federated XGBoost, and achieves more optimized system efficiency through new secure computation protocols. There are also some other work (Froelicher et al. 2021; Sav et al. 2021) that employs Multi-Party FHE to privacy preserving machine learning, but they are limited to logistic regression models and not applicable to tree-based models.

3 CryptoBoost

3.1 Vanilla XGBoost

As a boosting-based ensemble learning method, XGBoost trains an additive ensemble model $F(X)$ in a sequence of T iterations, and in the t-th iteration, a weaker model $f_t(X)$ is learnt and added to the ensemble model. The final stronger ensemble model $F_T(X) = \sum_{t=1}^{T} f_t(X)$ is the summation of all the weaker models. For a training set of n samples, let (X_i, y_i) be the feature set and ground-truth label of the i-th sample, and $\hat{y}_i = F(X_i)$ be the model prediction, the model is built to minimize the following objective loss function at t-th iteration.

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

In Equation 1, $\hat{y}_i^{(t-1)}$ is prediction of the ensemble model from the previous $t-1$ iterations, and $f_t(X_i)$ is prediction of the weaker model to be built from the current t-th iteration. Ω is the regularization term for model complexity related to the number and weight values of the leaf nodes. Next, a key idea of XGBoost is to approximate the objective function into the following Equation 2, based on Taylor's expansion.

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

In the above equation, $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$, which are the first and second order gradients of the loss function regarding the previous $t-1$ iterations. For a tree node with sample set I, let G and H be the sum of sample gradients in I, i.e., $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$. To determine the best split for a tree node, the following equation is evaluated for every possible split and the one with the highest gain is chosen.

$$Gain = \frac{1}{2}[\frac{(G^L)^2}{H_L + \lambda} + \frac{(G^R)^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}] - \gamma \quad (3)$$

G^R (resp. G^L) corresponds to the right (resp. left) child node after the split. λ and γ are the hyper parameters controlling the regularization terms. When a leaf node is reached, i.e., no more split will be made to the current node, the optimal weight of the node is computed as follows.

$$w = -\frac{G}{H + \lambda} \quad (4)$$

3.2 CryptoBoost Design

CryptoBoost is designed for federated learning of XGBoost models with vertically partitioned data. As show in Figure 1, each party holds a disjoint subset of sample features or the sample labels, and each tree model is trained as if all the feature subsets are combined together. However, due to data privacy enforcement, CryptoBoost cannot simply gather all the features and labels in a centralized place for the training. Instead, each party keeps his feature or label data in his local place, and jointly train the models by exchanging and computing on some encrypted intermediate data. The trained models are also in encrypted form and not disclosed to any party.

CryptoBoost employs MHE for encrypting the data and models during the federated training processes among the parties. The MHE common public key and evaluation keys are used by every party to encrypt and compute on the data

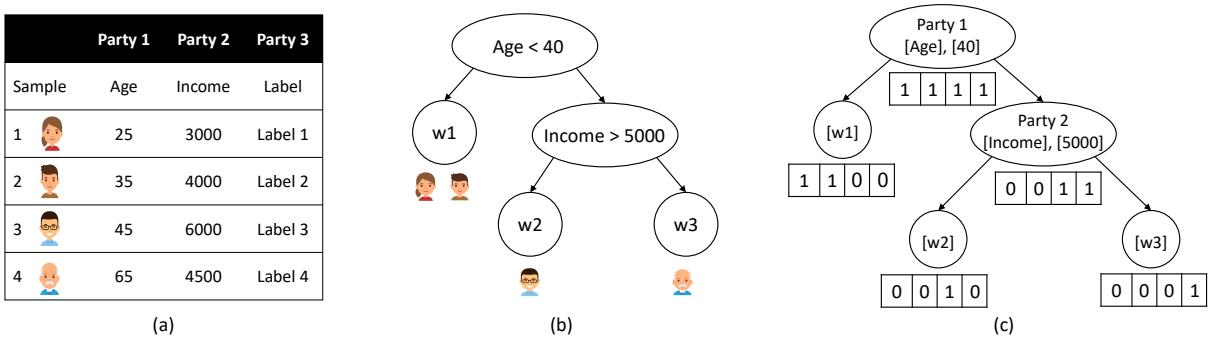

Figure 1: An example of (a) the vertically partitioned training dataset, (b) plaintext tree model and (c) federated tree model jointly trained among multiple parties. The federated tree model is end-to-end encrypted. Each internal tree node is *owned* by one party, from whom the feature and split value will be used to divide the samples. The feature and split value are only known to the party who owns the node. Each node is associated with an encrypted vector indicating which training samples are divided into this node, and the vector is unknown to all the parties. Finally, each leaf node is assigned with an encrypted weight that no party knows the value.

independently, while each party holds his share of the private key and all the parties need to collaborate to decrypt a ciphertext. Therefore, the MHE scheme ensures security of $N-1$ colluding parties among a total number of N parties. As with the analysis of vanilla XGBoost training, one essential step in building a tree model is to find the best feature and value for splitting each tree node. For CryptoBoost under the vertical FL setting, this has to be jointly done by all the parties through secure computations on the fully encrypted intermediate data and model.

The complete algorithm for training a regression tree model (i.e., weaker model) with CryptoBoost is shown in Algorithm 1. The whole XGBoost ensemble model with T weaker models can be trained through T such iterations. In the algorithm, encrypted items are marked with $[\,]$ brackets. As we can see, the entire model is kept in the encrypted form during the whole training process. In particular, each tree node is associated with an indicating vector IV, where only the indices of the samples divided to this node are ones and all the others are zeros. The indicating vectors are encrypted so no party knows the actual distribution of the training samples. For each internal node to be split, each party can propose all the possible divisions of the whole training set based on the features and feature values that he has. The party then creates two masking vectors for each division proposal and multiplies element-wise with the indicating vector of the node, which produces the candidate indicating vectors for the right and left child nodes after splitting. Then, the party computes the split gain based on the sample gradients in the encrypted form. The final split candidate is chosen based on the maximum split gain it brings among the proposals from all the parties. Again, the selection process is done through secure comparisons on the encrypted split gains across all the parties, and only the final result of the comparison, that is the index of the split candidate with the maximum gain, is revealed to the parties.

When a leaf node is reached, CryptoBoost computes the weight value for the node based on the gradients of the samples on that node. The computations and results of the weights are also in encrypted form, and not revealed to any party. After the weights for all the leaves are computed, the weaker tree model construction is completed, and it is added to the XGBoost ensemble model. Subsequently, CryptoBoost proceeds to update the predictions and gradients for all the training samples, which will be used for training the next weaker tree model in the next training iteration.

The prediction for a sample can be updated by adding the weight of the leaf node in which the sample resides. However, the distribution of the samples on the leaf nodes are not known, since the indicating vectors of the leaf nodes are encrypted. To overcome this problem, CryptoBoost creates for each sample an one-hot masking vector which contains all zeros expect an only one at the sample index, and compute the encrypted dot-product between the masking vector and the indicating vector of a leaf node. Obviously, the result of the dot-product is one only if the sample is on that leaf node, otherwise it will be zero. Then, the leaf weight for the sample can be computed by summing up the multiplication results between the leaf weights and the corresponding dot-product results. Finally, after the predictions for the samples are updated, their gradients can be securely updated based on their ground-truth labels and the loss function.

3.3 Secure Computation Algorithms

As shown in Algorithm 1, there are several functions in CryptoBoost training process that require secure computations on encrypted data. Among them, $HSum$, $HMult$, and $HDot$ (Homomorphic Dot-Product) basically involves homomorphic additions, subtractions and multiplications between cipehrtexts, which can be easily implemented with MHE primitives. However, functions such as $SecureComputeGain$, $SecureComputeWeight$ and $SecureComputeGradients$ involve secure divisions, and $SecureMax$ involves secure comparisons on encrypted data, which are generally harder to implement as MHE lacks support for such primitives. There have been some

Algorithm 1: CryptoBoost Training Algorithm for the t-th Weaker Tree Model

Input:

- n parties with m samples, where party P_i owns feature matrix X_i, and one party owns the labels y, for the training samples.
- The encrypted gradients of all the samples from previous ensemble model, $[g^{(t-1)}]$ and $[h^{(t-1)}]$.
- The encrypted predictions of all the samples from previous ensemble model, $[\hat{y}^{(t-1)}]$.
- The number of nodes in the tree, K.

Output:

- A tree model f_t to be added to the ensemble model.
- Updated predictions of all the samples for the next training iteration, $[\hat{y}^{(t)}]$.
- Updated gradients of all the samples for the next training iteration, $[g^{(t)}]$ and $[h^{(t)}]$.

f_t = InitTree(K)
$[IV_1] = \{1, ..., 1\}$ {Initialize the indicator vector of root node with all ones.}
$[G_1] = HSum([g^{(t-1)}])$
$[H_1] = HSum([h^{(t-1)}])$ {Compute the sum of gradients for all samples in root node.}
for $k = 1, 2, ..., K$ **do**
  if k is an internal node **then**
    Each party propose (feature, value) pairs and masking vectors MV^R and MV^L for child node splitting.
    Each party compute $[TmpIV^R] = HMult([IV_k], MV^R)$ and $[TmpIV^L] = HMult([IV_k], MV^L)$.
    Each party compute $[TmpG^R] = HDot([g^{(t-1)}], [TmpIV^R])$ and $[TmpG^L] = HDot([g^{(t-1)}], [TmpIV^L])$.
    Each party compute $[TmpH^R] = HDot([h^{(t-1)}], [TmpIV^R])$ and $[TmpH^L] = HDot([h^{(t-1)}], [TmpIV^L])$.
    Each party compute $[TmpGain] = SecureComputeGain([G_k], [H_k], [TmpG^R], [TmpG^L], [TmpH^R], [TmpH^L])$.
    All parties compute $index = SecureMax([TmpGain])$. {Finding index of splitting candidate with largest Gain.}
    $[IV_r] = [TmpIV^R_{index}]; [IV_l] = [TmpIV^L_{index}]$ {Update indicator vectors for child nodes.}
    $[G_r] = [TmpG^R_{index}]; [G_l] = [TmpG^L_{index}]$
    $[H_r] = [TmpH^R_{index}]; [H_l] = [TmpH^L_{index}]$ {Update sum of gradients for child nodes.}
  else
    $[w_k] = SecureComputeWeight([G_k], [H_k])$ {Compute the weights for leaf nodes.}
  end if
end for
for $i = 1, 2, ..., m$ **do**
  $OH_i = \{0, ..., 1, ..., 0\}$ {Create one-hot masking vector for each sample.}
  for all the leaf nodes S **do**
    $[\hat{y}_i^{(t)}] = [\hat{y}_i^{(t-1)}] + \sum_{s \in S} HMult([w_s], HDot([IV_s], OH_i))$ {Update prediction for each sample.}
  end for
  $[g_i^{(t)}], [h_i^{(t)}] = SecureComputeGradients([y_i], [\hat{y}_i^{(t)}])$ {Update gradients for each sample.}
end for

work proposed to implement divisions and comparisons with HE (Zhang et al. 2015; Yoo et al. 2019; Cheon, Kim, and Kim 2020), and some other work with MPC (Catrina and De Hoogh 2010; Catrina and Saxena 2010; Demmler, Schneider, and Zohner 2015), but they all require complicated and inefficient circuit design, which may suffer from performance or precision losses. To overcome these bottlenecks, we propose new algorithms to implement secure division, comparison and exponential functions under the MHE setting.

Secure Division Algorithm 2 shows our Secure Division protocol. The idea is for the parties to generate some random number in secret sharing mode, and then convert it into MHE ciphertext. Subsequently, the random ciphertext is homomorphically multiplied into the dividend and divisor ciphertexts respectively. After that, the randomized divisor is decrypted collaboratively by the parties and the division is finally done by multiplying the reciprocal of the divisor. In the process, the random factor as well as the original divisor are kept secret and not disclosed to any party. Note that, in XGBoost, the divisor is always non-zero for commonly adopted loss functions (e.g., square error or cross entropy).

Secure Exponent Algorithm 3 shows our secure protocol for computing the Nature Exponent of an encrypted exponent power. This function is often used in activation functions like $Sigmoid$ and $Softmax$. The protocol is similar like the Secure Division protocol, except that all the parties need to jointly generate a random number as well as the Nature Exponent of the number. In the subsequent process, only the difference of the random number and the input exponent power is decrypted. None of the random number, its Nature Exponent, or the input exponent power is disclosed to any party.

Algorithm 2: Secure Division with MHE

Input: MHE scheme among n parties, ciphertext $[c_1]$ encrypts the dividend, ciphertext $[c_2]$ encrypts the divisor.
Output: ciphertext $[c_3] = [\frac{c_1}{c_2}]$, i.e., the homomorphic division between c_1 and c_2.

1: For each party P_i, sample a random number r_i, and encrypt it $[r_i] = HEnc(r_i)$.
2: All parties add their random number together $[r] = \sum_{1 \le i \le n}[r_i]$.
3: $[c_x] = HMult([c_2], [r])$.
4: $c_x = HDec([c_x])$. {Collaborative decryption.}
5: $[c_y] = HMult([c_1], \frac{1}{c_x})$. {Scala multiplication.}
6: $[c_3] = HMult([c_y], [r])$.
7: **return** $[c_3]$.

Algorithm 3: Secure Exponent with MHE

Input: MHE scheme among n parties, ciphertext $[c_1]$ encrypts the power of the nature exponential function.
Output: ciphertext $[c_2] = [e^{c_1}]$, i.e., nature exponential base e to the power of c_1.

1: For each party P_i, sample a random number r_i, compute its nature exponent e^{r_i}, and encrypt them $[r_i] = HEnc(r_i), [e^{r_i}] = HEnc(e^{r_i})$.
2: All parties add their random number together $[r] = \sum_{1 \le i \le n}[r_i]$, and multiply their nature exponents together $[e^r] = \prod_{1 \le i \le n}[e^{r_i}]$.
3: $[r - c_1] = HSub([r], [c_1])$.
4: $r - c_1 = HDec([r - c_1])$. {Collaborative decryption.}
5: $[c_2] = HMult([e^r], e^{c_1 - r})$. {Scala multiplication.}
6: **return** $[c_2]$.

Secure Logarithm The Secure Logarithm protocol can be implemented in a very similar way like Secure Exponent. To compute the logarithm of a cipher value $[c]$, the parties first jointly generate a random number $[r]$ and its logarithm $[\log r]$, then they jointly decrypt the homomorphic product of $[r \times c]$ and compute its logarithm $\log(r \times c)$, and finally the logarithm of $[c]$ is computed by the homomorphic subtraction between $\log(r \times c)$ and $[\log r]$.

Secure Comparison Algorithm 4 shows our Secure Comparison protocol for two cipher values. In the start, the parties collaborate to generate two random numbers r and s in encrypted form, in which s is positive. Subsequently, both the cipher values are randomized by homomorphically multiplying with s and then homomorphically adding with r, and then decrypted jointly by the parties into plain values. It is obvious that the randomization process preserves the order of the cipher values, therefore they can be compared by directly comparing their corresponding plain values. Except for the comparison result, the Secure Comparison process does not reveal any additional information about the random numbers or input cipher values to any party.

Secure Sorting Secure Sorting refers to the task of securely sorting a vector of N cipher values where $N > 2$. To

Algorithm 4: Secure Comparison with MHE

Input: MHE scheme among n parties, two cipher values $[c_0]$ and $[c_1]$.
Output: 0 if $c_0 \ge c_1$, 1 otherwise.

1: For each party P_i, sample a random number r_i, a positive random number s_i, and encrypt them $[r_i] = HEnc(r_i), [s_i] = HEnc(s_i)$.
2: All parties add their random numbers together $[r] = \sum_{1 \le i \le n}[r_i], [s] = \sum_{1 \le i \le n}[s_i]$.
3: $[c'_0] = HAdd(HMult([c_0], [s]), [r])]$.
4: $[c'_1] = HAdd(HMult([c_1], [s]), [r])]$.
5: $c'_0 = HDec([c'_0])$.
6: $c'_1 = HDec([c'_1])$. {Collaborative decryption.}
7: $index = Max(c'_0, c'_1)$. {Compare in plaintext.}
8: **return** $index$.

Algorithm 5: Secure Sorting with MHE

Input: MHE scheme among n parties, a vector CV with N cipher values.
Output: Sorted CV with cipher values in ascending order.

1: For each party P_i, sample a random number r_i, positive random numbers s_i and t_i, and encrypt them $[r_i] = HEnc(r_i), [s_i] = HEnc(s_i), [t_i] = HEnc(t_i)$.
2: All parties add their random numbers together $[r] = \sum_{1 \le i \le n}[r_i], [s] = \sum_{1 \le i \le n}[s_i], [t] = \sum_{1 \le i \le n}[t_i]$.
3: **for** $i = 1, 2, ..., N$ **do**
4: $[CV'_i] = HComp([r] + [s] \times [CV_i] + [t] \times [CV_i]^3)$.
5: $PV_i = HDec(CV'_i)$. {Collaborative decryption.}
6: **end for**
7: $QuickSort(PV)$. {Sort in plaintext.}
8: Permutate CV based on the order of PV.
9: **return** CV.

implement secure sorting, it is straightforward to follow the traditional sorting algorithms in plaintext domain, and simply apply the Secure Comparison protocol between two cipher values. Obviously, the optimal complexity of the entire process requires $\mathcal{O}(N \times \log N)$ rounds of secure comparisons. To further optimize the efficiency of Secure Sorting, we propose a new protocol by extending the Secure Comparison protocol, as shown in Algorithm 5. The idea is for the parties to jointly map a ciphertext to a random degree-3 polynomial in the form of $[t] \times x^3 + [s] \times x + [r]$, hiding the true values as well as the possible correlation among the values. The r is any random number, t and s are positive random numbers to ensure the polynomial is monotonically-increasing. Then the polynomial is homomorphically evaluated with the cipher values respectively, and the evaluation results are decrypted and sorted in plaintext. The sorting result maintains the order of the original cipher values since the polynomial evaluation is monotonic (i.e., the correctness is guaranteed). It is worth noting that, we do not adopt the degree-1 polynomial like in the Secure Computation protocol, as it may disclose linear relationship among the cipher values. Furthermore, it is trivial to adjust the Secure Sorting

protocol to implement other functions like $SecureMax$.

All our secure computation protocols achieve lossless property, meaning the computation results have the same precision as their plaintext computation counterpart.

3.4 Threat Model

We consider the semi-honest adversary threat model. Each party will honestly follow the protocols, but may also be curious to infer other parties' private data based on what information he has. As described in Section 3, CryptoBoost achieves end-to-end encryption in the entire federated learning process. First, all the intermediate data derived from each party's local private data during training are encrypted as MHE ciphertexts, which include the first and second order gradients, the predictions of training samples, and the computed gains based on split proposals. Second, the trained models are encrypted as all the leaf weights are encrypted and the split feature and value for each internal node is known only to the party who *owns* the node. Third, the distribution of the training samples among the tree nodes are encrypted, as the indicator vectors of all the tree nodes are encrypted. In this way, the federated training process minimizes the risk of privacy leakage for each of the parties.

Thanks to the properties of MHE, CryptoBoost is also secure against $N-1$ colluding parties out of totally N parties. Since all the shared data and model information is encrypted under MHE, it would not be possible to decrypt the information without the agreement and collaboration of all the parties.

4 Performance Evaluation

We implement CryptoBoost based on the Palisade[1] MHE library, and deploy it on two Linux servers. Each of the servers has a 20-Core Intel i9-10900X CPU and 64 GB memory, and connected with 10 Gbps network. We evaluate the performance of CryptoBoost with the "Give Me Some Credit" dataset[2] published on Kaggle. The dataset contains 150000 data samples and 10 features per sample. The features and label of each sample are split onto the two CryptoBoost nodes. For the underlying XGBoost model, we set the number of tree estimators to be 3, and maximum depth for each tree is 3. As it is a binary classification task, Cross-Entropy Loss is utilized as the loss function.

We utilize the Multi-Party CKKS scheme for CryptoBoost, and test the performance of CryptoBoost under three sets of encryption parameters. The Ring Dimension size (a.k.a polynomial modulus degree) are set to be 8192, 16384, and 32768 respectively for the three sets. The CKKS scaling factor is set to be 2^{30} for providing 30 bits of precision during ciphertext computation. The ciphertext coefficient modulus Q is an important deciding factor for both the security level and the maximum multiplication depth of a ciphertext. A smaller Q translates to higher security level for the encryption scheme while a larger Q enables larger multiplication depth that a ciphertext can take. We set Q to be 210 bits in our experiments, since it not only ensures 128-bit Security Standard under all the three Ring Dimension sizes, but also accommodates large enough multiplication depths for our experiments.

[1]https://gitlab.com/palisade/palisade-release

[2]https://www.kaggle.com/c/GiveMeSomeCredit

Ring Dimension	Security (bits)	Batching Ciphertexts	Training Latency (hours)
8192	> 128	37	26.1
16384	> 128	19	29.5
32768	> 128	10	33.6

Table 1: CryptoBoost training time for building one tree estimator under different encryption parameters.

CryptoBoost achieves 93.48% accuracy on the training dataset, which has negligible difference compared to the vanilla XGBoost. Table 1 shows the CryptoBoost training performance under different encryption parameter settings. Pay attention to the fact that, CryptoBoost utilizes the powerful FHE packing feature, in which a ciphertext under Ring Dimension N can be encrypted and computed with $\frac{N}{2}$ values simultaneously. Therefore, to deal with a total number of 150000 sample data, it only needs a batch of 10, 19, and 37 ciphertexts respectively under the three Ring Dimension sizes. On the other hand, it is worth noting that, a larger Ring Dimension, although provides higher packing density and higher security level, leads to larger execution time because the ciphertext computation complexity increasing rapidly as Ring Dimension increases.

Our current CryptoBoost implementation experiences longer training latency compared to the non-secure counterparts. This is largely due to the end-to-end encryption design where all the operations are conducted on the ciphertexts. However, there are still many ways to improve the performance of CryptoBoost. For example, we can utilize the distributed bootstrapping of MHE during ciphertext computation, and this enables us to select smaller security parameters for more efficient computation. On the other hand, deploying CryptoBoost onto more client nodes will also largely improve its performance due to parallel computations. As the next step, we aim to improve CryptoBoost performance through all possible optimizations.

5 Conclusion

In this paper, we propose CryptoBoost, a federated XGBoost system with vertically partitioned data among multiple parties. CryptoBoost is ultra secure with end-to-end encryption on the data and model during the federated learning processes. It also achieves fully decentralized architecture as all the parties behave equally without any superior party who collects or controls more information. At last, we propose a set of new MHE-based secure computation protocols to enable more efficient and accurate ciphertext computations for various arithmetic operations in CryptoBoost, and these secure computation building blocks can also be utilized for other secure applications among multiple parties.

6 Acknowledgments

This research / project is supported by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Programme (Award A19E3b0099).

References

Beaver, D. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, 420–432. Springer.

Ben-Or, M.; Goldwasser, S.; and Wigderson, A. 2019. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 351–371.

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

Brakerski, Z.; Gentry, C.; and Vaikuntanathan, V. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3): 1–36.

Catrina, O.; and De Hoogh, S. 2010. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, 182–199. Springer.

Catrina, O.; and Saxena, A. 2010. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, 35–50. Springer.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Cheng, K.; Fan, T.; Jin, Y.; Liu, Y.; Chen, T.; Papadopoulos, D.; and Yang, Q. 2021. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*.

Cheon, J. H.; Kim, A.; Kim, M.; and Song, Y. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, 409–437. Springer.

Cheon, J. H.; Kim, D.; and Kim, D. 2020. Efficient homomorphic comparison methods with optimal complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, 221–256. Springer.

Damgård, I.; Pastro, V.; Smart, N.; and Zakarias, S. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, 643–662. Springer.

Demmler, D.; Schneider, T.; and Zohner, M. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation. In *NDSS*.

Fan, J.; and Vercauteren, F. 2012. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012: 144.

Fang, W.; Zhao, D.; Tan, J.; Chen, C.; Yu, C.; Wang, L.; Wang, L.; Zhou, J.; and Zhang, B. 2021. Large-Scale Secure XGB for Vertical Federated Learning. In *Proceedings of 30th ACM International Conference on Information and Knowledge Management (CIKM2021)*.

Feng, Z.; Xiong, H.; Song, C.; Yang, S.; Zhao, B.; Wang, L.; Chen, Z.; Yang, S.; Liu, L.; and Huan, J. 2019. Securegbm: Secure multi-party gradient boosting. In *2019 IEEE International Conference on Big Data (Big Data)*, 1312–1321. IEEE.

Froelicher, D.; Troncoso-Pastoriza, J. R.; Pyrgelis, A.; Sav, S.; Sousa, J. S.; Bossuat, J.-P.; and Hubaux, J.-P. 2021. Scalable Privacy-Preserving Distributed Learning. *Proceedings on Privacy Enhancing Technologies*, 2021: 323 – 347.

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 169–178.

Goldreich, O.; Micali, S.; and Wigderson, A. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 307–328.

Hu, Y.; Niu, D.; Yang, J.; and Zhou, S. 2019. FDML: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2232–2240.

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

Keller, M.; Orsini, E.; and Scholl, P. 2016. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 830–842.

Le, N. K.; Liu, Y.; Nguyen, Q. M.; Liu, Q.; Liu, F.; Cai, Q.; and Hirche, S. 2021. FedXGBoost: Privacy-Preserving XGBoost for Federated Learning. *arXiv preprint arXiv:2106.10662*.

Li, Q.; Wen, Z.; and He, B. 2020. Practical federated gradient boosting decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4642–4649.

Liu, Y.; Ma, Z.; Liu, X.; Ma, S.; Nepal, S.; Deng, R. H.; and Ren, K. 2020. Boosting Privately: Federated Extreme Gradient Boosting for Mobile Crowdsensing. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 1273–1282. PMLR.

McMahan, H. B.; Ramage, D.; Talwar, K.; and Zhang, L. 2018. Learning Differentially Private Recurrent Language Models. In *ICLR*.

Melis, L.; Song, C.; De Cristofaro, E.; and Shmatikov, V. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, 691–706. IEEE.

Mouchet, C.; Troncoso-Pastoriza, J.; Bossuat, J.-P.; and Hubaux, J.-P. 2021. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4): 291–311.

Paillier, P. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, 223–238. Springer.

Rivest, R. L.; Shamir, A.; and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2): 120–126.

Sav, S.; Pyrgelis, A.; Troncoso-Pastoriza, J. R.; Froelicher, D.; Bossuat, J.; Sousa, J. S.; and Hubaux, J. 2021. POSEIDON: Privacy-Preserving Federated Neural Network Learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*.

Shamir, A. 1979. How to share a secret. *Communications of the ACM*, 22(11): 612–613.

Tian, Z.; Zhang, R.; Hou, X.; Liu, J.; and Ren, K. 2020. Federboost: Private federated learning for gbdt. *arXiv preprint arXiv:2011.02796*.

Vaidya, J.; Shafiq, B.; Fan, W.; Mehmood, D.; and Lorenzi, D. 2013. A random decision tree framework for privacy-preserving data mining. *IEEE transactions on dependable and secure computing*, 11(5): 399–411.

Voigt, P.; and Von dem Bussche, A. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10: 3152676.

Wu, Y.; Cai, S.; Xiao, X.; Chen, G.; and Ooi, B. C. 2020. Privacy preserving vertical federated learning for tree-based models. In *Proceedings of the VLDB Endowment*.

Yao, A. C. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, 160–164. IEEE.

Yoo, J. S.; Hwang, J. H.; Song, B. K.; and Yoon, J. W. 2019. A bitwise logistic regression using binary approximation and real number division in homomorphic encryption scheme. In *International Conference on Information Security Practice and Experience*, 20–40. Springer.

Zhang, Y.; Dai, W.; Jiang, X.; Xiong, H.; and Wang, S. 2015. Foresee: Fully outsourced secure genome study based on homomorphic encryption. *BMC medical informatics and decision making*, 15(5): 1–11.