Secure Two-Party Decision Tree Classification Based on Function Secret Sharing

Kun Liu and Chunming Tang

School of Mathematics and Information Sciences, Guangzhou University, Guangzhou 510000, China

Correspondence should be addressed to Chunming Tang; ctang@gzhu.edu.cn

Received 13 April 2023; Revised 19 September 2023; Accepted 26 September 2023; Published 31 October 2023

Academic Editor: Roberto Natella

Copyright © 2023 Kun Liu and Chunming Tang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Decision tree models are widely used for classification tasks in data mining. However, privacy becomes a significant concern when training data contain sensitive information from different parties. This paper proposes a novel framework for secure two-party decision tree classification that enables collaborative training and evaluation without leaking sensitive data. The critical techniques employed include homomorphic encryption, function secret sharing (FSS), and a custom secure comparison protocol. Homomorphic encryption allows computations on ciphertexts, enabling parties to evaluate an encrypted decision tree model jointly. FSS splits functions into secret shares to hide sensitive intermediate values. The comparison protocol leverages FSS to securely compare attribute values to node thresholds for tree traversal, reducing overhead through efficient cryptographic techniques. Our framework divides computation between two servers holding private data. A privacy-preserving protocol lets them jointly construct a decision tree classifier without revealing their respective inputs. The servers encrypt their data and exchange function secret shares to traverse the tree and obtain the classification result. Rigorous security proofs demonstrate that the protocol protects data confidentiality in a semi-honest model. Experiments on benchmark datasets confirm that the approach achieves high accuracy with reasonable computation and communication costs. The techniques minimize accuracy loss and latency compared to prior protocols. Overall, the paper delivers an efficient, modular framework for practical two-party secure decision tree evaluation that advances the capability of privacy-preserving machine learning.

1. Introduction

The two stages of a machine learning process are as follows. A model or classifier is developed using a potentially vast collection of training data during the first phase, also known as the learning phase. Then, the raw data are classified using the model. In many fields, including healthcare, finance, spam filtering, intrusion detection, and remote diagnostics, machine learning (ML) classifiers are useful tools [1]. These classifiers frequently need access to highly sensitive personal information like medical or financial records to execute their duties. Investigating systems that guarantee data privacy while reaping the rewards of ML is, therefore, essential. On the one hand, the ML model itself could include private information. For instance, a bank that utilizes a decision tree to evaluate its credit by clients would wish to keep the information about the model private. On the other hand, the model may have been created using private information. So-called model inversion attacks are widely known. Furthermore, these attacks might jeopardize the confidentiality of the training data, which are promoted by white-box and, even worse, black-box access to ML models [2–4]. Therefore, publicizing the ML model can conflict with the training data privacy.

Private decision tree evaluation can be implemented using general secure two-party computation [5–7] techniques like secret sharing and garbled circuits. The goal is to protect the decision tree algorithm so that it may be reviewed without disclosing any personal information. Some frameworks like ObliVM [8] and CBMC-GC [9] can transform plaintext programs written in high-level programming languages into oblivious programs suitable for secure
computing. Their straightforward application to decision tree algorithms unquestionably improves performance compared to a manually created architecture. Nonetheless, the size of the resulting ignorant program is still proportional to the size of the tree. Generic methods are, therefore, typically useless, especially when the tree is large.

2. Overview of Our Construction

We make use of homomorphic encryption and function secret sharing to implement a server-server secure two-party decision tree classification evaluation protocol without the trusted third party.

The basic idea is as follows. (1) Without relying on a trusted third party, two different servers share their data sources and combine their common data to perform security decision tree training on the ciphertext. (2) In this study, we address the problem of private decision tree training on confidential data from different data sources. The characteristics of the data of the two servers are public, and the server requires to combine the data of the other party to classify the private attribute vector. The goal of the computation is to determine the classification while keeping the user input and the decision tree confidential. Once the calculations are complete, only the classification models and their respective training results are shared in secret; neither party knows anything else. Use any general secure multiparty computation to solve the problem. There are specialized solutions that integrate multiple methodologies and leverage subject matter expertise to create effective agreements.

In this study, we provide a 2PC-based (two-party computation) framework for decision tree training and inference that is quicker and more precise. We provide several new building blocks based on the comparison protocol [10–19], support and implement secret sharing comparison on 2PC, and establish a new preprocessing protocol for mask creation. The experimental findings demonstrate that our approach is more accurate and time-effective than the majority of existing frameworks.

It is more challenging to prevent collusion among participants when several parties are involved and there are issues with the deployment itself. Although the present 3PC (three-party computation) or multi-PC security architecture must ensure an honest majority, the real world makes it difficult to meet this condition. The cooperation of parties can only be easily regulated if it is implemented on cloud servers owned by many businesses. Nevertheless, 2PC can fulfill this need. Our intent is to deliver and implement a unique two-server protocol that gives both parties access to a complete classification model while maintaining the privacy of their own data and a tolerable level of speed. The plan is to evaluate ciphertext trees encrypted with a server public key while using fully or somewhat homomorphic encryption (FHE/SHE). Therefore, the evaluation server is not informed of any intermediate or final calculation results. Existing fully homomorphic encryption techniques have high computational overhead and data transmission costs. To address this, we use efficient data representation and algorithm improvements. However, fully homomorphic encryption still has substantial overhead compared to our approach.

We summarize the key differences between our method and prior work by De Cock et al. [20] and Lu et al. [21] in Table 1. In this work, we have introduced a novel framework for secure two-party decision tree classification that provides substantial improvements over prior art. As evidenced by the table, our approach achieves higher accuracy, lower communication overhead, and reasonable computation complexity compared to De Cock et al. [20] and Lu et al. [21]. Our innovations in computing decision bits and combining secret sharing with homomorphic encryption lead to a highly performant and accurate framework with demonstrable gains. The empirical results substantiate the concrete efficiency and accuracy advantages of our proposed techniques over existing methods.

In this paper, we present secure two-party decision tree classification for different data sources training and inference. The two-party setting is reasonable for real-world applications [22] and has been widely employed in privacy-preserving machine learning [23–26]. First, exploiting an advanced cryptographic primitive, function secret sharing (FSS) [27], we present an efficient comparison protocol for the choice of the best split. The main challenge is that directly using the general FSS scheme [28] leads to a high evaluation overhead, since it requires two FSS invocations to handle the wrap around problem illustrated in Section 3. We address this by providing a novel theoretical analysis, which shows that the probability of incurring the wrap around problem is negligible with appropriate parameter settings even though we only invoke one FSS evaluation. This achieves approximately 2× reduction in the online runtime compared to the most efficient FSS scheme [28], while resulting in a slight accuracy loss in the training of trees. For communication, our protocol only requires one communication round with 2 ring elements. Nonetheless, the computational workload can be parallelized, thereby reducing computational time, even though the computational overhead may still end up being larger than existing protocols. By providing encrypted input and returning only encrypted output, we are able to provide a noninteractive protocol that enables clients to outsource evaluation to servers. Furthermore, it is possible to make existing systems unilaterally simulatable and secure in a semihonest model by employing techniques that may double computation and communication costs.

3. Related Work

The scope of our research pertains to private function evaluation (PFE) [29, 30], specifically, privacy-preserving decision tree evaluation [10–16] as a component of secure multiparty computation [5–7, 31–35]. In this section, we will give a brief evaluation, while a more comprehensive analysis can be found in the literature.

Brickell et al. [12] introduced the first private decision tree evaluation protocol by utilizing a novel combination of homomorphic encryption (HE) and garbled circuits (GC). The server translates the decision tree into a GC, which the
De Cock et al.’s approach is secure in the computational environment. Lu et al. [21] have proposed XCMP, a noninteractive comparison protocol using BGV homomorphic method [38] with the polynomial encoding of the inputs. They have further employed output expressive XCMP to construct the private decision tree protocol suggested by Tai et al., thereby maintaining additive homomorphism.

The decision tree technique is efficient and noninteractive due to its short multiplicative depth. However, it has limitations and is not universal as it works best with small inputs and depends on BGV-type homomorphic encryption (HE) schemes. Furthermore, it lacks output expressiveness like XCMP. It cannot support SIMD operations, which makes it unsuitable for expanding to more complex protocols such as random forest [39] while preserving its noninteractive nature. Additionally, the output length of the technique is exponential in the depth of the tree. In contrast, our binary instantiation has a marginally linear output length, and the integer instantiation can further reduce it by utilizing SIMD.

Like most privacy-preserving techniques based on FSS [28, 40–42], their schemes derive correlated randomness through a third party. However, the role of the third party can be jointly simulated by the two parties using either generic two-party secure protocols such as garbled circuits (GCs) [43] and GMW [44], or specific techniques [45]. Specifically, (1) one can use generic GCs or GMW style protocols to produce the required correlated randomness during the offline phase. Although versatile, these protocols necessitate a private evaluation of underlying pseudorandom generators (PRGs) during the FSS key generation phase. (2) In a customized approach, Doerner and Shelat [45] proposed a new solution that offers significant efficiency advantages, as PRG evaluation takes place locally without the need for secure simulation. However, it is suitable only for moderate domain sizes and challenging to extend to more generalized and significant cases.

### 4. Preliminaries

This section provides essential definitions and notations for our system, serving as a background for the rest of the study. Fully or somewhat homomorphic encryption is the fundamental concept, wherein we have simplified the mathematical intricacies to facilitate the reader’s comprehension and presentation. In this work, we utilize the terminology presented in [46] to delineate several foundational concepts. Relevant literature [12, 46–52] regarding homomorphic encryption is recommended for further understanding.
5. Decision Tree Classifier

Machine learning relies heavily on decision trees for data classification and regression. This study considers two parties that provide their distinct input variables and independently reconstruct the tree model, with data from one party kept confidential from the other. The decision tree classification’s primary objective, given an input query \( x \), is to follow the tree model and compare the input entries to node-specific thresholds for each decision node. The left or right child node is chosen as the next node depending on the comparison result. The classification model eventually ends in a specific leaf node, giving the input query a unique classification label.

Let elements \( x \in \mathbb{Z}^k \) be a feature vector. A function \( \mathcal{T}(x) \) with a \( k \)-dimensional feature space is implemented by a decision tree \( \mathcal{T} : \mathbb{Z}^k \rightarrow \mathbb{Z} \). Let the input query \( q = (q_1, \ldots, q_n) \in \mathbb{Z}^k \) of the party \( i \) be an \( n_i \)-dimensional positive integer vector over \( \mathbb{Z} \). The Boolean function \( \text{Bool}(x) = 1 \{ x_i < t_i \} \) is connected to each child node \( v_i \) in the tree, where \( i \in [n] \) is indeed the index of the feature vector \( x \in \mathbb{Z} \) and \( t_k \) is the threshold.

Then the decision tree evaluation on input \( x = (x_1, \ldots, x_{n-1}) \) is given by \( \ell_{\text{label}} = \mathcal{T}(x) \) with \( \mathcal{T} : \mathbb{Z}^k \rightarrow \ell_{\text{label}}^m \), in which \( m \) is the number of leaf nodes. This function starts from the root node and then does a comparison at each decision node. Let \( j \) be the index of a decision node and \( f \) be the function mapping the decision node index \( j \) to the corresponding input index \( f(j) \). Besides, let \( t_j \) be the threshold value of decision node \( j \). Then if \( x_{f(j)} > t_j \) holds for node \( j \), the right child is chosen as the next decision node; otherwise, the left child node is chosen. At the end, the function outputs the classification label \( \ell_{\text{label}} \) of the final leaf node.

A decision tree (DT) is a function \( \mathcal{T} : \mathbb{Z} \rightarrow \{ c_0, \ldots, c_{k-1} \} \) that maps an attribute vector \( x = (x_0, \ldots, x_{m-1}) \) to a finite set of classification labels. The tree consists of

(i) Nodes that either contain a test condition or are internal decision nodes.
(ii) Nodes that contain a classification label and are considered as leaf nodes.

The decision tree model comprises a decision tree and the functions outlined below:

(i) A threshold value is assigned to each decision node by the function \( \text{thr} : [0, m-1] \rightarrow \mathbb{Z} \).
(ii) An attribute index is assigned to each decision node by the function \( \text{att} : [0, m-1] \rightarrow [0, n-1] \).
(iii) A label of each leaf node is assigned to each decision node by a labeling function \( \text{lab} : [m, M-1] \rightarrow \{ c_0, \ldots, c_{k-1} \} \).

At each decision node, a comparison of “greater-than” is made between the assigned threshold and attribute values, i.e., the decision at node \( v \) is \( x_{\text{att}(v)} \geq \text{thr}(v) \).

Node Indices. If we have a decision tree, the index of a node can be determined using breadth-first search (BFS) traversal, starting at the root with index 0. When the tree is complete, a node with index \( v \) will have a left child of \( 2v + 1 \) and a right child of \( 2v + 2 \).

6. Homomorphic Encryption

This article focuses on lattice-based homomorphic encryption methods that allow for computations on ciphertexts by generating an encrypted output that corresponds to the result of a function applied to the plaintexts. Such encryption schemes facilitate several linked additions and multiplications on plaintexts in a homomorphic manner.

**Definition 1.** Consider the plaintext space defined as a ring \( \mathbb{Z}_q[X]/(X^N + 1) \), where \( q \) is a prime number and \( N \) can be expressed as a power of two. The homomorphic encryption (HE) scheme under consideration includes the following algorithms:

(i) \( (pk, sk, ek) \leftarrow \text{KGen}(\lambda) \): The generation of private key \( sk \), public key \( pk \), and evaluation key \( ek \) is achieved through a probabilistic algorithm denoted by \( \text{KGen}(\lambda) \). This algorithm employs a security parameter \( \lambda \) to ensure the randomness and security of the generated keys.

(ii) \( c \leftarrow \text{Enc}(pk, m) \): An encryption algorithm using a probabilistic algorithm is employed to produce a ciphertext \( c \) from a given message \( m \) and public key \( pk \). We will denote the resulting encryption as \( [m] \).

(iii) \( c \leftarrow \text{Eval}(ek, f, c_1, \ldots, c_n) \): a probabilistic algorithm is utilized to generate a ciphertext \( c \) by employing the evaluation key \( ek \), an \( n \)-ary function \( f \), and \( n \) ciphertexts denoted as \( c_1, \ldots, c_n \).

(iv) \( m \leftarrow \text{Dec}(sk, c) \): a message \( m \) can be generated from a given ciphertext \( c \) and private key \( sk \) using a deterministic algorithm.

When using the encoding method in homomorphic encryption (HE), the ciphertext is modified by introducing “noise,” which can increase during homomorphic evaluation. While the noise level grows exponentially upon multiplication, adding ciphertexts results in a linear increase. If the noise level becomes too high, it makes the decryption of the ciphertext impossible. To avoid this problem, either the refresh algorithm can be employed or the depth of the circuit for the function \( f \) can be kept sufficiently low. These techniques include key-switching or bootstrapping procedures that convert a ciphertext encrypted with one key into a ciphertext of the same message encrypted with another key and a specified amount of noise [46].

7. Function Secret Sharing

Function secret sharing (FSS) works by splitting a function \( f \) into two succinct function parts such that each part reveals nothing about the function \( f \), but when the evaluations are combined at some point \( x \), the result is \( f(x) \).
Formally, an FSS scheme is a pair of algorithms \( \text{Gen} \) and \( \text{Eval} \) with the following syntax. We identify two FSS constructions [28, 40] as a natural fit for our scheme: (1) distributed point function (DPF) \((\text{Gen}^{\text{DPF}}, \text{Eval}^{\text{DPF}})\) that satisfies \( f_{a,b}(x) = b \) if \( x = a \) and 0 otherwise and (2) distributed comparison function (DCF) \((\text{Gen}^{\text{DCF}}, \text{Eval}^{\text{DCF}})\) that satisfies \( f_{a,b} = b \) if \( x < a \) and 0 otherwise.

Definition 2 (function secret sharing [27, 53]). A two-party function secret sharing (FSS) scheme is a pair of algorithms \((\text{Gen}, \text{Eval})\) such that

1. \( \text{Gen}(1^k, f) \) is a probabilistic polynomial-time (PPT) key generation algorithm that given secure parameter \( 1^k \) and a function \( f \in \{0, 1\}^* \), outputs a pair of keys \( (k_0, \ldots, k_n) \). We assume that \( f \) explicitly contains descriptions of input and output groups \( G_{in}, G_{out} \).
2. \( \text{Eval}(\sigma, k_s, x) \) is a polynomial-time evaluation algorithm that given \( \sigma \in \{0, \ldots, m\} \) (party index), \( k_s \) is defined as the key of function \( f_{\sigma} \): \( G_{in} \leftarrow G_{out} \). Let \( x \in G_{in} \) be the input of function \( f_{\sigma} \) and output a group element \( y_{\sigma} \in G_{out} \).

When \( \sigma \) is omitted, it is understood to be 2. When \( \sigma = 2 \), we sometimes index the parties by \( \sigma \in \{0, 1\} \) rather than \( i \in \{1, 2\} \).

Definition 3 (correctness and security [27, 53]). Let \( \mathcal{F} = \{ f \} \) be a function family and Leak be a function specifying the allowable leakage about \( f \). When Leak is omitted, it is understood to output only \( G_{in}, G_{out} \). We say that \( \text{Gen}(\text{Eval}) \) as in Definition 2 is an FSS scheme for \( \mathcal{F} \) (with respect to leakage Leak) if it satisfies the following requirements.

3. Correctness: for all \( f: G_{in} \leftarrow G_{out} \) and every \( x \in G_{in} \), if \( (k_0, k_1) \rightarrow \text{Gen}(1^k, f) \), then \( \Pr[\text{Eval}(0, k_0, x) + \text{Eval}(1, k_1, x) = f(x)] = 1 \).
4. Security: for each \( \sigma \in \{0, 1\} \) there is a PPT algorithm \( \text{Sim}_{\sigma} \) (simulator), such that for every sequence \( (\tilde{f}_1)_{\ell \in \lambda} \) of polynomial-size function descriptions from \( \mathcal{F} \) and polynomial-size input sequence \( x_\lambda \) for \( f_\lambda \), the outputs of the following experiments Real and Ideal are computationally indistinguishable:
   (i) Real\(_1\): \( (k_0, k_1) \rightarrow \text{Gen}(1^k, \tilde{f}_1) \); output \( k_\sigma \).
   (ii) Ideal\(_1\): Output \( \text{Sim}_{\sigma}(1^k, \text{Leak}(\tilde{f}_1)) \).

A central building block for many of our constructions is an FSS scheme for a special interval function referred to as a distributed comparison function (DCF) as defined below. We formalize it below.

Definition 4 (distributed comparison function). A special interval function \( f_{\alpha,\beta} \) also referred to as a comparison function, outputs \( \beta \) if \( x < \alpha \) and 0 otherwise. We refer to an FSS scheme for comparison functions as DCF. Analogously, function \( f_{\alpha,\beta} \) outputs \( \beta \) if \( x \leq \alpha \) and 0 otherwise. In all of these cases, we allow the default leakage \( \text{Leak}(\tilde{f}) = (G_{in}, G_{out}) \).

Theorem 5 (concrete cost of DCF). Given a PRG \( \{0, 1\}^n \rightarrow \{0, 1\}^{2n + 2} \), there exists a a DCF for \( f_{\alpha,\beta} \): \( G_{in} \leftarrow G_{out} \) with key size \( 4n \cdot (\lambda + 1) + n\ell + \lambda \), where \( n = \lceil \log(G_{\text{out}}) \rceil \) and \( \ell = \lceil \log(G_{\text{out}}) \rceil \). For \( \ell = \lceil \ell/\lambda + 2 \rceil \), the key generation algorithm \( \text{Gen} \) invokes \( G \) at most \( n \cdot (4 + \ell) \) times and the algorithm \( \text{Eval} \) invokes \( G \) at most \( n \cdot (2 + \ell) \) times.

We use \( \text{DCF}_{\alpha,\beta} \) to denote the total key size, i.e., \( |k_0| + |k_1| \), of the DCF key with input length \( n \) and output group \( G \). On the other hand, we use \( \text{DCF}_{\alpha,\beta} \) (nonbold) to denote the key size per party, i.e., \( |k_0|, |k_1| \in \{0, 1\} \). This captures the key size used in Eval algorithm. In the rest of the paper, we use \( \text{DCF}_{\alpha,\beta} \) to count number of invocations/evaluations as well as key size per evaluator \( P_a, b \in \{0, 1\} \).

8. Our Construction

This section outlines a modular description of our base protocol for secure two-party decision tree classification. We first introduce the data structures used in the protocol. By employing this structured representation of data, we can ensure that each party has access to necessary information while preserving the privacy of sensitive data. This enhances the security of our protocol, making it suitable for real-world applications requiring secure data analysis.

At last, we show the honest-but-curious adversarial model assumed in our protocol and cryptographic primitives like function secret sharing, homomorphic encryption, and secure comparison to prevent leakage of sensitive data to each party for our protocol. Overall, the modular design of our base protocol enables us to address specific security concerns by considering data structures and access control mechanisms. In subsequent sections, we describe the key components of the protocol in more detail, including the cryptographic primitives employed and the communication protocol used to facilitate secure multiparty computation.

9. Data Structure

Definition 6. For a decision tree model \( M = (\mathcal{F}, \text{thr}, \text{att}) \), we denote the tree \( \mathcal{T} \) of each node \( v \) which consists of

(i) \( v.\text{threshold} \): the threshold of node \( v \), denoted by \( \text{thr}(v) \), is stored in the variable \( v.\text{threshold} \).
(ii) \( v.\text{aIndex} \): the associated index, denoted by \( \text{att}(v) \), is stored in the variable \( v.\text{aIndex} \).
(iii) \( v.\text{parent} \): A pointer to the parent node is stored in the variable \( v.\text{parent} \). For the root node, this pointer is null.
(iv) \( v.\text{left} \): Pointers to the left child nodes are stored in the variables \( v.\text{left} \). For leaf nodes, these pointers are null.
(v) \( v.\text{right} \): Pointers to the right child nodes are stored in the variables \( v.\text{right} \). For leaf nodes, these pointers are null.
(vi) \( v.\text{cmp} \): During tree evaluation, the comparison bit \( b \leftarrow [\text{att}(v) < \text{thr}(v)] \) is computed and
stored in the variable v.cmp. If v is a right node, it stores b; otherwise, it stores 1 − b.

(vii) v.cLabel: the classification label is stored in the variable v.cLabel if v is a leaf node; otherwise, it stores an empty string.

Definition 7 (classification function). Let the attribute vector be \( x = (x_0, \ldots, x_{n-1}) \) and the decision tree model be \( \mathcal{M} = (\mathcal{D}, \mathcal{L}) \). We define the classification function to be \( f_x(x, \mathcal{M}) = \text{tr}(x, \text{root}) \), where root is the root node and \( \text{tr} \) is the traverse function defined as

\[
\text{tr}(x, v) = \begin{cases} 
\text{tr}(x, v.\text{left}), & \text{if } v \in \mathcal{D} \text{ and } x.v.\text{index} < v.\text{threshold}, \\
\text{tr}(x, v.\text{right}), & \text{if } v \in \mathcal{D} \text{ and } x.v.\text{index} \geq v.\text{threshold}, \\
v, & \text{if } v \in \mathcal{L}.
\end{cases}
\] (1)

10. Building Blocks

A one-time key is generated as part of the initialization process for a homomorphic encryption system. The server \( S_0 \) is responsible for creating the triple \((\text{pk}, \text{sk}, \text{ek})\), which consists of the public, private, and evaluation keys. Following this, \( S_i (i \in \{0,1\}) \) sends \((\text{pk}, \text{ek})\) to the other server \( S_{1-i} \). For each instance of data categorization, \( S_i \) encrypts their input and forwards it to the server \( S_{1-i} \). A trusted randomizer can be employed to reduce transmission costs, which is not authorized to cooperate with the server and does not participate in the actual protocol. This technique is similar to commodity-based cryptography, except that the client can act as the randomizer themselves and provide the list of \( \lfloor r \rfloor \) before the start of the protocol when the network is not overloaded.

The server starts by computing for each node \( v \in \mathcal{D} \) the comparison bit \( b \leftarrow [x.att(v) \geq \text{thr}(v)] \) and stores \( b \) at the right child node \((v.right.cmp = b)\) and \( 1-b \) at the left child node \((v.left.cmp = 1-b)\). It is illustrated in Algorithm 1.

10.1. Initialization. The initialization consists of a one-time key generation. One server \( S_i (i \in \{0,1\}) \) generates appropriate triple \((\text{pk}, \text{sk}, \text{ek})\) of public, private, and evaluation keys for a homomorphic encryption scheme. Then, another server \( S_{1-i} \) sends \((\text{pk}, \text{ek})\) to the server. For each input classification, \( S_i \) just encrypts its input and sends it to the other \( S_{1-i} \). To reduce the communication cost of sending input of \( S_{1-i} \), \( S_i \) can use a trusted randomizer that does not take part in the real protocol and is not allowed to collaborate with \( S_{1-i} \). The trusted randomizer generates a list of random strings \( r \) and sends the encrypted strings \( \lfloor r \rfloor \) to server and the list of \( r \) to \( S_{1-i} \). For an input \( x \), this server \( S_{1-i} \) then sends \( x + r \) to the server \( S_i \) in the real protocol. This technique is similar to the commodity-based cryptography with the difference that \( S_{1-i} \) can play the role of the randomizer itself and sends the list of \( \lfloor r \rfloor \)’s (when the network is not too busy) before the protocol setting.

10.2. Computing Decision Bits. The server starts by computing for each node \( v \in \mathcal{D} \) the comparison bit \( b \leftarrow [x.att(v) \geq \text{thr}(v)] \) and stores \( b \) at the right child node \((v.right.cmp = b)\) and \( 1-b \) at the left child node \((v.left.cmp = 1-b)\). It is illustrated in Algorithm 2.

10.3. Aggregating Decision Bits. Then for each leaf node \( v \), the server aggregates the comparison bits along the path from the root to \( v \). We implement it using a queue and traversing the tree in BFS as illustrated in Algorithm 1.

10.4. Finalizing. After aggregating the decision bits along the path to the leave nodes, each leaf node \( v \) stores either \( v.cmp = 0 \) or \( v.cmp = 1 \). Then, the server aggregates the decision bits at the leaves by computing for each leaf \( v \) the value \( [v.cmp] \oplus [v.cLabel] \) and splitting the data into \( b \) and \( 1-b \) for each input classification.

The comparison operation is used to select the maximum Gini impurity value. Algorithm 4 gives a specific comparison protocol Compare((\( [x] \), \( [y] \)) based on FSS, which outputs the shares of \( z = 1 \) \( \lfloor y \geq x \rfloor \). Note that the comparison protocol is executed over the secret-shared inputs rather than public values, which should be supported by our designed FSS scheme. As a result, the key idea is to construct the FSS scheme for the offset function \( f_{[r]} (x) = f(x + r) \), where \( r \) is randomly selected from \( \mathbb{Z}_2 \) and secret sharing between \( S_0 \) and \( S_1 \). In this way, \( S_0 \) and \( S_1 \) then receive \( x + r \) and then evaluate \( f_{[r]} (x) \), which exactly equals to evaluating \( f(x) \). Note that the offset function fails if \( x + r \) wraps around. Our protocol only invokes 1 DCF and introduces \( 2n \) communication bits within 1 round in the setup phase.

11. Secure Two-Party Decision Tree Classification

In this section, we present our secure two-party decision tree classification protocol that caters to scenarios where two counterpart parties provide privacy information, and both parties can own a decision tree model (see Figure 1). The proposed protocol ensures that both servers possess knowledge of the classification results but only of the individual inputs of the self-party. Our protocol is designed to be secure for “honest and curious” parties.

To establish the necessary functionality for the tree array \( A_\mathcal{T} \) and feature array \( \mathcal{F}, S_0 \) and \( S_1 \) perform the required setup work for function secret sharing. Additionally, \( S_i \) shares the root node \( A_\mathcal{T}[0] \) with \( S_1 \), which serves as the starting evaluation node. Our secure two-party decision tree classification protocol provides an effective solution for secure data analysis while maintaining data privacy. It enables both parties to access the classification results without compromising sensitive information, thereby ensuring transparency in the data analysis process.

In each iteration, the evaluation process starts with the call of the FSS functionality \( \mathcal{F} \) on \( S_i \), which initiates the sharing of \( \mathcal{F}[v] \) among the parties. The parties then perform a secure comparison between \( \mathcal{F}[v] \) and \( \text{thr} \), the purpose of which is to obtain a comparison result, denoted as \( b \). Subsequently, the MUX computation determines which child becomes the next evaluation node. The computation of
this decision incorporates the application of the XOR operator on two values: \(v_{\text{left}}\) and \(v_{\text{right}}\). In such a case where \(b\) equals 1, \([i\text{dx}]\) becomes equal to \(v_{\text{left}}\). Conversely, if \(b\) does not equal 1, \([i\text{dx}]\) becomes equal to \(v_{\text{right}}\). Thus, determining the next evaluation node during each iteration depends on the outcome of the secure comparison and the MUX computation.

From the shared index \(i\text{dx}\), the parities invoke \(\mathcal{F}\) to share \(A_\mathcal{F} [i\text{dx}]\). \(v_{\text{threshold}}\), \(v_{\text{left}}\), \(v_{\text{right}}\), and \(v_{\text{cLabel}}\) are then updated correspondingly. Besides, \(v_{\text{cLabel}}\) is stored in \([\text{rst}]\) where the final classification label will stay in. Note that we encode a self-loop for each leaf node, and thus \([\text{rst}]\) will always hold a correct classification label once the evaluation reaches a leaf node. Moreover, it is easy to hide length information: \(S_0\) and \(S_1\) just run \(d\) iterations of evaluation. In the end, \(S_0\) sends \([\text{rst}]_0\) to \(S_1\), and \(S_1\) recovers \(\text{rst}\) as classification result. The protocol runs in \(O(d)\) iterations with \(d\) secure comparison and MUX operations. If the comparison protocol is used over \(\mathcal{X}\) and a function secret sharing protocol is used over \(A_\mathcal{F}\).
Lemma 8 (Correctness). Assuming the evaluation correctness of the underlying FSS scheme and our Eval algorithm, then the above construction is a dual-server private decision tree classification protocol, outputting the correct classification label.

Proof. Based on the correctness of FSS and Eval, the cost $t_F$ is equal to 0 if and only if $v.cmp$ is equal to 0; then the corresponding result satisfies that result: $r = r^*_1 \times f_0 + \ell \cdot 0 + \ell_{label}$.

Lemma 9 (Security). The algorithm $\text{Compare}(\mathbb{X}, \mathbb{Y})$ securely realizes the functionality $F_{\text{Compare}}$ assuming the existence of secure protocols for FSS procedures.

Proof. We prove the security of $\text{Compare}(\mathbb{X}, \mathbb{Y})$. $S_i$ receives no private information of $S_{1-i}$, $i \in \{0, 1\}$, and hence this protocol is trivially secure against "curious but honest" adversary. Now, we prove the security against corruption of $S_i, S_{1-i}, i \in \{0, 1\}$, when server receives $[b]_{1-i} = [y]_{1-i} - [x_{1-i}] + r_{1-i}$ and $k_i$. Given the security of PRFs, $[r]_{1-i}$ is a random value unknown to $S_i$. Thus, the distribution of $[b]_{1-i}$ is uniformly random from the view of $S_i$. Then given the security of FSS, the information learned by $S_i$ can be perfectly simulated. Hence, our protocol is trivially secure against "curious but honest" corruption of $S_i$.

12. Security Analysis

We now present a formal security proof of our secure two-party decision tree classification protocol described in this section. We show that the protocol satisfies computational semihonest security by proving the existence of probabilistic polynomial-time (PPT) simulators whose output is computationally indistinguishable from the real view of each party during the protocol execution.

Let $\text{VIEW}_0(x_0, x_1)$ denote the view of party $S_0$ during an execution of the protocol on inputs $(x_0, x_1)$, consisting of its input $x_1$, internal random coins $r_1$, and received messages. Similarly, $\text{VIEW}_1(x_0, x_1)$ denotes the view of party $S_1$. We construct the following PPT simulators (Algorithms 5 and 6).

We now show that the output of each simulator is computationally indistinguishable from the real view.

Theorem 10. The secure two-party decision tree classification protocol satisfies computational semihonest security. Formally:

Sim0$(x_0, f(x_0, x_1)) \approx \text{VIEW}_0(x_0, x_1),$
Sim1$(x_1, f(x_0, x_1)) \approx \text{VIEW}_1(x_0, x_1).

The SETUP message and PRF randomness $r'_0$ generated by Sim1 are identically distributed as in the real protocol execution. The simulated transcript consists of

(1) Encrypted inputs computed on $x_0, x_1'$
(2) FSS keys generated independently of inputs
(3) Encrypted outputs that encrypt results from $x_0, x_1'$

These are all computationally indistinguishable from the real transcript due to the IND-CPA security of the encryption scheme and the security of the FSS scheme. Therefore, Sim0$(x_0, f(x_0, x_1)) \approx \text{VIEW}_0(x_0, x_1)$. By a similar argument, we can show Sim1$(x_1, f(x_0, x_1)) \approx \text{VIEW}_1(x_0, x_1)$. Since PPT simulators Sim0 and Sim1 exist where the output is computationally indistinguishable from the real view of each party during the protocol execution.
party, this proves that the protocol satisfies computational semihonest security.

This security proof demonstrates that our protocol protects the privacy of each party’s inputs and decision tree model during the secure two-party computation. By simulating the views using arbitrary inputs, we have shown that the views leak no additional information beyond the intended output. Therefore, our protocol provides provable security guarantees for practical applications requiring privacy-preserving decision tree classification.

13. Experiment

We present experimental results evaluating the performance of our secure two-party decision tree classification protocol on the MNIST dataset [54]. Specifically, we analyze the impact on accuracy of varying the number of training epochs. We also benchmark the runtime of training and inference under different model configurations. Finally, we compare our approach to prior frameworks from related works regarding efficiency and accuracy.

14. Experimental Setup

In our study, we implement the secure two-party decision tree training algorithm in Python. To facilitate communication between parties, we utilize the communication backend of the Porthos framework in EzPC [55]. We employ a pseudorandom function (PRF) based on the block cipher AES using the OpenSSL-AES library [56, 57].

At the same time, the fully homomorphic secret sharing (FSS) schemes are implemented using the LibFSS library. The implementation is executed on two terminals with Intel(R) Core(R) CPU i7-6700 running the Ubuntu 18.4 operating system and 16 GB of RAM, with each terminal representing a party ($S_0$ and $S_1$). The reported communication overhead includes the communication between the two parties, while the runtime incorporates the computational costs of local computation within each entity and the communication latency between them. For experiments conducted over a local area network (LAN), we assume a bandwidth of 2 Gbps and an echo latency of 0.3ms. We use secret-sharing protocols over the ring $\mathbb{Z}_{2^{264}}$ following existing works [23, 58]. We encode the inputs using a fixed-point representation with a precision of 20 bits.

Our implementation demonstrates the practical viability of secure two-party decision tree training for data analysis applications prioritising privacy. By leveraging commonly available resources such as Python and the Porthos framework in EzPC, we provide a simple yet effective solution that can be quickly adopted for particular data analysis tasks. In summary, our study presents an efficient and practical approach to implementing secure two-party decision tree training, providing insights into designing secure data analysis systems for real-world applications.

In this section, we give the accuracy of secure two-party decision classification. Our study aims to evaluate the effectiveness of secure two-party decision tree classification by conducting several epochs of training on a decision tree classifier. Specifically, we perform 5, 10, and 15 epochs of training on the model and record the corresponding accuracies obtained in each case.

Upon analyzing the results, we present the findings in Table 2. Due to space constraints, we only report the plaintext training results and the corresponding secure training results. The table shows that the trend in secure training accuracy is similar to that of plaintext training accuracy, with no discernible fluctuations. Furthermore, the difference between the accuracy obtained from secure and plaintext training is approximately $\pm 0.05\%$. These results suggest that the secure two-party decision tree classification method effectively achieves high accuracy while preserving data privacy. To sum up, the experimental results demonstrate that secure two-party decision tree classification can achieve performance comparable to plaintext training, with only a negligible difference in accuracy, making it an up-and-coming method for secure data analysis.

In our study, we investigate the impact of varying the number of training data and the maximum tree depth on the communication overhead of secure two-party decision tree classification. Firstly, we examine the relationship between
the number of training samples and the communication cost. Figure 2 shows that as the number of training samples increases, both the cost of data and communication grow roughly linearly. This result can be attributed to the secure two-party decision tree training phase requiring more multiplication operations to compute the impurity gain with a more significant number of training samples. Secondly, we explore the impact of varying the maximum tree depth on the communication overhead. As the well-trained tree tends towards a complete binary tree, approximately $2^{d-1}$ internal nodes are constructed for a given depth $d$. Therefore, as shown in Figure 3, the communication overhead increases logarithmically with the tree depth. This result is because deeper trees require more computation and excellent communication between parties.

Our results show that the communication overhead is influenced by critical factors such as the number of training samples and the maximum tree depth. As such, it is essential to carefully consider these factors when designing secure two-party decision tree classification systems to ensure optimal performance while maintaining data privacy. In conclusion, our study highlights the need for efficient and secure methods for decision tree classification, especially in situations where data privacy is of utmost importance.

The study [59] presents an initial GPU-based implementation of function secret sharing, although further optimizations could reduce the memory footprint of cryptographic keys by approximately 50% to match theoretical minimum bounds. Moreover, the marginal divide

### Table 2: The performance of secure two-party decision tree classification.

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Ours</th>
<th>Plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>78.41%</td>
<td>75.13%</td>
</tr>
<tr>
<td>10</td>
<td>75.19%</td>
<td>75.56%</td>
</tr>
<tr>
<td>15</td>
<td>75.37%</td>
<td>75.94%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (h)</th>
<th>Ours</th>
<th>Plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.24</td>
<td>0.09</td>
</tr>
<tr>
<td>10</td>
<td>0.47</td>
<td>0.17</td>
</tr>
<tr>
<td>15</td>
<td>0.73</td>
<td>0.29</td>
</tr>
</tbody>
</table>

### Figure 2: Communication: training data.

### Figure 3: Communication: tree depth.

between LAN and WAN runtimes intimates that computational overhead supersedes communication for sufficiently extensive networks. Thus, optimizing GPU-centric calculations proffers the potential to enhance overall efficiencies of inference and training paradigms markedly.

### 15. Discussion

We have demonstrated the utility of function secret sharing for private training and evaluation of decision trees. Compared to related works, our protocols are highly competitive and achieve negligible failure rates for ML applications.

Numerous opportunities remain to improve performance further and expand the applicability of private ML via function secret sharing. Running experiments at 16 bit precision versus 32 bit could be another promising improvement, as major ML frameworks now support 16 bit encoding on CPU. Reducing key sizes, leveraging lower precision, and GPU optimizations can help overcome scaling bottlenecks. Testing new model architectures and data modalities will be essential to gauge general viability. Overall, there is tremendous promise in employing function secret sharing primitives to enable practical secure computation for diverse machine learning pipelines.

Moreover, a core technical challenge in applying homomorphic encryption (HE) to secure machine learning is managing the noise growth inherent in lattice-based cryptosystems. Our scheme introduces randomness into the ciphertext to ensure security when applying HE. However, each subsequent homomorphic operation (addition or multiplication) also accumulates and amplifes this noise. Excessive noise during HE evaluation inhibits correct decryption and reduces arithmetic fidelity. In the context of secure decision tree protocols, imprecise calculations may propagate errors when calculating attribute thresholds and reduce model accuracy. While multiplication noise worsens exponentially, even repeated additions can produce considerable noise.

We employ techniques, including optimized circuitry, modular design, and regular ciphertext refresh, to suppress noise. However, some accuracy loss may still occur for deep
trees and large datasets. We will empirically quantify the potential degradation in accuracy due to noise in future work. Analyzing the impact on accurate data will better reveal the actual impact. If noise-induced inaccuracies prove unacceptable, an alternative, homomorphic encryption scheme with slower noise growth is a better choice. However, these usually require more computational overhead. Developing robust protocols for large amounts of noise remains an open problem when applying HE to machine learning.

The inherent stochasticity of lattice-based HE affects model accuracy when noise accumulates across multiple operations. While this paper mitigates noise growth through multiple strategies, more empirical analysis is needed to determine the extent of this problem in practice. Managing noise persistence remains an active research challenge in building efficient and accurate protocols for the secure computation of encrypted data. We have demonstrated the utility of function secret sharing in private training and evaluation of decision trees. Our protocol is highly competitive compared to related work and suffers a negligible failure rate for machine learning applications.

16. Conclusions

In conclusion, our research has introduced a two-party secure decision tree classification protocol that offers low communication and computational costs and minimal client interaction. Our approach enhances the practical implementation of the solution by improving the multiplication depth of the tree evaluation circuit and the efficiency of the underlying general FSS solution. Notably, we have utilized a unique approach of adding relatively small amounts of blurring noise by each participant in threshold decryption, resulting in a considerable reduction in the overall computational cost and ciphertext size of FSS. Together, our contributions have enabled the application of our protocol with a lower computational overhead while maintaining a higher level of security.

Data Availability

No underlying data were collected or produced in this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant no. 12171114.

References


