Hindawi
Complexity
Volume 2023, Article ID 5302915, 13 pages
https://doi.org/10.1155/2023/5302915

Research Article

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

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 semihonest 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

Table 1: Comparison to prior work.

	Our results	De Cock et al. [20]	Lu et al. [21]
Accuracy drop	0.05%	2–5%	—
Communication per comparison	2 ring elements	Information-theoretic	100 ring elements
Secure computation technology	Homomorphic encryption, secret sharing	Secret sharing	Homomorphic encryption
Security model	Semihonest	Malicious	Semihonest

client subsequently executes. This protocol combines homomorphic encryption and oblivious transmission, which enables the client to discover its garbling key (OT). While the evaluation time is sublinear in the tree size, the technique could be more efficient for large trees due to the linear secure program and communication cost. Barni et al. [10] improved upon this technique by removing the leaf node from the secure program, thereby reducing calculation costs by a constant factor.

Bost et al. [11] have modelled the decision tree as a multivariate polynomial where the constants in the polynomial signify the classification labels and the variables signify the outcomes of the Boolean conditions at the decision nodes. In order to clandestinely calculate the value of the Boolean conditions, each threshold is matched with the respective encrypted attribute values under the client's public key. Subsequently, the client receives the result once the server homomorphically evaluates the polynomial.

In their study, Wu et al. [4] have employed various methods that exclusively require additive homomorphic encryption (AHE). They have also used the protocol from [36] to compare data and broadcast the encrypted comparison bits with the client's public key to the server. Upon evaluating the tree, the server communicates the client's index of the matching categorization label, and the outcome is conveyed to the client via an OT. Tai et al. [15] have implemented the comparison methodology of [36] and AHE in their work. They have assigned costs to the left and right edges of each node, namely, b and $1-b$, respectively, where b is the result of the comparison at that specific node. Ultimately, the costs are tallied along each tree branch, and the classification label pertains to the path that yields zero cost.

Tueno et al. [16] have represented the tree as an array and conducted comparisons of the depth of the tree using small garbled circuits to obtain secret shares of the subsequent node's index along the tree. They have also introduced a novel primitive called oblivious array indexing to enable the selection of the following nodes without memorization. Using a modular approach, Kiss et al. [14] have incorporated subfunctionalities such as attribute selection, integer comparison, and route evaluation. For covertly computing these subfunctionalities, they have thoroughly examined the trade-offs and performance of various potential combinations of reduction protocols.

De Cock et al. [20] utilized a similar approach to earlier methods by initially carrying out comparisons. To minimize interactions, they have implemented secret sharing-based secure multiparty computation (SMC) and commodity-based cryptography [37] in an information-theoretic model. In contrast to ours and other protocols, 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 $\mathcal{T}\colon \mathbb{Z}^k \longleftarrow \mathbb{Z}$. 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 $\mathrm{Bool}(x) = 1, \{x_{i_k} < t_k\}$ is connected to each child node v_k in the tree, where $i_k \in [n]$ 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 $\mathcal{T}\colon \mathbb{Z}^k \longleftarrow \{\ell^1_{\text{label}}, \ldots, \ell^m_{\text{label}}\}$, 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 $x_{f(j)} \geq t_j$ 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 ℓ_{label} of the final leaf node.

A decision tree (DT) is a function $\mathcal{T}\colon \mathbb{Z} \longrightarrow \{c_0, \ldots, c_{k-1}\}$ that maps an attribute vector $x = (x_0, \ldots, x_{n-1})$ 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 leave 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 thr: $[0, m-1] \longrightarrow \mathbb{Z}$.

(ii) An attribute index is assigned to each decision node by the function att: $[0, m-1] \longrightarrow [0, n-1]$.

(iii) A label of each leaf node is assigned to each decision node by a labeling function lab: $[m, M-1] \longrightarrow \{c_0, \ldots, c_{k-1}\}$.

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) $(\text{pk}, \text{sk}, \text{ek}) \longleftarrow \text{KGen}(\lambda)$: 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 λ to ensure the randomness and security of the generated keys.

(ii) $c \longleftarrow \text{Enc}(\text{pk}, m)$: 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) $c \longleftarrow \text{Eval}(\text{ek}, f, c_1, \ldots, c_n)$: 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) $m' \longleftarrow \text{Dec}(\text{sk}, c)$: 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 Gen and Eval with the following syntax. We identify two FSS constructions [28, 40] as a natural fit for our scheme: (1) distributed point function (DPF) $(\text{Gen}_{a,b}^{\text{DPF}}, \text{Eval}_{a,b}^{\text{DPF}})$ that satisfies $f_{a,b}(x) = b$ is $x = a$ and 0 otherwise and (2) distributed comparison function (DCF) $(\text{Gen}_{a,b}^{\text{DCF}}, \text{Eval}_{a,b}^{\text{DCF}})$ that satisfies $f_{a,b} = b$ is $x < a$ and 0 otherwise.

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

(1) $\text{Gen}(1^{\kappa}, \widehat{f})$ is a probabilistic polynomial-time (PPT) key generation algorithm that given secure parameter 1^{κ} and a function $\widehat{f} \in \{0,1\}^*$ outputs a pair of keys $(k_0, \ldots, k_{\sigma})$. We assume that $\widehat{f}$ explicitly contains descriptions of input and output groups $\mathbb{G}_{\text{in}}, \mathbb{G}_{\text{out}}$.

(2) $\text{Eval}(\sigma, k_{\sigma}, x)$ is a polynomial-time evaluation algorithm that given $\sigma \in \{0, \ldots, m\}$ (party index), k_{σ} is defined as the key of function $f_{\sigma}\colon \mathbb{G}_{\text{in}} \longleftarrow \mathbb{G}_{\text{out}}$. Let $x \in \mathbb{G}_{\text{in}}$ be the input of function f_{σ} and output a group element $y_{\sigma} \in \mathbb{G}_{\text{out}}$.

When σ 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 $\widehat{f}$. When Leak is omitted, it is understood to output only $\mathbb{G}_{\text{in}}, \mathbb{G}_{\text{out}}$. We say that (Gen, 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 $\widehat{f}\colon \mathbb{G}_{\text{in}} \longleftarrow \mathbb{G}_{\text{out}}$ and every $x \in \mathbb{G}_{\text{in}}$, if $(k_0, k_1) \longrightarrow \text{Gen}(1^{\lambda}, \widehat{f})$, 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 Sim_{σ} (simulator), such that for every sequence $(\widehat{f}_{\lambda})_{\lambda \in \mathbb{N}}$ of polynomial-size function descriptions from $\mathcal{F}$ and polynomial-size input sequence x_{λ} for f_{λ}, the outputs of the following experiments Real and Ideal are computationally indistinguishable:

(i) $\text{Real}_{\lambda}\colon (k_0, k_1) \longleftarrow \text{Gen}(1^{\lambda}, \widehat{f}_{\lambda})$; output k_{σ}.
(ii) $\text{Ideal}_{\lambda}\colon$ Output $\text{Sim}_{\sigma}(1^{\lambda}, \text{Leak}(\widehat{f}_{\lambda}))$.

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 β if $x < \alpha$ and 0 otherwise. We refer to an FSS scheme for comparison functions as DCF. Analogously, function $f_{\alpha,\beta}^{\leq}$ outputs β if $x \leq \alpha$ and 0 otherwise. In all of these cases, we allow the default leakage $\text{Leak}(\widehat{f}) = (\mathbb{G}_{\text{in}}, \mathbb{G}_{\text{out}})$.

Theorem 5 (concrete cost of DCF). *Given a PRGG: $\{0,1\}^{\lambda} \longleftarrow \{0,1\}^{(2\lambda+2)}$, there exists a a DCF for $f_{\alpha,\beta}^{<}\colon \mathbb{G}_{in} \longleftarrow \mathbb{G}_{out}$ with key size $4n \cdot (\lambda + 1) + n\ell + \lambda$, where $n = \lceil \log|G^{in}| \rceil$ and $\ell = \lceil \log|G^{out}| \rceil$. For $\ell' = \lceil \ell/\lambda + 2 \rceil$, the key generation algorithm Gen invokes G at most $n \cdot (4 + \ell')$ times and the algorithm Eval invokes G at most $n \cdot (2 + \ell')$ times.*

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

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 $\mathcal{M} = (\mathcal{T}, \text{thr}, \text{att})$, we denote the tree $\mathcal{T}$ of each node v which consists of

(i) v.threshold: the threshold of node v, denoted by thr(v), is stored in the variable v.threshold.
(ii) v.aIndex: the associated index, denoted by att(v), is stored in the variable v.aIndex.
(iii) v.parent: A pointer to the parent node is stored in the variable v.parent. For the root node, this pointer is null.
(iv) v.left: Pointers to the left child nodes are stored in the variables v.left. For leaf nodes, these pointers are null.
(v) v.right: Pointers to the right child nodes are stored in the variables v.right. For leaf nodes, these pointers are null.
(vi) v.cmp: During tree evaluation, the comparison bit $b \longleftarrow [x_{\text{att}(v.\text{parent})} \geq x_{\text{thr}(v.\text{parent})}]$ 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 $\mathscr{M}=(\mathscr{D},\mathscr{L})$. We define the classification function to be $f_c(x,\mathscr{M})=\mathrm{tr}(x,\mathrm{root})$, where root is the root node and tr is the traverse function defined as

$$\mathrm{tr}(x,v)=\begin{cases}\mathrm{tr}(x,v.\mathrm{left}), & \text{if } v\in\mathscr{D} \text{ and } x_{v.\mathrm{Index}}<v.\mathrm{threshold},\\ \mathrm{tr}(x,v.\mathrm{right}), & \text{if } v\in\mathscr{D} \text{ and } x_{v.\mathrm{Index}}\geq v.\mathrm{threshold},\\ v, & \text{if } v\in\mathscr{L}.\end{cases} \tag{1}$$

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 $(\mathrm{pk},\mathrm{sk},\mathrm{ek})$, which consists of the public, private, and evaluation keys. Following this, $S_i\,(i=\{0,1\})$ sends $(\mathrm{pk},\mathrm{ek})$ 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 $[\![r]\!]$ before the start of the protocol when the network is not overloaded.

The server starts by computing for each node $v\in\mathscr{D}$ the comparison bit $b\longleftarrow[x_{\mathrm{att}(v)}\geq\mathrm{thr}(v)]$ and stores b at the right child node $(v.\mathrm{right.cmp}=b)$ and $1-b$ at the left child node $((v.\mathrm{left.cmp}=1-b))$. 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 $(\mathrm{pk},\mathrm{sk},\mathrm{ek})$ of public, private, and evaluation keys for a homomorphic encryption scheme. Then, another server S_{1-i} sends $(\mathrm{pk},\mathrm{ek})$ 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 $[\![r]\!]$ 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 $[\![r]\!]$'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 $v\in\mathscr{D}$ the comparison bit $b\longleftarrow[x_{\mathrm{att}(v)}\geq\mathrm{thr}(v)]$ and stores b at the right child node $(v.\mathrm{right.cmp}=b)$ and $1-b$ at the left child node $(v.\mathrm{left.cmp}=1-b)$. 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 $v.\mathrm{cmp}=0$ or $v.\mathrm{cmp}=1$. Then, the server aggregates the decision bits at the leaves by computing for each leaf v the value $[\![v.\mathrm{cmp}]\!]\oplus[\![v.\mathrm{cLabel}]\!]$ and summing all the results. This is illustrated in Algorithm 3.

The comparison operation is used to select the maximum Gini impurity gain. Algorithm 4 gives a specific comparison protocol $\mathrm{Compare}([\![x]\!],[\![y]\!])$ based on FSS, which outputs the shares of $z=1\{y>x\}$. 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 $\mathbb{Z}_{2^n}$ and secret sharing between S_0 and S_1. In this way, S_0 and S_1 first reconstruct $x+r$ 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 $A_{\mathscr{T}}$ and feature array $\mathscr{X}$, S_0 and S_1 perform the required setup work for function secret sharing. Additionally, S_0 shares the root node $A_{\mathscr{T}}[0]$ 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 FFS functionality $\mathscr{F}$ on S_i, which initiates the sharing of $\mathscr{X}[\mathrm{v}]$ among the parties. The parties then perform a secure comparison between $[\![\mathscr{X}[v]]\!]$ and 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

```
(1) function EVALPATHS ($\mathscr{D}, \mathscr{L}$)
(2)   let Q be a queue3
(3)   Q.enqueue(root)
(4)   while Q.empty = false do
(5)     $v \longleftarrow$ Q.dequeue()
(6)     $[\![v.\text{left.cmp}]\!] \longleftarrow [\![v.\text{left.cmp}]\!] \odot [\![v.\text{cmp}]\!]$
(7)     $[\![v.\text{right.cmp}]\!] \longleftarrow [\![v.\text{right.cmp}]\!] \odot [\![v.\text{cmp}]\!]$
(8)     if $v$.left $\in \mathscr{D}$ then
(9)       Q.enqueue($v$.left)
(10)    if $v$.right $\in \mathscr{D}$ then
(11)      Q.enqueue($v$.right)
```

ALGORITHM 1: Aggregating decision bits.

```
(1) function EVALDNODE ($\mathscr{D}, [\![x]\!]$)
(2)   for each $v \in \mathscr{D}$ do
(3)   $[\![b]\!] \longleftarrow [\![ [x_{v.\text{Index}} \geq v.\text{threshold}] ]\!]$
(4)   $[\![v.\text{right.cmp}]\!] \longleftarrow [\![b]\!]$
(5)   $[\![v.\text{left.cmp}]\!] \longleftarrow [\![1 - b]\!]$
```

ALGORITHM 2: The decision bit computation.

```
(1) function FINALIZE ($\mathscr{L}$)
(2)   [[result[[ $\longleftarrow [\![0]\!]$
(3)     for each $v \in \mathscr{L}$ do
(4)       $[\![\text{result}]\!] \longleftarrow [\![\text{result}]\!] \oplus ([\![v.\text{cmp}]\!] \odot [\![v.\text{cLabel}]\!])$
(5)     return $[\![\text{result}]\!]$
```

ALGORITHM 3: Finalizing.

```
(1) function COMPARE ($[\![x]\!], [\![y]\!]$)
(2)   $S_i$ generate $[\![r]\!]_0$ using PRFs with seed.
(3)   $S_i$ samples $r \in \mathbb{Z}_{z^n}$ and sends $[\![r]\!]_1 = r - [\![r]\!]_0$ to $S_{1-i}$
(4)   $S_i$ evaluates $(k_0, k_1) \longleftarrow \text{Gen}_{r,1}$ and sends $k_i$ to $S_i$
(5)   $S_i$ sends $[\![y]\!]_i - [\![x]\!]_i + [\![r]\!]_i$ to $S_{1-i}$, and
(6)   $S_i$ evaluates $[\![x]\!]_i \longleftarrow \text{Eval}_{r,1}(i, k_i, y - x + r)$
(7) Return $[\![z]\!]$
```

ALGORITHM 4: Secure comparison.

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, $[\![\text{idx}]\!]$ becomes equal to $v.[\![\text{left}]\!]$. Conversely, if b does not equal 1, $[\![\text{idx}]\!]$ 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 idx, the parities invoke $\mathscr{F}$ to share $A_{\mathscr{T}}[\text{idx}]$. v.threshold, v.left, v.right, v, and v.cLabel are then updated correspondingly. Besides, v.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 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 $\mathscr{X}$and a function secret sharing protocol is used over $A_{\mathscr{T}}$.

```
Parameters: Computational security functionality $\mathcal{F}$. $S_i$ provides a tree $\mathcal{T}$, where $i \in \{0,1\}$. $S_i$ provides a feature vector $\mathcal{X}$, and the longest depth $d$.
Setup:
  1. $S_i$ encodes its decision tree $\mathcal{T}$ to an array $\mathsf{A}_{\mathcal{T}}$.
  2. $\mathrm{S_i}$ invoke functionality $\mathcal{F}$. $\mathrm{S_i}$ sends SETUP, $\mathsf{A}_{\mathcal{T}}$, $\ell$ to $\mathcal{F}$, and $\mathrm{S_{1-i}}$ sends SETUP to $\mathcal{F}$.
  3. $\mathrm{S_{1-i}}$ invoke functionality $\mathcal{F}$. $\mathrm{S_i}$ sends SETUP, $\mathsf{A}_{\mathcal{T}}$, $\ell$, to $\mathcal{F}$, and $\mathrm{S_i}$ sends SETUP to $\mathcal{F}$.
  4. $\mathrm{S_0}$ and $\mathrm{S_1}$ invoke functionality F. $\mathrm{S_1}$ sends SETUP, $\mathcal{X}$, $\ell$ to $\mathcal{F}$, and $\mathrm{S_0}$ sends SETUP to $\mathcal{F}$.
  5. $\mathrm{S_0}$ shares root node $\mathsf{A}_{\mathcal{T}}[0]$ with $\mathrm{S_1}$. Both parties parse $\mathsf{A}_{\mathcal{T}}[0]$ as secret.
Evaluation:
  1. for $j \in [1, d]$ do
     a. $S_i$ sends (EVAL, $[\![v]\!]_i$) and $S_{1-i}$ sends (EVAL, $[\![v]\!]_{1-i}$) to FSS function $\mathcal{F}$. In the end, both parties share $\langle \mathcal{X}[v] \rangle$.
     b. $S_i$ execute secure comparison algorithm 3 to compute $[\![b]\!] \leftarrow [\![\mathcal{X}[v]]\!] > [\![t]\!]$.
     c. $S_i$ compute the index of next tree node $[\![\mathsf{idx}]\!] \leftarrow [\![v.\mathsf{left}]\!] \oplus [\![b]\!] \cdot ([\![v.\mathsf{left}]\!] \oplus [\![v.\mathsf{right}]\!])$.
     d. $S_i$ sends EVAL$[\mathsf{idx}]_i$, and $S_{1-i}$ sends EVAL$[v]_{1-i}$ to $F$. In the end, both parties share $[\![\mathsf{A}_{\mathcal{T}}[\mathsf{idx}]]\!]$.
     e. update ($[\![v.\mathsf{threshold}]\!]$, $[\![v.\mathsf{left}]\!]$, $[\![v.\mathsf{right}]\!]$, $[\![v]\!]$, $[\![v.\mathsf{cLabel}]\!]$) $\leftarrow [\![\mathcal{A}_{\mathcal{T}}[\mathsf{idx}]]\!]$.
     f. set $[\![\mathsf{rst}]\!] \leftarrow [\![v.\mathsf{cLabel}]\!]$
  2. $S_i$ and $S_{1-i}$ reveal rst to $S_i$ as output.
```

Figure 1: Secure two-party decision tree classification.

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 cos t_ℓ is equal to 0 if and only if v.cmp is equal to 0; then the corresponding result satisfies that $\text{result}_\ell = r_1^\ell \times 0 + \ell_{\text{label}} = \ell_{\text{label}}$. □

Lemma 9 (security). *The algorithm Compare($[\![x]\!]$, $[\![y]\!]$) securely realizes the functionality $\mathcal{F}_{Compare}$, assuming the existence of secure protocols for FSS procedures.*

Proof. We prove the security of Compare($[\![x]\!]$, $[\![y]\!]$). 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:*

$$\begin{aligned} \text{Sim0}(x_0, f(x_0, x_1)) &\approx \text{VIEW}_0(x_0, x_1), \\ \text{Sim1}(x_1, f(x_0, x_1)) &\approx \text{VIEW}_1(x_0, x_1). \end{aligned} \tag{2}$$

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, $\text{Sim0}(x_0, f(x_0, x_1)) \approx \text{VIEW}_0(x_0, x_1)$. By a similar argument, we can show $\text{Sim1}(x_1, f(x_0, x_1)) \approx \text{VIEW}_1(x_0, x_1)$. Since PPT simulators Sim0 and Sim1 exist where the output is computationally indistinguishable from the real view of each

```
(1) Generate random coins $r_0'$ for PRF evaluation
(2) Generate SETUP message to FSS functionality $\mathcal{F}$
(3) Run protocol execution locally on inputs $(x_0, x_1')$, outputting $f(x_0, x_1)$
(4) Use $r_0'$ as randomness and arbitrary $x_1'$ as input
(5) Output view $(x_0, r_0', \text{simulated transcript})$
```

Algorithm 5: Simulator sim0.

```
(1) Generate random coins $r_0'$ for PRF evaluation
(2) Generate SETUP message to FSS functionality $\mathcal{F}$
(3) Run protocol execution locally on inputs $(x_0, x_1')$, outputting $f(x_0, x_1)$
(4) Use $r_0'$ as randomness and arbitrary $x_1'$ as input
```

Algorithm 6: Simulator sim1.

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 $\mathbb{Z}_{2^{64}}$ 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

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

	Epoch	Ours	Plaintext
Accuracy	5	78.41%	75.13%
	10	75.19%	75.56%
	15	75.37%	75.94%
Time (h)	5	0.24	0.09
	10	0.47	0.17
	15	0.73	0.29

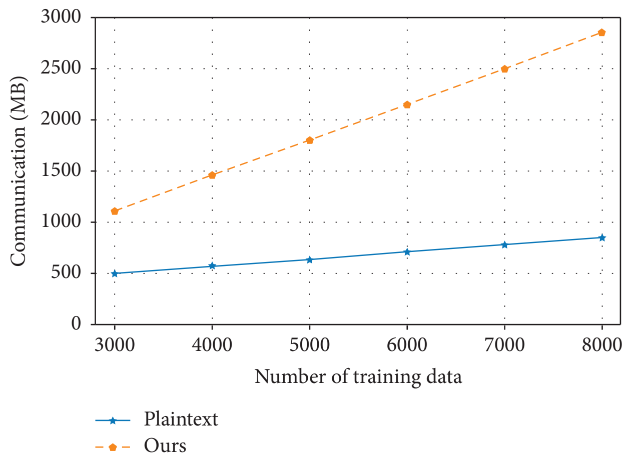

Figure 2: Communication: training data.

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 $2h - 1$ 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

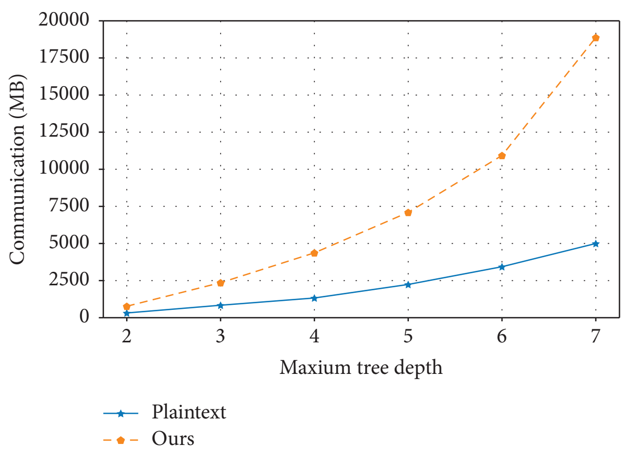

Figure 3: Communication: tree depth.

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 amplifies 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

[1] H. W. Ian, E. F. Rank, and M. Hall, "Data mining: practical machine learning tools and techniques," *Annals of Physics*, vol. 54, no. 2, 2011.

[2] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333, Denver, CO, USA, October 2015.

[3] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proceedings of the USENIX security symposium*, vol. 16, pp. 601–618, Santa Clara, CA, USA, August 2016.

[4] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pp. 355–370, IEEE, Lisbon, Portugal, June 2016.

[5] R. Cramer, I. Damgård, and B. Jesper, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck*, vol. 20, pp. 280–300, Springer, Berlin, Germany, 2001.

[6] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*, Cambridge University Press, Cambridge, UK, 2009.

[7] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pp. 160–164, IEEE, Washington, DC, USA, November 1982.

[8] L. Chang, S. Xiao, K. Nayak, Y. Huang, and E. Shi, "Oblivm: a programming framework for secure computation," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 359–376, IEEE, Washington, DC, USA, May 2015.

[9] M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, and H. Veith, "Cbmc-gc: an ansi c compiler for secure two-party computations æ," in *Compiler Construction: 23rd International Conference, CC 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014*,vol. 8409, p. 244, Springer, Berlin, Germany, 2014.

[10] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider, "Secure evaluation of private linear branching programs with medical applications," in *Proceedings of the Computer Security–ESORICS 2009: 14th European Symposium on Research in Computer Security*, vol. 14, pp. 424–439, SaintMalo, France, September 2009.

[11] R. Bost, A. P. Raluca, S. Tu, and S. Goldwasser, *Machine Learning Classification over Encrypted Data*, Cryptology ePrint Archive, London, UK, 2014.

[12] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 498–507, Lisbon, Portugal, October 2007.

[13] M. Joye and F. Salehi, "Private yet efficient decision tree evaluation," in *Proceedings of the Data and Applications Security and Privacy XXXII: 32nd Annual IFIP WG 11.3 Conference, DBSec 2018*, vol. 32, pp. 243–259, Bergamo, Italy, July 2018.

[14] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, "Sok: modular and efficient private decision tree evaluation," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 2, pp. 187–208, 2019.

[15] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. M. Sherman, "Privacy-preserving decision trees evaluation via linear functions," in *Proceedings of the Computer Security– ESORICS 2017: 22nd European Symposium on Research in Computer Security*, vol. 22, pp. 494–512, Oslo, Norway, September 2017.

[16] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," *Proceedings*

on Privacy Enhancing Technologies, vol. 2019, no. 1, pp. 266–286, 2019.

[17] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, *Privately Evaluating Decision Trees and Random Forests*, Cryptology ePrint Archive, London, UK, 2015.

[18] G. Bhase and R. S. Mangrulkar, "An access control system using visual cryptography and steganography," in *Proceedings of the 2018 14th international conference on information processing (ICINPRO)*, pp. 1–6, Bengaluru, India, December 2018.

[19] P. V. Chavan and R. S. Mangrulkar, "Encrypting informative color image using color visual cryptography," in *Proceedings of the 2010 3rd International Conference on Emerging Trends in Engineering and Technology*, pp. 277–281, Goa, India, November 2010.

[20] M. De Cock, R. Dowsley, C. Horst et al., "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 217–230, 2019.

[21] W.-J. Lu, J.-J. Zhou, and J. Sakuma, "Noninteractive and output expressive private comparison from homomorphic encryption," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 67–74, Incheon, Korea, June 2018.

[22] C. Chen, J. Zhou, L. Wang et al., "When homomorphic encryption marries secret sharing: secure large-scale sparse logistic regression and applications in risk control," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2652–2662, Washington, DC, USA, August 2021.

[23] P. Mohassel and Y. Zhang, "Secureml: a system for scalable privacy-preserving machine learning," in *Proceedings of the 2017 IEEE symposium on security and privacy (SP)*, pp. 19–38, IEEE, San Jose, CA, USA, May 2017.

[24] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. K. Chameleon, "A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 707–721, Incheon, Korea, October 2018.

[25] N. A. Mohammad and S. Kana, "Private evaluation of a decision tree based on secret sharing," in *International Conference on Information Security and Cryptology*, pp. 171–194, Springer, Berlin, Germany, 2022.

[26] G. Thakur, S. Nayak, and R. Mangrulkar, "Maldexa-a malware detection system using xgboost on amazon web services," in *Proceedings of the 2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, pp. 1–10, IEEE, Delhi, India, October 2021.

[27] E. Boyle, N. Gilboa, and I. Yuval, "Function secret sharing," in *Proceedings of the Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 337–367, Sofia, Bulgaria, April 2015.

[28] E. Boyle, N. Gilboa, and I. Yuval, "Secure computation with pre-processing via function secret sharing," in *Proceedings of the Theory of Cryptography: 17th International Conference, TCC 2019*, vol. 17, pp. 341–371, Nuremberg, Germany, December 2019.

[29] V. Kolesnikov and T. Schneider, "A practical universal circuit construction and secure evaluation of private functions," in *Financial Cryptography*, vol. 5143, pp. 83–97, Springer, Berlin, Germany, 2008.

[30] P. Mohassel, S. Sadeghian, and N. P. Smart, "Actively secure private function evaluation," in *Proceedings of the Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 486–505, Kaoshiung, Taiwan, December 2014.

[31] B. Michael, G. Shafi, and W. Avi, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 351–371, Springer, Berlin, Germany, 2019.

[32] D. Chaum, C. Claude, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 11–19, Kaoshiung, Taiwan, February 1988.

[33] M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 830–842, Vienna, Austria, October 2016.

[34] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of the Advances in Cryptology– CRYPTO 2012: 32nd Annual Cryptology Conference*, pp. 643–662, Santa Barbara, CA, USA, August 2012.

[35] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits," in *Proceedings of the Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security*, vol. 18, pp. 1–18, Egham, UK, September 2013.

[36] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Proceedings of the Information Security and Privacy: 12th Australasian Conference*, vol. 12, pp. 416–430, Townsville, Australia, July 2007.

[37] D. Beaver, "Commodity-based cryptography," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 446–455, New York, NY, USA, December 1997.

[38] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[39] A. Cutler, D. R. Cutler, and J. R. Stevens, *Random forests. Ensemble machine learning: methods and applications*, Springer, Berlin, Germany, 2012.

[40] E. Boyle, N. Chandran, N. Gilboa et al., "Function secret sharing for mixed-mode and fixed-point secure computation," in *Proceedings of the Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 871–900, Zagreb, Croatia, October 2021.

[41] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: low-interaction privacypreserving deep learning via function secret sharing," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 291–316, 2022.

[42] H. Chen, H. Li, Y. Wang, M. Hao, G. Xu, and T. Zhang, "Privdt: an efficient two-party cryptographic framework for vertical decision trees," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1006–1021, 2023.

[43] C.-C. Y. Andrew, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, pp. 162–167, Washington, DC, USA, October 1986.

[44] O. Goldreich, S. Goldwasser, and S. Micali, “How to construct random functions,” *Journal of the ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[45] J. Doerner and A. Shelat, “Scaling oram for secure computation,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 523–535, Dallas, TX, USA, June 2017.

[46] A. Tueno, Y. Boev, and F. Kerschbaum, “Non-interactive private decision tree evaluation,” in *Proceedings of the Data and Applications Security and Privacy XXXIV: 34th Annual IFIP WG 11.3 Conference, DB-Sec 2020*, vol. 34, pp. 174–194, Regensburg, Germany, June 2020.

[47] M. C. Martin, J. D. HaoChen, S. Goldwasser et al., “Homomorphic encryption standard,” *Protecting privacy through homomorphic encryption*, Cryptology ePrint Archive, vol. 62, p. 31, London, UK, 2021.

[48] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, “Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds,” in *Proceedings of the Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security*, pp. 3–33, Hanoi, Vietnam, December 2016.

[49] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe,” in *Proceedings of the Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, pp. 377–408, Hong Kong, China, December 2017.

[50] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Tfhe: fast fully homomorphic encryption over the torus,” *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[51] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, *Tfhe: Fast Fully Homomorphic Encryption Library*, Springer, Berlin, Germany, 2019.

[52] N. P. Smart and F. Vercauteren, “Fully homomorphic simd operations,” *Designs, Codes and Cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[53] E. Boyle, N. Gilboa, and I. Yuval, “Function secret sharing: improvements and extensions,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1292–1303, Vienna, Austria, October 2016.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[55] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, “Ezpc: programmable and efficient secure two-party computation for machine learning,” in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 496–511, Stockholm, Sweden, June 2019.

[56] E. A. Young, J. Tim, and R. S. Engelschall, “Openssl. World wide web,” 2001, http://www.openssl.org/.

[57] E. Cronin, S. Jamin, T. Malkin, and P. McDaniel, “On the performance, feasibility, and use of forward-secure signatures,” in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 131–144, Washington, DC, USA, October 2003.

[58] S. Adams, C. Choudhary, M. De Cock et al., “Privacypreserving training of tree ensembles over continuous data,” *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 2, pp. 205–226, 2022.

[59] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, “Ariann: low-interaction privacypreserving deep learning via function secret sharing,” 2020, https://arxiv.org/abs/2006.04593.