Private Decision Tree Evaluation with Malicious Security via Function Secret Sharing

Jiaxuan Fu, Ke Cheng$^{(\boxtimes)}$, Yuheng Xia, Anxiao Song, Qianxing Li, and Yulong Shen

School of Computer Science and Technology, Xidian University, Xi'an, China
{jiaxuanfu,yuhengxia,songanxiao,qianxing.li}@stu.xidian.edu.cn
chengke@xidian.edu.cn
ylshen@mail.xidian.edu.cn

Abstract. Private Decision Tree Evaluation (PDTE) allows the client to use a decision tree classification model on the server to classify their private data, without revealing the data or the classification results to the server. Recent advancements in PDTE have greatly enhanced its effectiveness in scenarios involving semi-honest security, offering a viable and secure alternative to traditional, less secure approaches for decision tree evaluation. However, this model of semi-honest security may not always align well with real-world problems. In this work, we present FSSTree, a malicious-secure three-party PDTE protocol using function secret sharing (FSS). FSSTree achieves its high performance against malicious adversaries via several innovative cryptographic designs. Especially, 1) we transform a comparison operation into a prefix parity query problem, allowing us to implement malicious-secure comparisons rapidly using lightweight and verifiable FSS. 2) Building upon this, we further propose a constant-round protocol for securely evaluating Conditional Oblivious Selection (COS). 3) We utilize these optimized protocols to enhance the PDTE processes, achieving a considerable decrease in both communication costs and the number of rounds. The experimental results show that FSSTree reduces the runtime in a WAN environment by up to $22.3\times$ times and saves up to $4.1\times$ the communication cost compared to the state-of-the-art work.

Keywords: Private Decision Tree Evaluation · Function Secret Sharing · Malicious Model.

1 Introduction

Decision trees are popular in machine learning for applications like e-commerce recommendations, spam filtering, and medical diagnostics. In a cloud-assisted service, a model provider can deploy a trained decision tree model in the cloud for remote client evaluation. This offers benefits like scalability, ubiquitous access, and cost efficiency. However, outsourcing decision tree evaluation to the cloud raises privacy concerns for both the model and input data. The model must remain confidential, and the model provider should not know the client's input.

Table 1. Comparison of PDTE Protocols

Protocol	Parties	Security	Feature Selection	Conditional Selection	Communication Costs*	Rounds*
Liu et al. [20]	2PC	○	-	HE,MT	$O(2^d\ell)$	$O(d)$
Zheng et al. [29]	2PC	○	MT	MT,SS	$O(2^d n\ell)$	$O(\ell)$
Ma et al. [21]	2PC	◑	OT	GC, OT	$O(dn\ell)$	$O(d)$
Ji et al. [15]	3PC	○	FSS	FSS	$O(d(\log m + \log n + 2\ell))$	$O(d)$
Bai et al. [3]	3PC	●	RSS	RSS, FSS	$O(dn\ell)$	$O(d\ell)$
Ours	3PC	●	RSS	FSS	$O(dn\ell)$	$O(d)$

* Communications and Rounds only statistics online evaluation phase.
○: semi-honest, ◑: one-side malicious, ●: malicious, m: the number of tree nodes, n: the number of features, d: the longest depth of a tree, ℓ: the bit-length of feature values. HE: homomorphic encryption, MT: triple multiplication, SS: secret sharing, GC: garbled circuit, OT: oblivious transfer, RSS: replicated secret sharing, FSS: function secret sharing.

This requires Private Decision Tree Evaluation (PDTE) to ensure no information leakage about the client's input or the model provider's decision tree.

Recently, many PDTE protocols [3, 15, 20, 21, 29] have been proposed with different security and efficiency trade-offs. As shown in Table 1, most existing protocols [15, 20, 29] assume a semi-honest adversary who follows the protocol honestly. Since this assumption is hard to meet in reality, Bai et al. [3] recently propose Mostree to achieve security against malicious adversaries with sublinear communication, forming a three-party architecture where a malicious adversary can compromise one party, which could be the model provider, client, or assistant computing party. However, as an initial attempt, Mostree is not entirely satisfactory and needs performance optimization. The number of communication rounds in the online phase is linearly related to the number of bits ℓ for value representation, causing significant latency. This is because node feature selection in PDTE involves a secure comparison protocol under a malicious model. Mostree implements these comparisons bit-wise on additive shared data using MSB extraction techniques, requiring $O(\ell)$ communication rounds. In a LAN setting, over 85% of the online phase time overhead in Mostree is due to this malicious secure comparison protocol. This overhead is even more pronounced in a WAN setting, reducing the scheme's practicality. To address this, a new PDTE scheme under a malicious model with fewer communication rounds is needed to enhance efficiency and practicality.

1.1   Related Works

Semi-honest PDTE schemes. Most previous works focus on PDTE against semi-honest adversaries. Existing works [4, 14, 20, 27] use homomorphic encryption (HE) for decision tree evaluation, but the high computational and communication costs make it impractical. Tai et al. [25] introduce the concept of path cost to reduce communication overhead without traversing the entire tree. Subsequent works [11, 17, 29] improved computational efficiency further. Ji et

al. [15] achieve efficient decision tree evaluation using Function Secret Sharing (FSS), based on distributed point functions (DPF) and distributed comparison functions (DCF), implementing oblivious transfer (OT) and conditional oblivious transfer (COT) protocols with optimal communication complexity. However, this work only defends against semi-honest adversaries.

Malicious PDTE schemes. Recently, PDTE schemes against malicious adversaries have gained attention. The Works [21,25,27] focus on protecting models from malicious clients. Some works [25, 27] use Zero-Knowledge Proofs (ZKPs) to detect malicious input, while Ma et al. [21] use the malicious-secure Garbled Circuits (GC) protocol [2]. Damgaard et al. [10] propose a PDTE protocol using SPDZ in a dishonest majority setting. The work most closely related to ours is Mostree [3], which is the first PDTE protocol to achieve sublinear communication and malicious security in a three-party honest-majority setting. Mostree uses DPF and replicated secret sharing (RSS) for oblivious selection protocols with sublinear communication costs, combined with lightweight consistency checks for a secure PDTE protocol. However, Mostree mainly optimizes oblivious selection under the malicious model, without improving the time-consuming comparison operations in decision trees, leaving room for performance improvement.

1.2   Our Contributions

This paper proposes FSSTree, a malicious-secure PDTE protocol using function secret sharing (FSS). Our design follows the same architecture of Mostree [3] and also works in the three-party honest-majority setting [1, 12, 19, 22], yet with significant optimizations to achieve largely boosted performance compared to the state-of-the-art work. Our contributions are summarized as follows:

- First, we transform a comparison operation into a prefix parity query problem, and based on this, propose a verifiable comparison protocol (Π_{VCMP}) with plaintext input. The protocol implements malicious-secure comparisons rapidly using lightweight and verifiable FSS, which do not make use of any public key operations or arithmetic MPC.
- Second, by providing the lightweight Π_{VCMP} protocol, we further propose a secure conditional oblivious selection protocol (Π_{COS}) that achieves malicious security with constant communication rounds. We carefully combine the proposed Π_{COS} protocol and consistency check mechanism of replicated secret sharing (RSS), to design FSSTree, achieving a considerable decrease in both communication costs and the number of rounds.
- Third, we formally prove the security of all our protocols using the proof-by-simulation technique [18]. We implement FSSTree and evaluate its performance over various public datasets. Experiments demonstrate that FSSTree can obtain better performance in time and communication overhead. Compared with Mostree, the online runtime over a WAN environment is up to $22.3\times$ better. In the meantime, FSSTree offers up to $4.1\times$ savings in communication costs.

2   Preliminaries

In this section, we initially introduce decision tree evaluation, followed by an introduction to various secret-sharing primitives and their foundational computation protocols. Table 2 lists the main notations utilized throughout our work.

Table 2. Notation Descriptions

Notations	Definitions
$[\![x]\!]$	$\binom{n}{n}$-additive secret shares of the value x
$[\![x]\!]_b$	the $\binom{n}{n}$-shares of x stored in party P_b
$\langle x \rangle$	$\binom{3}{2}$-additive secret shares of the value x
$\langle x \rangle_b = ([\![x]\!]_b, [\![x]\!]_{b+1})$	the $\binom{3}{2}$-shares of x stored in party P_b
ℓ	the bit length of the value x
λ	the security parameter of FSS

2.1   Decision Tree Evaluation

Decision tree evaluation utilizes a binary tree structure for classification, integrating decision nodes and leaf nodes. Each decision node contains a feature attribute and a threshold, guiding the evaluation process. Starting from the root, the feature vector is compared to these thresholds at each node, directing the path to the left child for values that surpass the threshold and to the right for others. This traversal continues until a leaf node is reached, which bears a classification label. This label, generated in the leaf, signifies the classification result of the input feature vector.

2.2   Additive Secret Sharing

Sharing Semantics. Our scheme works on a three-party computation (3PC) model where parties $P = (P_0, P_1, P_2)$ are connected by bidirectional synchronous channels. To further elucidate, we use the following sharing semantics:

$\binom{n}{n}$-**sharing** $[\![x]\!]^n$. A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $[\![\cdot]\!]$-shared among n parties, if party P_b for $b \in \mathbb{Z}_n$ holds $[\![x]\!]_b^n = x_b$ such that $x = \sum_{i=0}^{n-1} x_i$. For the brevity of the description, we omit the corner n when there is no ambiguity. We use $n = 2$ and 3 in this paper.

$\binom{3}{2}$-**sharing** $\langle x \rangle$ (a.k.a. Replicated Secret Sharing, RSS). A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $\langle \cdot \rangle$-shared among P, if party P_b for $b \in \mathbb{Z}_3$ holds $\langle x \rangle_b = ([\![x]\!]_b, [\![x]\!]_{b+1})$.

We extend the above definitions to vectors, where $\vec{x} \in \mathbb{Z}^m$ denotes an m-dimensional vector. Accordingly, we use $[\![\vec{x}]\!]$ and $\langle \vec{x} \rangle$ to denote $\binom{n}{n}$-sharing and $\binom{3}{2}$-sharing of a vector $\vec{x}$, respectively. Without special declaration, all computations are performed in $\mathbb{Z}_{2^\ell}$. For brevity, we omit the notation (mod 2^ℓ). Note that the additive sharing schemes discussed above can be seamlessly expanded to handle Boolean-type data, which we refer to as Boolean sharing.

Semi-honest Protocols. $\binom{3}{2}$-sharing mainly includes the following (semi-honest) computation protocols:

- Reshare: Given $\binom{3}{3}$-shared values $[\![x]\!]$, to obtain $\langle x \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $x'_b = x_b + r_b$, where $r_0 + r_1 + r_2 = 0$ and the auxiliary parameter r_b can be generated offline as [26]. Then, P_b sends x'_b to P_{b-1} and sets $\langle x \rangle_b = (x'_b, x'_{b+1})$.
- Add: Given secret shares $\langle x \rangle$ and $\langle y \rangle$, to obtain $\langle x+y \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $\langle x+y \rangle_b = (x_b + y_b, x_{b+1} + y_{b+1})$. For a public constant c, $\langle x \rangle + c$ can be computed as $(x_0 + c, x_1, x_2)$ where P_b can compute its share locally.
- Mul: Given secret shares $\langle x \rangle$ and $\langle y \rangle$, to obtain $\langle xy \rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $z_b = x_b y_b + x_{b+1} y_b + x_b y_{b+1}$ so that $z = z_0 + z_1 + z_2 = xy$ and z is $\binom{3}{3}$-shared. Then, parties invoke Reshare with inputting $[\![z]\!]$ to get $\langle z \rangle = \langle xy \rangle$. Additionally, for a public constant c, $c \cdot \langle x \rangle$ can be computed as $(c \cdot x_0, c \cdot x_1, c \cdot x_2)$ where P_b can compute its share locally.

Malicious Protocols. Our work uses the following assumed ideal functionalities to achieve malicious security:

- $\mathcal{F}_{\mathsf{mul}}^{[\![\cdot]\!]}$: Given $\binom{2}{2}$-shared $[\![x]\!]$, $[\![y]\!]$, share $[\![x \cdot y]\!]$ between two parties.
- $\mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}$: Given secret shares $\langle x \rangle$, $\langle y \rangle$, share $\langle x \cdot y \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{rand}}$: Sample a random value $r \leftarrow \mathbb{Z}$ and share $\langle r \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{share}}$: Inputting a secret x held by P_b, share $\langle x \rangle$ between three parties.
- $\mathcal{F}_{\mathsf{recon}}$: Given a $\binom{3}{2}$-shared $\langle x \rangle$ and a party index b, reveal x to P_b.
- $\mathcal{F}_{\mathsf{open}}$: Given a $\binom{3}{2}$-shared $\langle x \rangle$, reveal x to all three parties.
- $\mathcal{F}_{\mathsf{os}}$: Given a $\binom{3}{2}$-shared index $\langle i \rangle$ and a $\binom{3}{2}$-shared list $\langle L \rangle$, share $\langle L[i] \rangle$ between three parties.

All these functionalities can be securely computed with malicious security using well-established protocols [3, 9, 12, 19, 28]. Furthermore, whenever a three-party RSS-based secure computation (referred to as 3PC), is used in a black-box manner, we simplify it to the 3PC ideal function $\mathcal{F}_{\mathsf{3PC}}^{\mathbb{Z}}$.

2.3   Function Secret Sharing

A Function Secret Sharing (FSS) scheme [5, 6] for a function family $\mathcal{F}$ splits a function $f(x) \in \mathcal{F}$ into two additive shares $f_0(x)$ and $f_1(x)$, such that each share hides f and ensures that $\forall x \in \mathbb{G}^{in}$, $f_0(x) + f_1(x) = f(x)$.

Definition 1. *(FSS: Syntax [5,6]). A (two-party) FSS scheme is a pair of probabilistic polynomial-time (PPT) algorithms $\{\mathsf{Gen}(\cdot), \mathsf{Eval}(\cdot)\}$ such that:*

- *$\mathsf{Gen}(1^\lambda, f)$ is a key generation algorithm that, given the security parameter λ and the description of a function f, outputs a pair of functional keys (k_0, k_1). Each key is capable of efficiently describing f_0 and f_1 without revealing f.*
- *$\mathsf{Eval}(b, k_b, x)$ is a polynomial-time evaluation algorithm that, given the party index $b \in \{0, 1\}$, the key k_b, and the input x, it outputs an additive share $f_b(x)$, such that $f_0(x) + f_1(x) = f(x)$.*

$(\boldsymbol{k}_0, \boldsymbol{k}_1)$ are called FSS *keys*. The number of bits required to store an FSS key is called the *key size*. In this work, we focus on the Distributed Point Function (DPF), an FSS scheme for point functions that evaluate a non-zero value at exactly one index in their domain. Formally, such a function is defined as follows:

$$f_{\alpha,\beta}^{\bullet}(x) = \begin{cases} \beta, & x = \alpha \\ 0, & otherwise \end{cases} \tag{1}$$

2.4   Verifiable Distributed Point Function (VDPF)

Since the party responsible for generating and distributing DPF keys may be malicious, it may generate incorrect keys. To prevent this, the Verifiable DPF (VDPF) [8] provides an additional verifiable attribute defined as follows:

Definition 2. *(Verifiable DPF). A (two-party) verifiable distributed point function (VDPF) scheme is a set of probabilistic polynomial-time (PPT) algorithms* {*Gen*(·), *BVEval*(·), *Verify*(·)} *such that:*

- *VDPF.Gen*$(1^{\lambda}, f_{\alpha,\beta}^{\bullet}) \rightarrow (k_0, k_1)$ *is a key generation algorithm that, given the security parameter λ and the description of a point function $f_{\alpha,\beta}^{\bullet}$, outputs a pair of functional keys (k_0, k_1).*
- *VDPF.BVEval*$(b, \boldsymbol{k}_b, \{x_i\}_{i \in \{1,...,L\}}) \rightarrow (\llbracket y_b^{(x_i)} \rrbracket_{i \in \{1,...,L\}}, \pi_b)$ *is a polynomial-time evaluation algorithm that, given a batch of L inputs $\{x_i\}_{i \in \{1,...,L\}}$, outputs a tuple of values. The first set of values are the FSS outputs, where $\llbracket y_b^{(x_i)} \rrbracket = f_{\alpha,\beta}^{\bullet}(x_i)$. The second output is a proof π_b that is used to verify the well-formedness of the output.*
- *VDPF.Verify*$(\pi_0, \pi_1) \rightarrow$ (*Accept*/*Reject*) *is an algorithm that, given a pair of proofs π_0 and π_1, outputs either *Accept* or *Reject*. The output should only be *Accept* if $y_0 + y_1$ defines the truth table of some point function, which occurs if it is non-zero in at most one location.*

3   Secure Conditional Selection

In the PDTE, the most crucial step is conditional selection based on feature value f and threshold t. For each decision node, the formal definition of conditional selection is as follows:

$$next = \begin{cases} l, & f \geq t \\ r, & otherwise \end{cases} \tag{2}$$

where l and r represent its left and right child index.

   In most previous schemes, comparing feature values and thresholds usually requires a loop related to the bit length ℓ with multiple rounds of communication. This approach is significantly affected by network latency, causing performance degradation. Therefore, our design aims to reduce the number of communication rounds during protocol execution. In this section, we first propose a verifiable

comparison protocol with plaintext input that implements the comparison of feature values and thresholds. Based on this, we further design a constant-round secure verifiable conditional oblivious selection protocol that hides the access patterns when selecting the child index.

3.1   Verifiable Comparison Protocol with Plaintext Input

We first propose a verifiable comparison protocol (Π_{VCMP}) in the 2PC setting with plaintext input, which will be a key component of the subsequent algorithms. Our strategy for verifiable comparison is to convert the comparison into a prefix parity query and then use the VDPF scheme [8] for verification.

Given a bitstring $x = x_0 x_1 \cdots x_{n-1}$ of length n, assume there is a substring $x_a x_{a+1} \cdots x_{b-1}$ of x, denoted $x[a, b)$, where $0 \leq a < b \leq n$. The *parity query* for $x[a, b)$ is defined as:

$$parity(x[a,b)) := \bigoplus_{i=a}^{b-1} x_i$$

When comparing a public value x with a public threshold t, if $x < t$, x always appears to the left of t on the number line. For a one-hot encoding of x in $\mathbb{Z}_{2^\ell}$, denoted $\hat{\alpha} = \alpha_0 \alpha_1 \cdots \alpha_{\ell-1}$, where $\hat{\alpha}$ is all 0s except for a 1 at α_x, the equation $\alpha_0 \oplus \alpha_1 \oplus \cdots \oplus \alpha_{t-1} = 1$ holds. Thus, we can transform a comparison of plaintext values into a prefix parity query on $\hat{\alpha}$. A recent work [24] proposes a *parity-segment tree* data structure that answers parity queries with complexity $O(\ell)$ and extends this to the DPF scheme, allowing comparison functionality through the DPF scheme.

Inspired by the parity-segment tree and VDPF [8], we propose the verifiable comparison protocol with plaintext input to compute the following function:

$$f_{y,1}^{>}(x) = \begin{cases} 1, & x > y \\ 0, & otherwise \end{cases} \tag{3}$$

We abbreviate the above equation as $1\{x > y\}$. The details of our protocol are shown in Algorithm 1. Note that the variables CW, s, and t in the algorithm follow the original definitions in [6], representing correcting words, output seeds, and disagreeing bits, respectively.

Our protocol requires a pair of verified DPF keys $k_b, b \in \mathbb{Z}_2$, which we generate by invoking $\mathsf{VDPF.Gen}$ and running the DPF key verification protocol in [8]. In the key evaluation phase, our protocol's execution logic is similar to the $\mathsf{DPF.Eval}$ algorithm, with the addition of res_b and d, representing an updated parity value and the current tree traversal direction. The protocol evaluates the verified key following the traversal method in [24] and outputs the value of $f_{y,1}^{>}(x)$ in Boolean-shared form. Using the VDPF key generation and verification process ensures the protocol can detect malformed shared values when the evaluator is semi-honest, as defined below:

Algorithm 1 Verifiable Comparison Protocol with Plaintext Input (Π_{VCMP})

Input: P_b for $b \in \{0, 1\}$ holds a verified DPF key k_b for the point function $f_{y,1}^{\bullet}$ and a public index x. Let $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$.

Output: P_b outputs res_b where $res_0 \oplus res_1 = f_{y,1}^{>}(x)$

P_b for $b \in \{0, 1\}$ does:

Let $x = (x_1, \cdots, x_\ell) \in \{0,1\}^\ell$ be the bit decomposition
Parse $k_b = s^{(0)} || t^{(0)} || CW^{(1)} || \cdots || CW^{(\ell+1)}$
$res_b \leftarrow 0$, $d \leftarrow 0$
for $i \in \{1, ..., \ell\}$ **do**
    Parse $CW^{(i)} = s_{CW} || t_{CW}^L || t_{CW}^R$
    $\tau^{(i)} \leftarrow G(s^{(i-1)}) \oplus (t^{(i-1)} \cdot [s_{CW} || t_{CW}^L || s_{CW} || t_{CW}^R])$
    Parse $\tau^{(i)} = s^L || t^L || s^R || t^R \in \{0,1\}^{2(\lambda+1)}$
    if $x_i == 0$ **then**
        $s^{(i)} \leftarrow s^L, t^{(i)} \leftarrow t^L$
    else
        $s^{(i)} \leftarrow s^R, t^{(i)} \leftarrow t^R$
    end if
    if $d \neq x_i$ **then**
        $d = x_i$
        $res_b = res_b \oplus t^{(i-1)}$
    end if
end for
if $d == 1$ **then**
    $res_b = res_b \oplus t^{(\ell)}$
end if
return res_b

Definition 3. *(Verifiable Comparison Share Integrity, or Security Against Malicious Share Generation). Let k_b be the (possibly maliciously generated) share received by the server P_b. For an adversarially chosen set of inputs $\{x_i\}_{i=1}^\eta$, let $(\{y_b^{(x_i)}\}_{i=1}^\eta, \pi_b) \leftarrow \mathsf{VDPF.BVEval}(b, k_b, \{x_i\}_{i=1}^\eta)$ and $\{z_b^{(x_i)}\}_{i=1}^\eta \leftarrow \Pi_{\mathsf{VCMP}}(b, k_b, \{x_i\}_{i=1}^\eta)$. We say that Π_{VCMP} is secure against malicious share generation if the following holds with all but negligible probability over the adversary's choice of randomness. If $\mathsf{VDPF.Verify}(\pi_0, \pi_1)$ outputs Accept, then the values $z_0^{(x_i)} + z_1^{(x_i)}$ must satisfy:*

$$z_0^{(x_i)} + z_1^{(x_i)} = \begin{cases} 1, & x_i > p \\ 0, & otherwise \end{cases} \tag{4}$$

It should be noted that the result of the above protocol is a bit of $\binom{2}{2}$-Boolean sharing, and further, we need to execute a bit injection $\mathcal{F}_{\mathsf{B2A}}$ to convert it to $\binom{2}{2}$-Additive sharing. Due to space limitations, please refer to the full version [16] for details of the Π_{B2A} protocol.

3.2   Secure Verifiable Conditional Oblivious Selection Protocol

Next, we present a secure verifiable conditional oblivious selection protocol (Π_{cos}) in the 3PC setting, which consists of two parts: the comparison of feature values and thresholds, and the oblivious selection.

Since the comparison protocol given in the previous section requires public input, we first address how to hide the input to preserve privacy, which allows us to use Π_{VCMP} in combination with an additive secret sharing scheme. In general, we can add an offset to the public input x, that is, $x' = x + r$. However, this may lead to an over-ring problem, resulting in incorrect results for Π_{VCMP}. To address this problem, we use the technique from [13, 23] to perform the following transformation:

$$\mathsf{MSB}_\ell(x) = \mathsf{MSB}_\ell(x_0) \oplus \mathsf{MSB}_\ell(x_1) \oplus 1\{y_0 + y_1 \geq 2^{\ell-1}\}$$

where $x = x_0 + x_1$, $y_0 = x_0 \bmod 2^{\ell-1}$, and $y_1 = x_1 \bmod 2^{\ell-1}$, and $\mathsf{MSB}_\ell(x)$ represents the most significant bit extraction for x. The above equation converts $\mathsf{MSB}_\ell(x)$ into extracting the most significant bit of two shared values and a comparison. Note that both y_0 and y_1 are less than $2^{\ell-1}$, so that $y_0 + y_1$ does not lead to any over-ring problems. Using the above equation, we get:

$$\begin{aligned}
1\{f \geq t\} &= \mathsf{MSB}_\ell(\delta = f - t + r) \\
&= \mathsf{MSB}_\ell(\delta) \oplus \mathsf{MSB}_\ell(2^\ell - r) \oplus 1\{y_0 > 2^{\ell-1} - y_1 - 1\} \oplus 1
\end{aligned}$$

where $y_0 = \delta \bmod 2^{\ell-1}$ and $y_1 = (2^\ell - r) \bmod 2^{\ell-1}$. Based on the above equation, we only need to invoke Π_{VCMP} to compute $1\{y_0 > 2^{\ell-1} - y_1 - 1\}$.

Recall that for the $\langle \cdot \rangle$-shared values $\langle l \rangle$ and $\langle r \rangle$ in the 3PC setting. P_b and P_{b+1} hold common share values l_{b+1} and r_{b+1}.

This implies that P_b and P_{b+1} can initially perform a secure comparison of f with t to obtain the result $[\![\theta]\!]$ in $\binom{2}{2}$-sharing form. Next, the parties jointly compute $[\![next]\!]_b^2 = l_{b+1} \cdot [\![\theta]\!] + r_{b+1} \cdot (1 - [\![\theta]\!])$ in an oblivious choice of l_{b+1} or r_{b+1}. This process is repeated for two out of three parties. At the end of the three-round execution, each party holds two $\binom{2}{2}$-shares (since each party executes oblivious selection twice with the other two parties). After that, the parties locally sum up the two $\binom{2}{2}$-shares. In this manner, three parties jointly produce a $\binom{3}{3}$-sharing of $next$, which can be reshared back to an RSS-sharing $\langle next \rangle$. Since the result of the comparison of f and t is the same for each round of execution, the parties will always choose both l or both r, which makes it easy to see that $next$ satisfies Eq. 2. Fig. 1 sketches the above process. The details of the Π_{cos} protocol are shown in Algorithm 2.

Next, we show how our design resists malicious attacks. Recall that in the three-party honest-majority setting, there is at most one malicious party, which may add errors in the execution of the protocol or not execute the protocol as agreed. For our protocol, multiple malicious attacks can occur in different interactive parts, including corrupted DPF keys, incorrect sharing reconstructions, and additive errors present in the resharing phase.

Algorithm 2 Verifiable Conditional Oblivious Selection Protocol (Π_{cos})

[*Offline phase*]:

Upon initialization, the party $P_b, b \in \mathbb{Z}_3$ does:

1: Randomly sample $\alpha_b \leftarrow \mathbb{Z}_{2^\ell}$ and $\alpha_b^0 \leftarrow \mathbb{Z}_{2^\ell}$
2: $\alpha_b^1 = \alpha_b - \alpha_b^0$
3: $x_b = 2^\ell - \alpha_b$
4: $y_b = x_b \bmod 2^{\ell-1}$
5: $(k_0^\bullet, k_1^\bullet) \leftarrow \text{VDPF.Gen}(1^\lambda, 2^{\ell-1} - y_b - 1, 1, \{0,1\})$
6: $c_b = \text{MSB}_n(x_b) \oplus 1$
7: Randomly sample c_0 from $\mathbb{Z}_2$
8: $c_b^1 = c_b \oplus c_b^0$
9: $k_i = c_i || k_i^\bullet$, for $i \in \{0,1\}$.
10: P_b sends $(\alpha_b^0, k_b^0, c_b^0)$ to P_{b+2} and sends $(\alpha_b^1, k_b^1, c_b^1)$ to P_{b+1}.
11: P_{b+1} and P_{b+2} jointly run the DPF key verification protocol from [8] to check the well-formness of DPF keys. If the check fails, abort.
12: P_{b+1} and P_{b+2} expand its DPF key on domain $\mathbb{Z}_{2^\ell}$ to produce a shared vector $[\![\vec{v}]\!]$ in the $\binom{2}{2}$-sharing. Then, P_{b+1} and P_{b+2} jointly compute:
13: $[\![t]\!] = \Sigma_{i \in \mathbb{Z}_{2^\ell}} [\![\vec{v}[i]]\!]$
14: $[\![s]\!] = [\![y]\!]_b - \Sigma_{i \in \mathbb{Z}_{2^\ell}} (i \cdot [\![\vec{v}[i]]\!])$
15: P_{b+1} and P_{b+2} open $[\![t]\!], [\![s]\!]$ then check if $t = 1$ and $s = 0$. If the check fails, abort.

[*Online phase*]:

Upon receiving the shared messages $\langle l \rangle$, $\langle r \rangle$ and the shared conditions $(\langle f \rangle, \langle t \rangle)$, the party $P_b, b \in \mathbb{Z}_3$ does:

16: $\langle \delta \rangle = \langle f \rangle - \langle t \rangle$
17: Set $\langle \alpha_b \rangle = (\alpha_b^0, \alpha_b^1)$, $\langle \alpha_{b+1} \rangle = (0, \alpha_{b+1}^0)$, and $\langle \alpha_{b+2} \rangle = (\alpha_{b+2}^1, 0)$
18: **for** $i \in \{0,1,2\}$ **do**
19:     $\langle \delta_i \rangle = \langle \delta \rangle + \langle \alpha_i \rangle$.
20:     Party P_{i+1} obtains δ_i by invoking $\mathcal{F}_{\text{recon}}(\langle \delta_i \rangle, i+1)$
21:     Party P_{i+2} obtains δ_i by invoking $\mathcal{F}_{\text{recon}}(\langle \delta_i \rangle, i+2)$
22: **end for**
23: **for** $i \in \{0,1\}$ **do**
24:     Parse k_{b+i+1}^i as $c_i || k_i^\bullet$
25:     $\hat{y}_i^b = \delta_{b+i+1} \bmod 2^{\ell-1}$
26:     $res_i^b \leftarrow \Pi_{\text{VCMP}}(k_i^\bullet, \hat{y}_i^b)$
27:     $\theta_i^b = i \cdot \text{MSB}_n(\delta_{b+i+1}) \oplus c_i \oplus res_i^b$
28: **end for**

The party P_b and P_{b+1}, $b \in \mathbb{Z}_3$ do:

29: Obtain $[\![\theta_0^b]\!]$ by invoking Π_{B2A} with P_b input θ_0^b and P_{b+1} input θ_1^{b+1}.
30: $[\![next_0]\!] = [\![l]\!]_{b+1} \cdot [\![\theta_0^b]\!] + [\![r]\!]_{b+1} \cdot (1 - [\![\theta_0^b]\!])$

The party P_b and P_{b+2}, $b \in \mathbb{Z}_3$ do:

31: Obtain $[\![\theta_1^b]\!]$ by invoking Π_{B2A} with P_b inputting θ_1^b and P_{b+2} inputting θ_0^{b+1}.
32: $[\![next_1]\!] = [\![l]\!]_b \cdot [\![\theta_1^b]\!] + [\![r]\!]_b \cdot (1 - [\![\theta_1^b]\!])$

The party $P_b, b \in \mathbb{Z}_3$ does:

33: $[\![next]\!] = [\![next_0]\!] + [\![next_1]\!]$
34: $\langle next \rangle \leftarrow \text{reshare}([\![next]\!])$
35: **return** $\langle next \rangle$
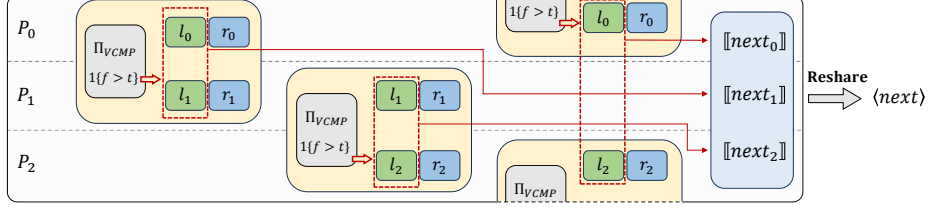
Fig. 1. Our design of conditional oblivious selection.

First, when generating and distributing DPF keys, the key provider P_{b+2} can be corrupted and launch two types of attacks. The corrupted key generator may create an incorrect DPF key that doesn't conform to the protocol, i.e., generate a key for the point function $f_{\alpha,\beta}^{\bullet}$ with $\beta \neq 1$, which disrupts the Π_{VCMP} protocol as it only works correctly with $\beta = 1$. Additionally, the corrupted party may generate the correct DPF key but share an incorrect index $\alpha^* \neq \alpha$ between P_b and P_{b+1}, leading to an incorrect opening of $\delta^{in} = \delta + \alpha + e$ where $e = \alpha^* - \alpha$, causing incorrect conditional selection. Our protocol uses the VDPF scheme [8] to prevent the generation of incorrect DPF keys. This scheme allows key executors to run a verification protocol to check key correctness, ensuring the DPF keys are well-formed and executed accurately. We use the lightweight verification method from [3] to ensure $\beta = 1$. Specifically, during DPF key full-domain verification, a $\binom{2}{2}$-shared vector $[\![v]\!]$ is obtained. By checking if the sum of all elements in $[\![v]\!]$ equals 1, we confirm $\beta = 1$. Additionally, by computing $[\![s]\!] = [\![\alpha]\!]_b - \sum_{i \in \mathbb{Z}_{2^\ell}}(i \cdot [\![\vec{v}[i]]\!])$ and opening $[\![s]\!]$, then verifying if $s = 0$, we ensure the correctness of α corresponding to the point function.

Second, we notice that even if the DPF keys are correctly generated and α is correctly shared, how to correctly reconstruct the public input δ_{b+2} to P_b and P_{b+1} remains a question. If only P_b and P_{b+1} are involved in the reconstruction of δ_{b+2}, it is impossible to ensure correctness because an adversary who corrupts either P_b or P_{b+1} can always add errors during the reconstruction process. Fortunately, for the reconstruction of δ_{b+2}, the parties can define a consistent $\langle\cdot\rangle$-sharing of $\langle\alpha\rangle$ from $[\![\alpha]\!] = ([\![\alpha]\!]_0, [\![\alpha]\!]_1)$ as:

$$\langle\alpha\rangle = (([\![\alpha]\!]_0, [\![\alpha]\!]_1), ([\![\alpha]\!]_1, 0), (0, [\![\alpha]\!]_0))$$

Therefore, the parties can reconstruct the offset input $\langle\delta_{b+2}\rangle = \langle\delta\rangle + \langle\alpha\rangle$ through $\mathcal{F}_{\mathsf{recon}}$, which ensures consistency.

Finally, a corrupted party may maliciously add errors before resharing the $\binom{3}{3}$-sharing selected secret, resulting in an additive attack. All the above attacks only compromise correctness, not privacy. For the first two attacks, we have implemented a well-formed consistency verification. So far, our protocol is only limited to adding errors in the re-sharing phase, which can be verified by pre-computing SPDZ-like MAC values. The details are described in Section 4.

4   The Proposed FSSTree Scheme

In this section, we present the FSSTree scheme, where the decision tree encoding is similar to previous work [15]. For a tree with depth d and number of nodes m, we follow the encoding from [3] to hide data access patterns. During privacy-preserving evaluation, once a leaf node is reached, the server repeatedly accesses itself until the protocol reaches d steps, treating each leaf node as a decision node with left and right children pointing to itself. This design eliminates the need to fill the tree into a full binary tree with m nodes, saving storage. We encode the decision tree as a multi-dimensional array $\vec{T}$, where each element represents a node. The i-th node contains five values: left child index l, right child index r, feature selection vector $\vec{f}$, threshold value t, and classification value c, denoted as $T[i] \leftarrow l||r||\vec{f}||t||c$. Note that we use a selection vector (for index k, its selection vector $\vec{k}$ comprises all 0s except for a single 1 at k, this vector is also widely used in previous work [3, 17]) to encode the feature value index, since the input features are generally not excessive, which is relevant to reduce subsequent computational overhead.

The model provider performs this encoding as it is lightweight and negligible in cost. For a decision node, the classification value is set randomly. For a leaf node, the left/right child indexes point to itself, and the threshold value is random. This encoding ensures correct evaluation since there is only one leaf node per path, and the decision tree must end at a leaf node.

To address errors in the Π_{cos} protocol used in FSSTree, we use SPDZ-like MACs. We pre-compute the MAC values of tree contents $\vec{T}$ during setup to verify consistency in Π_{cos}. Any error affects the relationship between data and its MAC, detectable through the Batch MAC Check protocol Π_{MacCheck} [3].

Algorithm 3 shows the details of the FSSTree scheme. During the setup phase, the model provider encodes the decision tree T and shares it between parties. Upon receiving $\vec{T}$, the parties jointly compute the MAC values for each node in the decision tree $\vec{T}$ and store them in $\vec{M}$. At this point, the parties are ready to proceed with decision tree inference.

In the inference phase, our algorithm consists of three stages: feature selection, node evaluation, and tree traversal. For the feature selection, we start from the root node and use the selection vector $\vec{f}$ to obtain the required feature value x for the current node. For the node evaluation, by invoking the Π_{cos} protocol, we determine the next direction of traversal based on the comparison result between the feature value x and the threshold t. It is important to note that we simultaneously select both the child selection vector and its MAC value, resulting in a pair of selection outcomes $(\vec{\eta}, \vec{\eta}_m)$. The child selection vector $\vec{\eta}$ is used to choose the next node, and we repeat the above steps to achieve tree traversal. At the end of the entire loop, by calling $\mathcal{F}_{\text{MacCheck}}$, we verify all pairs of selections during the traversal. If there is any MAC mismatch, the protocol aborts. Finally, only the client can obtain the final result res through $\mathcal{F}_{\text{recon}}$.

Security. We formally prove the security of all our protocols using the proof-by-simulation technique, the details can be found in Appendix A.

Algorithm 3 The FSSTree Scheme (Π_{FSSTree})

Input: A shared tree model $\langle \vec{T} \rangle$, an input feature vector $\langle \vec{F} \rangle$
Output: The RSS sharing result $\langle res \rangle$

    [**Setup**]:
1: The model provider encodes the decision tree T as a array $\vec{T}$. Then the parties jointly invoke $\mathcal{F}_{\mathsf{share}}$ in which the model provider inputs $\vec{T}$.
2: The parties jointly invoke $\mathcal{F}_{\mathsf{rand}}$ to get a shared MAC key $\langle \alpha \rangle$, where $\alpha \in \mathbb{Z}_{2^\ell}$.
3: **for** $i = 1$ to m **do**
4:     Compute MACs: $\langle \vec{M}[i] \rangle = \mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}(\langle \alpha \rangle, \langle \vec{T}[i] \rangle)$
5: **end for**
6: The parties run $\Pi_{\mathsf{MacCheck}}(\langle \vec{T} \rangle, \langle \vec{M} \rangle)$. Abort if the check fails.
7: Output $\langle \vec{T} \rangle, \langle \vec{M} \rangle$.

    [**Inference**]:
    Upon receiving $\langle \vec{F} \rangle$ from the client, the party $P_b, b \in \mathbb{Z}_3$ does:
8: Parties jointly initialize $\langle res \rangle = \langle 0 \rangle$
9: Parse $\langle l \rangle || \langle r || \langle \vec{f} \rangle || \langle t \rangle || \langle c \rangle \leftarrow \langle \vec{T} \rangle[0]$
10: Parse $\langle l_m \rangle || \langle r_m \rangle || \langle \vec{f}_m \rangle || \langle t_m \rangle || \langle c_m \rangle \leftarrow \langle \vec{M} \rangle[0]$
11: **for** $i = 1$ to d **do**
12:     $\langle x \rangle = \sum_{j \in [n]} \langle \vec{F}[j] \rangle \cdot \langle \vec{f}[j] \rangle$
13:     $l_{pack} = \langle l \rangle || \langle l_m \rangle$
14:     $r_{pack} = \langle r \rangle || \langle r_m \rangle$
15:     $\langle \eta \rangle || \langle \eta_m \rangle = \Pi_{\mathsf{cos}}((l_{pack}, r_{pack}), (\langle x \rangle, \langle t \rangle))$
16:     $\langle \vec{T}[\eta] \rangle || \langle \vec{M}[\eta] \rangle = \mathcal{F}_{\mathsf{os}}(\langle \eta \rangle, \langle \vec{T} \rangle || \langle \vec{M} \rangle)$
17:     Parse $\langle l \rangle || \langle r \rangle || \langle \vec{f} \rangle || \langle t \rangle || \langle c \rangle \leftarrow \langle \vec{T}[\eta] \rangle$
18:     Parse $\langle l_m \rangle || \langle r_m \rangle || \langle \vec{f}_m \rangle || \langle t_m \rangle || \langle c_m \rangle \leftarrow \langle \vec{M} \rangle[\eta]$
19:     $\langle res \rangle = \langle c \rangle$
20: **end for**
21: Use Π_{MacCheck} to check each pair of $\langle \eta \rangle || \langle \eta_m \rangle$ from Π_{cos} and $\langle \vec{T}[\eta] \rangle || \langle \vec{M}[\eta] \rangle$ from $\mathcal{F}_{\mathsf{os}}$. Abort if the check fails.
22: Invoke $\mathcal{F}_{\mathsf{recon}}$ to reconstruct res to the client.

---

5 Experiment

5.1 Setup

Our scheme is implemented with Pytorch based on the NssMPClib[1]. Experiments are conducted on a server running Ubuntu 20.04 with an Intel Core i9-10900k @ 3.7GHz processor, 128GB RAM, and an NVIDIA GeForce RTX 3090. We use the Linux *tc* tool to simulate LAN (RTT: 0.1 ms, 1 Gbps), MAN (RTT: 6 ms, 100 Mbps), and WAN (RTT: 80 ms, 40 Mbps) networks. FSSTree operates in the secret sharing domain, with data represented as integers over the ring $\mathbb{Z}_{2^{32}}$ and a security parameter $\lambda = 128$. We use public data sets from the UCI repository as listed in Table 4. We scale rational numbers to integer representation with a factor of 10^6 for the computations.

---
[1] https://github.com/XidianNSS/NssMPClib

Table 4. Paramters of Public Datasets

Dataset	Depth d	Features n	#(Nodes) m
wine	5	7	23
breast	7	12	43
digits	15	47	337
spambase	17	57	171
diabetes	28	10	787
Boston	30	13	851
MNIST	20	784	4179

Table 5. Runtimes(s) Comparision of Conditional Oblivious Selection Protocol

Input Size	Scheme	LAN			WAN		
		CPU	GPU	A.R.	CPU	GPU	A.R.
10^2	ASS-based [3]	0.16	0.36	**0.45**	64.08	64.32	**0.99**
	Ours	**0.03**	**0.07**	0.44	**3.24**	**3.35**	0.97
10^4	ASS-based [3]	**0.60**	0.67	0.89	74.01	74.36	1.00
	Ours	3.55	**0.08**	**43.59**	**5.16**	**3.18**	**1.62**
10^6	ASS-based [3]	60.06	51.04	1.18	1290.82	1300.81	0.99
	Ours	**44.84**	**3.22**	**13.91**	**102.80**	**72.17**	**1.42**

* A.R. represents the accelerative ratio.

5.2   Evaluation of Verifiable Conditional Oblivious Selection

Since our work focuses on optimizing conditional oblivious selection, we first benchmark the performance of Π_{COS} as the building blocks used for FSSTree. We perform experiments in different network settings with different input sizes and compare them with the ASS-based protocol used in the state-of-the-art [3] to demonstrate the superiority of our protocol. Since the protocol provided in [3] cannot support GPU acceleration, we re-implement the conditional oblivious selection performed in their protocol based on their open-source code (consisting of one verifiable comparison and one verifiable dot multiplication) using our experimental environment to make a fair comparison.

Table 5 shows the detailed results of the comparison. In most cases, our protocol performs better. Only in the LAN setting is our protocol slower than ASS-based protocols, especially when the input size is 10^4. This is because our protocol is based on FSS, which has better communication overhead, but also higher computation overhead compared to the ASS-based protocol, and thus does not have an advantage when computing small batches of data in a good network environment. When the network environment is poor or the input size is large, our protocol performs significantly better than the ASS-based protocol. On the other hand, when the computation is accelerated using GPUs, all computations need to be performed on the Video RAM. However, the data exchanged among parties first needs to be transmitted to the CPU RAM, causing substantial IO overhead and thus impacting the performance of the protocol. Since our protocol has constant communication rounds, it is more suitable for

acceleration using GPUs. From the experimental results, it can be seen that the ASS-based protocol cannot be accelerated by GPUs in all network settings, while our protocol can be accelerated by GPUs even in the worst network settings.

5.3   Evaluation of FSSTree Scheme

We evaluate the FSSTree scheme using different public datasets and compare it with the state-of-the-art work Mostree. Compared to Mostree, during the key generation phase, we need to perform an additional key generation and full domain evaluation for DPF keys in the conditional oblivious selection operation, which takes 5 minutes for one FSS key. Since the number of comparisons of the PDTE is only related to the depth of the decision tree and only one value is used for comparison at a time, we consider the additional overhead in this phase to be acceptable. Furthermore, our decision tree encoding and sharing phase is the same as in Mostree. Next, we focus on the performance of the online decision tree evaluation phase.

Table 6 illustrates the runtime and communication costs for the online evaluation phase between Mostree and our scheme in the LAN, MAN, and WAN network settings. Our scheme demonstrates a lower runtime across the various network settings than Mostree, particularly in the WAN setting, where our work improves the evaluation runtime by approximately $1.5 - 22.3\times$ over Mostree in different datasets. Regarding the MNIST and digits datasets, in the LAN setting, our work has only a $1.5\times$ improvement compared to Mostree, as these two datasets have more features and nodes, leading to plenty of multiplication operations. However, when the network environment becomes worse, e.g., in MAN and WAN settings, our work achieves $11.2 - 22.3\times$ improvement. This is because the feature comparison in Mostree requires a large number of interactions and is thus greatly affected by communication latency. In contrast, our work has constant communication rounds, and its runtime mainly depends on the depth of the decision tree, making it less susceptible to communication delays.

Regarding communication overhead, since our Π_{COS} protocol features significantly lower communication rounds and costs, and the main overhead of PDTE stems from conditional selection, FSSTree offers better communication efficiency compared to Mostree, which offers up to $4.1\times$ savings in communication costs.

Table 6. Comparison of Runtime (s) and Communication Cost (KB) with Mostree

| Datasets | Runtime | | | | | | Comm. Costs | |
| | LAN | | MAN | | WAN | | | |
	Mostree	Ours	Mostree	Ours	Mostree	Ours	Mostree	Ours
wine	0.56	0.05	25.65	1.34	317.79	14.54	22.82	6.32
breast	0.68	0.06	35.96	1.83	443.38	19.87	31.45	8.36
digits	3.72	2.01	77.94	4.69	954.75	41.81	104.32	54.84
spambase	2.01	0.17	87.14	4.33	1081.71	46.72	130.19	74.11
diabetes	3.07	0.27	143.85	7.12	1782.19	76.18	121.89	29.53
Boston	3.42	0.31	154.11	7.59	1909.18	81.56	136.93	37.96
MNIST	6.10	3.88	106.15	9.43	1281.20	56.05	1175.51	1109.53

Table 7. Comparison of Runtime(s) with Mostree on CPU and GPU

Dataset	Scheme	LAN			MAN			WAN		
		CPU	GPU	A.R.	CPU	GPU	A.R.	CPU	GPU	A.R.
digits	Mostree	3.72	3.38	1.10	77.94	78.89	0.99	954.75	960.60	0.99
	Ours	2.01	0.69	2.91	4.69	3.98	1.18	41.82	41.54	1.01
MNIST	Mostree	6.10	4.63	1.32	106.15	105.38	1.01	1281.20	1283.30	1.00
	Ours	3.88	1.04	3.73	9.43	5.46	1.73	56.05	55.33	1.01

For digits and MNIST datasets, we accelerated their computation by the GPUs, with the results illustrated in Table 7. Our work achieves an accelerative ratio of $2.9 - 3.5\times$ in the LAN setting, while Mostree is only $1.1 - 1.3\times$. Whereas in the MAN setting, our scheme can still be accelerated using the GPUs, Mostree suffers from a communication bottleneck and even runs slower using the GPUs. In the WAN setting, neither scheme will have acceleration using GPUs, and the bottleneck at this point comes entirely from communication. However, our scheme has better performance in this setting.

6   Conclusion

In this paper, we propose FSSTree, a secure three-party PDTE protocol under the malicious model, which enables clients and model providers to evaluate decision trees without revealing both clients' inputs and model privacy. Technically, we design a verifiable comparison protocol with plaintext input, and based on this, propose a constant-round protocol for securely evaluating conditional oblivious selection. This serves as the foundational bedrock for FSSTree, which enhances the PDTE processes to achieve a considerable decrease in both runtime and communication costs. Extensive experiments demonstrated that compared with the state-of-the-art, FSSTree achieves up to $23.5\times$ better online runtime over a WAN setting, as well as enjoys up to $4.1\times$ savings in communication cost.

A   Security Proofs

We first prove the security of Verifiable Comparison Protocol Π_{VCMP}, then we employ the universal composition (UC) theory [7] to prove that the other protocols satisfy the following definition.

Definition 4. *(Three-party Secure Computation [3]). Let $\mathcal{F}$ be the three-party functionality. A protocol Π securely computes $\mathcal{F}$ with abort in the presence of one malicious party, if for every party P_b corrupted by a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real world there exists a PPT simulator S in the ideal world with F, such that*

$$\{IDEAL_{\mathcal{F},\mathcal{S}(z),b(x_0,x_1,x_2,n)}\} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A}(z),b(x_0,x_1,x_2,n)}\}$$

where $x_b \in \{0,1\}^*$ is the input provided by P_b for $b \in \mathbb{Z}_3$, and $z \in \{0,1\}^*$ is the auxiliary information that includes the public input length information $\{|x_b|\}_{b\in\mathbb{Z}_3}$. The protocol Π securely computes $\mathcal{F}$ with abort in the presence of one malicious party with statistical error $2^{-\lambda}$ if there exists a negligible function $\mu(\cdot)$ such that the distinguishing probability of the adversary is less than $2^{-\lambda} + \mu(\kappa)$.

A.1   Security Proof of Verifiable Comparison Protocol Π_{VCMP}

We prove that the verifiable comparison protocol Π_{VCMP} in Section 3.1 is secure by the following lemma.

Lemma 1. *(Detection of Malicious Function Shares). The verifiable plaintext comparison protocol satisfies Definition 3.*

Proof. We prove this lemma by contradiction. According to Lemma 2 and Lemma 3 in [8], when VDPF.Verify(π_0, π_1) outputs Accept, then the values $y_0^{(x_i)} + y_1^{(x_i)}$ must be a subset of the truth table of the point function $f_{p,1}^\bullet$. Suppose there exists an input $u \in \{x_i\}_{i=1}^{\eta}$ such that u does not satisfy Eq. 4. We categorize the discussion based on the values of u, when $u > p$, based on the assumption, we have $z_0^{(u)} + z_1^{(u)} = 0$. However, according to the definition of Π_{VCMP}, we know that $z_0^{(u)} + z_1^{(u)} = parity(p[0,u))$, since $u > p$, it follows that $parity(p[0,u)) = z_0^{(u)} + z_1^{(u)} = 1$. This contradicts the assumption that $z_0^{(u)} + z_1^{(u)} = 0$. For the case where $u \le p$, we can arrive at a similar conclusion, therefore the verifiable plaintext comparison protocol satisfies Definition 3.

A.2   Security Proof of Verifiable Conditional Oblivious Selection Protocol Π_{cos}

Theorem 1. *Π_{cos} securely compute $\mathcal{F}_{\mathsf{cos}}$ in the $\{\mathcal{F}_{\mathsf{recon}}, \mathcal{F}_{\mathsf{mul}}^{[\![\cdot]\!]}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

Proof. For any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ that can simulate the adversary's view with accessing the functionality $\mathcal{F}_{\sf cos}$. In the cases where $\mathcal{S}$ aborts or terminates the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Simulating offline phase. When the corrupted party P_b is the DPF key generator, $\mathcal{S}$ acts as honest parties P_{b+1} and P_{b+2} and receives a pair of DPF keys from $\mathcal{A}$. $\mathcal{S}$ checks the keys and aborts if they are incorrect. If an honest party generates the keys, $\mathcal{S}$ creates a pair of VDPF keys and samples a random share $r_b \leftarrow \mathbb{Z}_{2^\ell}$, then sends (k_b, r_b) to $\mathcal{A}$. $\mathcal{S}$ runs the VDPF verification protocol with $\mathcal{A}$ and aborts if $\mathcal{A}$ aborts in the verification. All other messages from honest parties to the corrupted one are randomly sampled by the simulator.

The above simulation is indistinguishable from real-world execution. If $\mathcal{A}$ sends incorrect DPF keys, the real-world execution would abort due to VDPF check. In the ideal world, $\mathcal{S}$ can directly check the keys, causing the ideal world to abort with indistinguishable probability. When the corrupted party is the DPF evaluator, $\mathcal{S}$ honestly generates the DPF keys. If $\mathcal{A}$ follows the VDPF key verification protocol, the check will pass; otherwise, it will abort. Thus, $\mathcal{S}$ runs the check protocol with $\mathcal{A}$ as in the real-world protocol, always aborting with indistinguishable probability.

Simulating online phase. If the protocol does not abort after the simulation stage, all DPF keys generated by $\mathcal{A}$ in $\mathcal{S}$ are correct. The only issue in the online phase is whether $\mathcal{A}$ follows the protocol honestly and whether $\mathcal{S}$ successfully extracts the error term. We assume that the shared messages $\langle l \rangle$, $\langle r \rangle$ and conditions $(\langle f \rangle, \langle t \rangle)$ are correctly shared/simulated initially, meaning the honest parties hold correct and consistent RSS shares.

For all local computations, $\mathcal{S}$ is easy to simulate. $\mathcal{S}$ only needs to simulate interactions between honest and corrupted parties and abort with indistinguishable probability. Only a few parts in the selection phase need interaction. First, securely reconstructing δ: since $\langle \delta \rangle = \langle f \rangle - \langle t \rangle + \langle \alpha \rangle$ and $\langle f \rangle$, $\langle t \rangle$, and $\langle \alpha \rangle$ are correct RSS sharings, $\langle \delta \rangle$ is consistent. $\mathcal{S}$ checks if $\mathcal{A}$ sends the correct value using P_{b+2}'s share: 1) if $\mathcal{A}$ sends an incorrect share, $\mathcal{S}$ aborts; 2) if correct, $\mathcal{S}$ samples a random $\delta \in \mathbb{Z}_{2^\ell}$, computes shares for P_{b+1} and P_{b+2} from δ and P_b's RSS shares, and sends them to $\mathcal{A}$. The parties then open δ to $\mathcal{A}$. Second, from $\mathcal{F}_{\sf B2A}$, which uses $\mathcal{F}_{\sf mul}^{[\![\cdot]\!]}$ with security proved in [28]. Lastly, re-sharing $[\![\text{next}]\!]$ from $\binom{3}{3}$-sharing to RSS-sharing. $\mathcal{A}$ can add an error, and $\mathcal{S}$ extracts it: $\mathcal{S}$ gets next_i^* from the corrupted party to an honest party and updates P_{b+1}'s share. $\mathcal{S}$ computes the correct next_i (knowing all shares to compute z_i), then computes $res \leftarrow \text{next}_i - \text{next}_i^*$ and sends res to $\mathcal{F}_{\sf cos}$, completing the simulation.

Note that in simulating the open step of δ, δ uses shares of honest parties, either simulated or computed from local data. These shares match those of the corrupted party, making the simulation indistinguishable from real execution. For the re-sharing phase, the simulation randomly samples P_{b+2}'s share and sends it to $\mathcal{A}$. In the real protocol execution, this is generated by a PRF F, so the simulation remains computationally indistinguishable due to the PRF's security. Overall, the simulation is indistinguishable from real execution.

A.3 Security Proof of the FSSTree Evaluation Protocols Π_{FSSTree}

Theorem 2. *Π_{FSSTree} securely compute $\mathcal{F}_{\mathsf{FSSTree}}$ in the $\{\mathcal{F}_{recon}, \mathcal{F}_{os}, \mathcal{F}_{coin}, \mathcal{F}_{rand},$ $\mathcal{F}_{open}, \mathcal{F}_{cos}, \mathcal{F}_{mul}^{\langle \cdot \rangle}, \mathcal{F}_{\mathsf{CheckZero}}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

Proof. For any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ that simulates the adversary's view using the functionality $\mathcal{F}_{\mathsf{FSSTree}}$. If $\mathcal{S}$ aborts or terminates the simulation, it outputs whatever $\mathcal{A}$ outputs.

Simulating setup. $\mathcal{S}$ samples random shares for $\vec{T}$ and gives them to $\mathcal{A}$. It also samples shares for honest parties for later simulations. $\mathcal{S}$ acts as $\mathcal{F}_{rand}$, using its simulator for simulation. For $i \in \{1, ..., m\}$, $\mathcal{S}$ acts as $\mathcal{F}_{mul}^{\langle \cdot \rangle}$ and receives α_i from the adversary. From α_i, $\mathcal{S}$ adds the error into the share of MAC value $\langle \sigma(\vec{T}[i]) \rangle$ for honest party P_{b+1}. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{CheckZero}}$ in Π_{MacCheck}. If any $i \in \{1, ..., m\}$ has the simulation aborts from $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{CheckZero}}$ in Π_{MacCheck}, $\mathcal{S}$ aborts.

The above simulation is indistinguishable from real protocol execution. Using a hybrid model, $\mathcal{S}$ applies existing simulation strategies for $\mathcal{F}_{\mathsf{rand}}$, $\mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}$, and $\mathcal{F}_{\mathsf{CheckZero}}$. $\mathcal{S}$ can extract $\mathcal{A}$'s strategy from the error term in $\mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}$ and abort if necessary. The MAC is over $\mathbb{F}_{2^\ell}$ and any additive error will fail the MAC check except with probability $\frac{1}{2^\ell - 1}$. Thus, the simulation is indistinguishable from real execution, with a statistical error of $\frac{1}{2^\ell - 1}$.

Simulating inference. $\mathcal{S}$ acts as $\mathcal{F}_{\mathsf{mul}}^{\langle \cdot \rangle}$ and uses $\mathcal{F}_{\mathsf{cos}}$'s simulator for inner product and comparison. Any non-zero error in $\mathcal{F}_{\mathsf{cos}}$ causes $\mathcal{S}$ to abort. For oblivious selection, $\mathcal{S}$ uses $\mathcal{F}_{\mathsf{os}}$ from [3], making this part indistinguishable from real execution. $\mathcal{S}$ aborts if $\mathcal{A}$ sends any non-zero error. Overall, the simulation is indistinguishable from real execution, with a statistical error of $\frac{1}{2^\ell - 1}$.

B Verifiable B2A Extraction

The Π_{B2A} extraction protocol for 1 bit $\binom{2}{2}$-boolean sharing is shown in Algorithm 4, which requires performing a 2PC $\binom{2}{2}$-additive sharing multiplication. We use the well-established verifiable multiplication protocol proposed in [28].

Algorithm 4 B2A Extraction Π_{B2A}

Input: P_b for $b \in \{0, 1\}$ holds a boolean-shared data $_2[\![x]\!]$
Output: Additive share $[\![x]\!]$
1: P_0 set two variables $[\![a]\!]_0 =_2 [\![x]\!]_0, [\![b]\!]_1 = 0$.
2: P_1 set two variables $[\![a]\!]_1 = 0, [\![b]\!]_1 =_2 [\![x]\!]_1$.
3: P_0 and P_1 set $[\![x]\!] = [\![a]\!] + [\![b]\!] - 2 \cdot \mathcal{F}_{\mathsf{mul}}^{[\![\cdot]\!]}([\![a]\!], [\![b]\!])$.
4: **return** $[\![x]\!]$

References

1. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 805–817 (2016)
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 535–548 (2013)
3. Bai, J., Song, X., Zhang, X., Wang, Q., Cui, S., Chang, E.C., Russello, G.: Mostree: Malicious secure private decision tree evaluation with sublinear communication. In: Proceedings of the 39th Annual Computer Security Applications Conference. pp. 799–813 (2023)
4. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS Symposium 2015. pp. 1–14. Internet Society (2015)
5. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367. Springer (2015)
6. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303 (2016)
7. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of CRYPTOLOGY **13**, 143–202 (2000)
8. de Castro, L., Polychroniadou, A.: Lightweight, maliciously secure verifiable function secret sharing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 150–179. Springer (2022)
9. Chida, K., Hamada, K., Ikarashi, D., Kikuchi, R., Genkin, D., Lindell, Y., Nof, A.: Fast large-scale honest-majority mpc for malicious adversaries. Journal of Cryptology **36**(3), 15 (2023)
10. Damgård, I., Escudero, D., Frederiksen, T., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure mpc over rings with applications to private machine learning. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1102–1120. IEEE (2019)
11. De Cock, M., Dowsley, R., Horst, C., Katti, R., Nascimento, A.C., Poon, W.S., Truex, S.: 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 **16**(2), 217–230 (2017)
12. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 225–255. Springer (2017)
13. Gupta, K., Jawalkar, N., Mukherjee, A., Chandran, N., Gupta, D., Panwar, A., Sharma, R.: Sigma: secure gpt inference with function secret sharing. Cryptology ePrint Archive (2023)
14. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Theory of Cryptography Conference. pp. 575–594. Springer (2007)
15. Ji, K., Zhang, B., Lu, T., Li, L., Ren, K.: Uc secure private branching program and decision tree evaluation. IEEE Transactions on Dependable and Secure Computing **20**(4), 2836–2848 (2023)

16. Jiaxuan Fu, Ke Cheng, Y.X.A.S.Q.L., Shen, Y.: Private Decision Tree Evaluation with Malicious Security via Function Secret Sharing. Tech. rep., Xidian University (2024), https://github.com/BlackFH/FSSTree/blob/main/FSSTree-full.pdf
17. Kiss, Á., Naderpour, M., Liu, J., Asokan, N., Schneider, T.: Sok: Modular and efficient private decision tree evaluation. Proceedings on Privacy Enhancing Technologies (2), 187–208 (2019)
18. Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346. Springer (2017)
19. Lindell, Y., Nof, A.: A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 259–276 (2017)
20. Liu, L., Su, J., Chen, R., Chen, J., Sun, G., Li, J.: Secure and fast decision tree evaluation on outsourced cloud data. In: Machine Learning for Cyber Security: Second International Conference, ML4CS 2019, Xi'an, China, September 19-21, 2019, Proceedings 2. pp. 361–377. Springer (2019)
21. Ma, J.P., Tai, R.K., Zhao, Y., Chow, S.S.: Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In: NDSS (2021)
22. Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
23. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)
24. Storrier, K., Vadapalli, A., Lyons, A., Henry, R.: Grotto: Screaming fast (2+ 1)-pc or 2n via (2, 2)-dpfs. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 2143–2157 (2023)
25. Tai, R.K., Ma, J.P., Zhao, Y., Chow, S.S.: Privacy-preserving decision trees evaluation via linear functions. In: 22th European Symposium on Research in Computer Security (ESORICS 2017). pp. 494–512. Springer (2017)
26. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies **2021**(1), 188–208
27. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. Proceedings on Privacy Enhancing Technologies **2016**(4), 335–355
28. Xu, G., Han, X., Deng, G., Zhang, T., Xu, S., Ning, J., Yang, A., Li, H.: Verifyml: Obliviously checking model fairness resilient to malicious model holder. IEEE Transactions on Dependable and Secure Computing (2023)
29. Zheng, Y., Duan, H., Wang, C.: Towards secure and efficient outsourcing of machine learning classification. In: 24th European Symposium on Research in Computer Security (ESORICS 2019). pp. 22–40. Springer (2019)