Scalable Private Decision Tree Evaluation with Sublinear Communication

Jianli Bai
University of Auckland
Auckland, New Zealand
jbai795@aucklanduni.ac.nz

Xiangfu Song
National University of Singapore
Singapore
songxf@comp.nus.edu.sg

Shujie Cui
Monash University
Melbourne, Australia
shujie.cui@monash.edu

Ee-Chien Chang
National University of Singapore
Singapore
changec@comp.nus.edu.sg

Giovanni Russello
University of Auckland
Auckland, New Zealand
g.russello@auckland.ac.nz

ABSTRACT

Private decision tree evaluation (PDTE) allows a decision tree holder to run a secure protocol with a feature provider. By running the protocol, the feature provider will learn a classification result. Nothing more is revealed to either party. In most existing PDTE protocols, the required communication grows exponentially with the tree's depth d, which is highly inefficient for large trees. This shortcoming motivated us to design a sublinear PDTE protocol with $O(d)$ communication complexity. The core of our construction is a shared oblivious selection (SOS) functionality, allowing two parties to perform a secret-shared oblivious read operation from an array. We provide two SOS protocols, both of which achieve sublinear communication and propose optimizations to further improve their efficiency. Our sublinear PDTE protocol is based on the proposed SOS functionality and we prove its security under a semi-honest adversary. We compare our protocol with the state-of-the-art, in terms of communication and computation, under various network settings. The performance evaluation shows that our protocol is practical and more scalable over large trees than existing solutions.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Classification and regression trees**.

KEYWORDS

decision tree, secure computation, sublinear communication

ACM Reference Format:
Jianli Bai, Xiangfu Song, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2022. Scalable Private Decision Tree Evaluation with Sublinear Communication. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3517413

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan.

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3517413

1 INTRODUCTION

Decision trees are popular machine learning techniques for data classification. Due to their effectiveness and simplicity, decision trees have been widely adopted in various applications, such as spam filtering [8], credit risk assessment [22] and disease diagnosis [31]. Typically, there are two parties: a tree holder holding a tree model; and a feature provider holding a feature vector that needs to be classified. However, performing the evaluation processed in such a two-party setting can lead to privacy issues. On the one hand, if the feature vectors is sent in plaintext to the model provider it might reveal individuals' information that are privacy sensitive. This might be the case in healthcare and credit risk assessment applications. On the other hand, the model is a valuable asset for the model provider. If freely accessible, it may leak sensitive information about the training data.

Private Decision Tree Evaluation (PDTE) protocols [21, 32, 35] address the above privacy issues. A PDTE protocol enables two reciprocal-distrustful parties to collaboratively perform the tree evaluation without revealing any sensitive information to each other.

There are several crucial aspects when dealing with decision trees under privacy settings. For instance, in non-private decision tree evaluations, the tree is traversed from root to leaf along one path. Ideally, PDTE should also traverse the tree along on path. In this case, the total number of comparisons is linear to the depth d of the tree and sublinear to the size of the tree. However, revealing the evaluation path can leak sensitive information even when the tree and the feature vector are well protected (*e.g.*, by encryption). For instance, given the evaluation path, the tree holder can learn whether two feature vectors have the same range of attributes by comparing two evaluation paths; the feature provider can learn information about the tree structure during evaluation. In addition, even the length of a decision path can reveal significant information. For example, if the length is unique among all decision paths, it will immediately reveal the path being evaluated.

To protect the evaluation path, previous PDTE protocols [5, 21, 32, 35] pad the tree to be complete or near-complete and run comparisons for all internal nodes to conceal the decision path information. As a consequence, these protocols suffer from (super) linear computation/communication complexity and are inefficient when evaluating large trees, *e.g.*, trees containing millions of nodes [10].

Table 1: Summary of Existing Two-party PDTE Protocols.

Protocol	Comparison	Communication	Round	Leakage	SC	One-time Setup	Primitives
Bost *et al.* [5]	$\lceil m/2 \rceil$	$O(n+m)$	≥ 6	m	○	●	Leveled-FHE
Wu *et al.* [35]	2^d	$O(2^d + (n+m)\ell)$	6	m, d	○	●	AHE,OT
Tai *et al.* [32]	$\lceil m/2 \rceil$	$O((n+m)\ell)$	4	m	○	●	AHE
Kiss *et al.* [21](GGG)	d	$O(\overline{m}\ell)$	2	$\overline{m}, d$	○	○	GC,OT
Kiss *et al.* [21](HHH)	$\lceil m/2 \rceil$	$O((n+m)\ell)$	4	m	○	●	AHE
Brickell *et al.* [9]	d	$O((n+m)\ell)$	2	m	○	○	AHE,GC,OT
Joye *et al.* [20]	d	$O(d(\ell+n) + 2^d)$	$2d$	d	○	●	AHE, OT
Tueno *et al.* [33](OT)	d	$O((m+n)\ell)$	$4d$	d	○	●	SS,OT
Tueno *et al.* [33](GC)	d	$O((m+n)\ell)$	$4d$	d	○	●	SS,GC,OT
Tueno *et al.* [33](ORAM)	d	$O(d^4\ell)$	$d^2 + 3d$	d	●	●	SS,ORAM,GC
Ma *et al.* [26]	d	$O(dn\ell)$	$2d - 1$	m, d	◐	◐	SS,GC,OT
Our PRF-based	d	$O(dn\ell)$	$(3r_F + 5)d$	m, d	●	●	SS,OT,PRF
Our HE-based	d	$O(dn)$	$8d$	m, d	●	●	SS,OT,AHE

SC represents sublinear communication, **One-time Setup** denotes the tree holder is not required to re-send the tree to the feature provider. m: the number of tree nodes, $\overline{m}$: the number of tree nodes in a depth-padded tree, see [21], n: the dimension of a feature vector, d: the longest depth of a tree, ℓ: the bit size of feature values, r_F: the number of rounds required for securely evaluating PRF F. ●: yes, ○: no, ◐: partially support.

Techniques. This paper proposes a PDTE protocol to obliviously perform decision tree evaluation without leaking the tree model, the feature values or the evaluation path. More importantly, our protocol has a sublinear communication complexity without relying on generic RAM-based secure computation [33].

To hide which node is being accessed during decision tree evaluation, we formalize a functionality called Shared Oblivious Selection (SOS). The functionality allows two parties to obliviously read an element from an array, meanwhile hiding the location and the selected value with secret sharing. We design two efficient SOS protocols based on different techniques. Our first PRF-based SOS protocol adapts Floram [14], which is a communication-efficient Oblivious RAM (ORAM) protocol, for read-only mode. We propose a new preprocessing technique, moving most of its communication overhead to the offline phase. We also design optimized masking mechanisms to make the SOS protocol more efficient. Our second HE-based SOS protocol explores the additive homomorphic property of Paillier encryption [29] to eliminate two-party PRF evaluation. This is done by a share conversion protocol from additive arithmetic sharing to multiplicative arithmetic sharing. Notably, both SOS protocols achieve sublinear offline communication and constant online communication.

We design our PDTE protocols by combining a tree encoding method, the SOS functionality and secure computation. By initializing SOS functionality with either PRF-based or HE-based SOS protocols, we obtain two PDTE protocols with different trade-offs. Our PDTE protocols enjoy sublinear communication with the best security properties of existing PDTE protocols. We prove the security against a semi-honest adversary and analyze its complexity. As shown in Table 1: although many existing PDTE protocols can support sublinear comparisons, only the ORAM-based PDTE protocol [33] requires sublinear communication both in the online and offline phases. We also observe that the two-party PDTE protocol from Ma *et al.* [26] only supports one single classification under standard PDTE security definition. It is unclear how to enhance [26] to support multiple invocations without re-sending new permuted encrypted trees, which essentially incurs linear (offline) communication. However, in some real applications like disease diagnosis, the feature provider (patient) may frequently or periodically interact with the tree holder (health center) to monitor his/her health. Our protocols fully support multiple PDTE queries but only need one-time setup. The setup still needs linear communication, but the overhead will be amortized across queries.

We implemented our PDTE protocols and performed experiments to evaluate the communication and computation performance for different trees under different network conditions. We also compared the performance of our PDTE with the protocols proposed in [21, 27, 33]. The results show that our PRF-based protocol reduces communication around 62× for large trees when compared to [21] and 0.2× than [26]. In the WAN setting with high network latency, our HE-based protocol outperforms the state-of-the-art [21] by 83× in terms of online computation. When compared with [33], our PRF-based protocol requires approximately 40× less total running time while our HE-based protocol saves around 5.5× total running time. Experiments show that our PDTE protocols are practical and scalable, especially for the evaluation of large trees.

Contributions. Our contributions can be summarized as below:

- We propose two SOS protocols that enable two parties to collaboratively and obliviously share an element from an array using only sublinear communication.
- We propose two sublinear-communication PDTE protocols by carefully combining a modified tree encoding method, the SOS functionality and efficient secure computation techniques. We also propose various optimization techniques to make our protocols even more efficient.
- We implemented our PDTE protocols and evaluate their performance. The experimental results show that our protocols are practical: in particular, our PDTE protocols are scalable when evaluating large trees.

Paper organization. We introduce background information in Section 2, and provide an overview of our techniques in Section 3. We construct our primitives and protocols in Section 4, and report

Table 2: Description of Symbols & Notations

Symbols	Descriptions
κ	computational security parameter
λ	statistical security parameter
m	the number of nodes in a decision tree
d	the length of the longest path in a decision tree
d'	a pre-defined depth satisfying $d' \geq d$
$\mathcal{T}$	the decision tree
$\mathcal{X} = (x_1, ..., x_n)$	feature vector of length n
$\mathsf{A}_{\mathcal{T}}$	the encoded array for a decision tree $\mathcal{T}$
$t/l/r/v/c$	the threshold/left child index/right child index/ feature ID/label of a tree node (some non-existing items will be given during tree encoding)
ℓ	the default boolean sharing bit length, *i.e.*, $\ell = \lvert t \rvert = \lvert l \rvert = \lvert r \rvert = \lvert v \rvert = \lvert c \rvert = \lvert x_i \rvert$
ℓ_{v}	the bit length of array elements
ℓ_{b}	the bit length of PRF outputs, *e.g.*, 64, 128, or 256
$B = \lceil \frac{\ell_{\mathsf{v}}}{\ell_{\mathsf{b}}} \rceil$	the number of blocks for ℓ_{v}-bit element

experiments and evaluation results in Section 5. We summarize related work in Section 6 and conclude the paper in Section 7.

2 BACKGROUND

In this section, we introduce background information of decision tree evaluation, cryptographic primitives and definitions used in this paper. Table 2 shows notations used throughout this paper.

2.1 Decision Tree Evaluation

In a decision tree $\mathcal{T}$, each non-leaf node, also called *decision node*, has a threshold $t \in \mathbb{Z}_{2^\ell}$ and each leaf node, also known as *classification label*, has a label value $c \in \mathbb{Z}_{2^\ell}$. A *feature vector*, *i.e.*, a query, is the data to be classified and is denoted as $\mathcal{X} = (x_1, ..., x_n) \in \mathbb{Z}_{2^\ell}^n$ with n feature values.

Decision tree evaluation takes a tree and a feature vector as input and outputs a label as the classification result. The evaluation starts from the root, and it compares the threshold t_1 with $x_{v(1)}$ where $v : i \in \{1, 2, ..., m\} \rightarrow j \in \{1, 2, ..., n\}$ is a map that determines which feature value in $\mathcal{X}$ should be compared with the threshold of i-th node. We will simply use x_{v_i} and $x_{v(i)}$ interchangeably throughout the paper. Depending on whether the comparison results in 1 ($x_{v_i} < t_i$) or 0 ($x_{v_i} \geq t_i$), the evaluation goes either to the left or to the right child and continues the comparison until reaching a leaf. We call this path from the root to a leaf as the *decision path* or *evaluation path* for input $\mathcal{X}$. The *depth* for a decision tree is the length of the longest path. Without ambiguity, we use $\mathcal{T}(\mathcal{X})$ to denote the classification result when using $\mathcal{T}$ over feature vector $\mathcal{X}$.

2.2 Cryptographic Primitives

Oblivious Transfer (OT). OT allows a receiver to obliviously choose one out of many values from a sender [16]. The security of OT guarantees that the receiver only learns the chosen message, and the sender has no idea which value is chosen by the receiver. OT is generally computationally expensive since it requires public-key operations. With OT extension protocols [18], it is efficient to generate (polynomially) many OTs from a small number of OTs.

Boolean Sharing. We denote boolean sharing $\langle x \rangle$ as sharing of $x \in \mathbb{Z}_2$. For a two-party case, $\langle x \rangle$ denotes P_0 holds $\langle x \rangle_0$ and P_1 holds $\langle x \rangle_1$, such that $x = \langle x \rangle_0 \oplus \langle x \rangle_1$, where $\oplus$ represents bitwise XOR. For boolean sharing $\langle x \rangle$ and $\langle y \rangle$, P_0 and P_1 can compute the following operations over shares without interaction. Here $\oplus$ and $\cdot$ denotes addition and multiplication over $\mathbb{Z}_2$.

- $\langle z \rangle \leftarrow \langle x \rangle \oplus \langle y \rangle$: Given $\langle x \rangle$ and $\langle y \rangle$, to compute boolean sharing of $z = x \oplus y$, P_0 just computes $\langle z \rangle_0 \leftarrow \langle x \rangle_0 \oplus \langle y \rangle_0$ and P_1 computes $\langle z \rangle_1 \leftarrow \langle x \rangle_1 \oplus \langle y \rangle_1$.
- $\langle z \rangle \leftarrow \langle x \rangle \oplus c$: Given $\langle x \rangle$ and a constant c, to compute boolean sharing of $z = x \oplus c$, P_0 just computes $\langle z \rangle_0 \leftarrow \langle x \rangle_0 \oplus c$ and P_1 computes $\langle z \rangle_1 \leftarrow \langle x \rangle_1$.
- $\langle z \rangle \leftarrow c \cdot \langle x \rangle$: Given a constant $c \in \mathbb{Z}_2$ and a boolean sharing $\langle x \rangle$, to compute boolean sharing of $z = c \cdot x$, P_0 just computes $\langle z \rangle_0 \leftarrow c \cdot \langle x \rangle_0$ and P_1 computes $\langle z \rangle_1 \leftarrow c \cdot \langle x \rangle_1$.

P_0 and P_1 need to perform an interactive protocol to compute $\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$. One of efficient approaches is using a Beaver Multiplication Triple (BMT) [4]. A BMT $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ satisfies $a \cdot b = c$. Suppose P_0 and P_1 have pre-shared a BMT $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, then they can compute $\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$ efficiently. Specifically, P_0 and P_1 first compute $\langle e \rangle \leftarrow \langle x \rangle \oplus \langle a \rangle$ and $\langle f \rangle \leftarrow \langle y \rangle \oplus \langle b \rangle$, and reveal e and f. In the end, they compute $\langle z \rangle \leftarrow \langle c \rangle \oplus (e \cdot \langle b \rangle) \oplus (f \cdot \langle a \rangle) \oplus (e \cdot f)$, which can be done by local computation. BMTs over $\mathbb{Z}_2$ can be efficient prepossessed by OT extension [23].

In this paper, we will mainly use boolean sharing over $\mathbb{Z}_2^\ell$ for secure computation. The parties share each bit of x using boolean sharing, and computation is done bit-by-bit.

Arithmetic Sharing. We denote sharing a secret $x \in \mathbb{Z}_n$ with arithmetic sharing as $[\![x]\!]$, such that P_0 holds $[\![x]\!]_0 \in \mathbb{Z}_n$ and P_1 holds $[\![x]\!]_1 \in \mathbb{Z}_n$ satisfying $x = [\![x]\!]_0 + [\![x]\!]_1 \pmod n$. Arithmetic sharing is an ideal sharing semantic for computing arithmetic operations such as addition, subtraction and multiplication. Adding arithmetic-shared $[\![x]\!]$ with $[\![y]\!]$ or adding $[\![x]\!]$ with a constant $c \in \mathbb{Z}_n$ can be efficiently done by local computation, and multiplication between two shared data can be done with the help of a BMT over $\mathbb{Z}_n$.

Share Conversion. Different sharing methods have their advantages/disadvantages for different kinds of computation. In particular, boolean sharing is friendly to the boolean circuit, including XOR, AND, *etc.*, while arithmetic sharing is friendly to arithmetic computation such as addition and multiplication. Typical computation usually contains different types of computation, thus it is better to mix-use different types of sharing forms for better efficiency. Share conversion techniques can be used for converting between boolean sharing and arithmetic sharing:

- *Boolean to Arithmetic (B2A) conversion* : Given $\langle x \rangle$ over $\mathbb{Z}_2^\ell$, B2A conversion transforms $\langle x \rangle$ to its arithmetic sharing $[\![x]\!]$ over $\mathbb{Z}_{2^\ell}$. B2A can be done with ℓ OT [13] in $O(1)$ round.
- *Arithmetic to Boolean (A2B) conversion* : Given $[\![x]\!]$ over $\mathbb{Z}_{2^\ell}$, A2B conversion transforms $[\![x]\!]$ to its boolean sharing $\langle x \rangle$ over $\mathbb{Z}_2^\ell$. A2B can be done by computing an addition circuit over $[\![x]\!]_0$ and $[\![x]\!]_1$ using boolean sharing [13, 30] in $O(\log \ell)$ rounds.

For efficiency reason, we mainly use boolean sharing in this paper, and deploy B2A and A2B conversion whenever necessary.

Distributed Oblivious RAM. A similar primitives related to our paper is called Distributed Oblivious RAM (DORAM) for oblivious

data access. A famous DORAM protocol is Floram [14] proposed by Doerner and Shelat, which supports both read and write over secret-shared data. Since we only care about read operation in our setting, we show the following two read-related protocols of Floram:

- $(\mathsf{sk_s}, \mathsf{sk_r}, \mathsf{C}) \leftarrow \mathsf{Init}(1^\kappa, \mathsf{M})$: the protocol initializes a masked array C from an array M of length m. $\mathcal{S}$ obtains $\mathsf{sk_s}$ and $\mathcal{R}$ obtains $\mathsf{sk_r}$, and both parties locally store C. In details, $\mathcal{S}$ holds $\langle \mathsf{M}[i] \rangle_\mathsf{s}$, chooses a PRF key $\mathsf{sk_s} \xleftarrow{\$} \{0,1\}^\kappa$, and sends $\langle \mathsf{M}[i] \rangle_\mathsf{s} \oplus F(\mathsf{sk_s}, i)$ for $i \in [0, m)$ to $\mathcal{R}$, where F represents a PRF function. Similarly, $\mathcal{R}$ holds $\langle \mathsf{M}[i] \rangle_\mathsf{r}$, chooses $\mathsf{sk_r} \xleftarrow{\$} \{0,1\}^\kappa$, and sends $\langle \mathsf{M}[i] \rangle_\mathsf{r} \oplus F(\mathsf{sk_r}, i)$ for $i \in [0, m)$ to $\mathcal{S}$. In the end, both parties can locally compute and store C such that $\mathsf{C}[i] = \langle \mathsf{M}[i] \rangle_\mathsf{s} \oplus \langle \mathsf{M}[i] \rangle_\mathsf{r} \oplus F(\mathsf{sk_s}, i) \oplus F(\mathsf{sk_r}, i) = \mathsf{M}[i] \oplus F(\mathsf{sk_s}, i) \oplus F(\mathsf{sk_r}, i)$ for $i \in [0, m)$.
- $(\langle \mathsf{M}[\mathsf{idx}] \rangle) \leftarrow \mathsf{Read}(\mathsf{sk_s}, \mathsf{sk_r}, \langle \mathsf{idx} \rangle, \mathsf{C})$: the protocol takes $(\mathsf{sk_s}, \langle \mathsf{idx} \rangle_\mathsf{s})$ from $\mathcal{S}$, $(\mathsf{sk_r}, \langle \mathsf{idx} \rangle_\mathsf{r})$ from $\mathcal{R}$, and C locally stored by both parties. In the end, the parties boolean-share $\mathsf{M}[\mathsf{idx}]$, *i.e.*, $\mathcal{S}$ receives $\langle \mathsf{M}[\mathsf{idx}] \rangle_\mathsf{s}$ and $\mathcal{R}$ receives $\langle \mathsf{M}[\mathsf{idx}] \rangle_\mathsf{r}$. In details, the parties use function secret sharing (FSS) [6] to share a weight-1 bit vector S of size m that satisfies $\mathsf{S}[\mathsf{idx}] = 1$ and $\mathsf{S}[i] = 0$ for all $i \neq \mathsf{idx}$. $\mathcal{S}$ computes $c_\mathsf{s} \leftarrow \bigoplus_{i \in [0,m)} \langle \mathsf{S}[i] \rangle_\mathsf{s} \cdot \mathsf{C}[i]$ and $\mathcal{R}$ computes $c_\mathsf{r} \leftarrow \bigoplus_{i \in [0,m)} \langle \mathsf{S}[i] \rangle_\mathsf{r} \cdot \mathsf{C}[i]$. It is easy to check $\mathsf{C}[\mathsf{idx}] = c_\mathsf{s} \oplus c_\mathsf{r}$. The parties then perform two invocations of two-party PRF evaluation to boolean-share $\langle F(\mathsf{sk_s}, \mathsf{idx}) \rangle_\mathsf{s}$ and $\langle F(\mathsf{sk_r}, \mathsf{idx}) \rangle_\mathsf{r}$, thus the parties can share $\mathsf{M}[\mathsf{idx}]$ by taking off two masks.

Paillier Encryption. Paillier encryption scheme is a public key scheme based on Decisional Composite Residuosity problem [29]. The scheme $\mathcal{PE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^\kappa)$: Take as input a security parameter κ, generate two primes p, q (size determined by κ). Compute $N = p \cdot q$. The public key $\mathsf{pk} = N$ and the private key $\mathsf{sk} = (N, p, q)$.
- $\mathsf{Enc}_\mathsf{pk}(x, r)$: Take as input a message $x \in \mathbb{Z}_N$ and a public key pk, with a uniform $r \xleftarrow{\$} \mathbb{Z}_N^*$, the ciphertext is $c := (1 + N)^x \cdot r^N \bmod N^2$. We also use $\mathsf{Enc}(x)$ if we do not care r.
- $\mathsf{Dec}_\mathsf{sk}(c)$: Take as a ciphertext $c \in \mathbb{Z}_{N^2}$ and a private key sk, the decrypted plaintext is $x := \frac{(c^{\phi(N)} \bmod N^2) - 1}{N} \cdot \phi(N)^{-1} \bmod N$, where $\phi(N) = (p-1)(q-1)$.

Paillier encryption is an additively homomorphic encryption (AHE) scheme. At a high level, we can express the homomorphic operations as the following:

- Homomorphic addition: Given two ciphertexts $c = \mathsf{Enc}_{pk}(x, r)$ and $c' = \mathsf{Enc}_{pk}(x', r')$, then $c_\mathsf{add} = c \cdot c' = ((1+N)^x \cdot r^N) \cdot ((1+N)^{x'} \cdot r'^N) = (1+N)^{x+x'} \cdot (rr')^N = \mathsf{Enc}_{pk}(x+x', rr')$.
- Homomorphic multiplication with a constant: Given a ciphertext $c = \mathsf{Enc}_{pk}(x, r)$ and a constant a, then $c_\mathsf{mult} = c^a = ((1+N)^x \cdot r^N)^a = (1+N)^{ax} \cdot (r^a)^N = \mathsf{Enc}_{pk}(a \cdot x, r^a)$.

2.3 Semi-honest Security

We design our protocols and prove their security under semi-honest security model [16, 17]. A protocol Π securely computes a function f under semi-honest adversary if the adversary cannot learn more information beyond what can be computed from his input and output. A protocol may allow the parties to learn certain leakages after execution, and we treat such leakages as a part of the output. Formally, let $f(x, y) = (f_0(x, y), f_1(x, y))$ be a function with inputs x, y and outputs $(f_0(x, y), f_1(x, y))$. For a two-party protocol Π computing function $f(x, y)$, we use $\mathsf{View}_i^\Pi(1^\lambda, x, y) = (\omega, r^i; m_1^i, ..., m_t^i)$ to denote the view of the i-th party ($i \in \{0, 1\}$) during protocol execution where $\omega \in \{x, y\}$ depends on i , r^i represents the contents of its random values, and $m_1^i, ..., m_t^i$ denotes the messages received by the i-th party.

Definition 1. The protocol Π securely computes f for any inputs (x, y) if for any party P_i ($i \in \{0, 1\}$) corrupted by a semi-honest adversary $\mathcal{A}$, there exists a *probabilistic polynomial time* (PPT) simulator Sim_i that can produce a simulated view that is computationally indistinguishable from $\mathsf{View}_i^\Pi(1^\lambda, x, y)$:

$$\{\mathsf{Sim}_i(1^\lambda, x, f_i(x, y))\} \overset{c}{\equiv} \{\mathsf{View}_i^\Pi(1^\lambda, x, y)\}.$$

3 OVERVIEW OF OUR APPROACH

In this section, we show our protocol setting, security requirement and techniques overview for our PDTE protocol.

3.1 Protocol Setting and Security Guarantee

Our PDTE protocol is in the two-party setting in which a tree holder P_0 owns a decision tree model $\mathcal{T}$ and a feature provider P_1 provides a feature vector $\mathcal{X} = (x_1, \cdots, x_n)$. At the end of the protocol, P_1 receives a prediction $\mathcal{T}(\mathcal{X})$. We assume both parties have sufficient storage to store $\mathcal{T}$ or $\mathcal{X}$ locally. In our protocol, there is no third-party involved in the computation.

We assume an adversary is static semi-honest. A corrupted party will strictly follow the protocol but may attempt to learn as much information as possible. Formally, let $\mathcal{F}_\mathrm{DT}(\mathcal{T}, \mathcal{X}) \rightarrow (\mathcal{F}_0(\mathcal{T}, \mathcal{X}), \mathcal{F}_1(\mathcal{T}, \mathcal{X}))$ be the decision tree functionality where a decision tree model $\mathcal{T}$ and a feature vector $\mathcal{X}$ is provided by P_0 and P_1 respectively. In the end, the output of $\mathcal{F}_\mathrm{DT}$ is that P_i obtains $\mathcal{F}_i(\mathcal{T}, \mathcal{X})$ for $i \in \{0, 1\}$. A PDTE protocol Π securely computes $\mathcal{F}_\mathrm{DT}$ if there exist PPT simulators Sim_0 and Sim_1 such that for any $\mathcal{T}$ and $\mathcal{X}$:

$$\{\mathsf{Sim}_0(1^\lambda, \mathcal{T}, \mathcal{F}_0(\mathcal{T}, \mathcal{X}))\} \overset{c}{\equiv} \{\mathsf{View}_0^\Pi(1^\lambda, \mathcal{T}, \mathcal{X})\},$$
$$\{\mathsf{Sim}_1(1^\lambda, \mathcal{X}, \mathcal{F}_1(\mathcal{T}, \mathcal{X}))\} \overset{c}{\equiv} \{\mathsf{View}_1^\Pi(1^\lambda, \mathcal{T}, \mathcal{X})\}.$$

For our setting, $\mathcal{F}_0(\mathcal{T}, \mathcal{X}) = \{\perp, \mathcal{L}_0(\mathcal{X})\}$ and $\mathcal{F}_1(\mathcal{T}, \mathcal{X}) = \{\mathcal{T}(\mathcal{X}), \mathcal{L}_1(\mathcal{T})\}$. $\mathcal{L}_0$ and $\mathcal{L}_1$ denote two stateless leakage functions where $\mathcal{L}_0(\mathcal{X}) = \{n\}$ and $\mathcal{L}_1(\mathcal{T}) = \{m, d\}$, *i.e.*, P_0 obtains no output but only the number of features n in the query whereas the output of P_1 contains classification result $\mathcal{T}(\mathcal{X})$, the number of tree nodes m, and the length of longest decision path d.

3.2 Design Goals

- We aim to achieve a high security standard for our PDTE protocol. The protocol should only output the classification result to the feature provider. Besides that, the parties only learn minimal leakages as we defined.
- We aim to design PDTE protocol with sublinear communication, and with concretely practical efficiency than construction from generic ORAM-based secure computation.

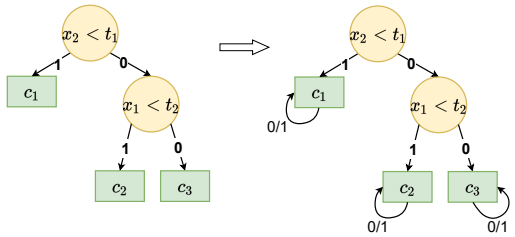
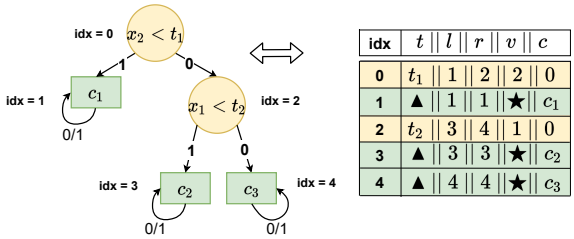


Figure 1: A Modified Decision Tree



idx	$t \,\|\|\, l \,\|\, r \,\|\, v \,\|\, c$
0	$t_1 \,\|\|\, 1 \,\|\, 2 \,\|\, 2 \,\|\, 0$
1	▲ $\|\|\, 1 \,\|\, 1 \,\|$★$\|\, c_1$
2	$t_2 \,\|\|\, 3 \,\|\, 4 \,\|\, 1 \,\|\, 0$
3	▲ $\|\|\, 3 \,\|\, 3 \,\|$★$\|\, c_2$
4	▲ $\|\|\, 4 \,\|\, 4 \,\|$★$\|\, c_3$

Figure 2: Encoding A Decision Tree as an Array: '▲' represents the value can set as any value, and '★' represents the value can be randomly selected from $[0, 1, \ldots, n-1]$, where n is the dimension of corresponding feature vector

- We aim to design our PDTE protocol in a modular manner. This allows us to optimize each component for the whole PDTE protocol, which also allows us to argue the security of our PDTE protocol easily.

3.3 Technique Overview

Encoding Decision Trees and Feature Vectors. We follow the OAI approach [33] to encode decision trees. The difference is that we modify a traditional decision tree as shown in Fig. 1. We redirect each leaf node by setting its left and right children indexing to the leaf itself. The modified decision tree is encoded as an array $\mathsf{A}_{\mathcal{T}}$ shown in Fig. 2. A node in the tree is stored in $\mathsf{A}_{\mathcal{T}}$ in Depth First Search (DFS) order, and we use $\mathsf{A}_{\mathcal{T}}[i]$ to store all necessary information of the i-th node. Specifically, $\mathsf{A}_{\mathcal{T}}[i]$ is constructed by five values: 1) threshold, t; 2) left child index, l; 3) right child index, r; 4) feature ID, v and 5) classification label, c. For example, the right most leaf in Fig. 2 can be represented as $\mathsf{A}_{\mathcal{T}}[4] = a||4||4||b||c_3$, where $a \xleftarrow{\$} R$ and $b \in [0, n-1]$. Similarly, a feature vector from the feature provider can be naturally represented as an array $\mathcal{X}$ of length n.

Algorithm 1 $\mathsf{rst} \leftarrow \mathsf{DT}(\mathsf{A}_{\mathcal{T}}, \mathcal{X})$

1: $\mathsf{idx} \leftarrow 0, \mathsf{rst} \leftarrow \perp$
2: **for** $0 \le i < d$ **do**
3: $\quad t||l||r||v||c \leftarrow \mathsf{A}_{\mathcal{T}}[\mathsf{idx}]$
4: $\quad b \leftarrow \mathcal{X}[v] < t$
5: $\quad \mathsf{idx} \leftarrow r \oplus b \cdot (l \oplus r)$ ▷ if $b = 1$, $\mathsf{idx} = l$; else $\mathsf{idx} = r$
6: $\quad \mathsf{rst} \leftarrow c$
7: **end for**
8: **return** rst

The Proposed Decision Tree Algorithm. Given a decision tree array $\mathsf{A}_{\mathcal{T}}$ and a feature array $\mathcal{X}$, we can perform decision tree evaluation over the two arrays. Algorithm 1 shows the algorithm, notably, it always runs d iterations to output a correct classification result, independent of which path is taken.

The algorithm starts from the root node, *i.e.*, $\mathsf{idx} = 0$, and it allocates a value rst for classification result. In each iteration, the algorithm first selects a node $t||l||r||v||c \leftarrow \mathsf{A}_{\mathcal{T}}[\mathsf{idx}]$ according to idx. From the feature ID v, the algorithm can select $\mathcal{X}[v]$, and do a comparison $b \leftarrow \mathcal{X}[v] < t$. If $b = 1$, then set $\mathsf{idx} \leftarrow l$, otherwise $\mathsf{idx} \leftarrow r$. In the end of each iteration, update $\mathsf{rst} \leftarrow c$. Due to our modification to decision tree, we can ensure that rst will hold a correct classification result once the evaluation reaches a leaf.

Functionality $\mathcal{F}_{\mathsf{sos}}^{(\mathsf{M})}$

Parameters: Two parties denoted as $\mathcal{S}$ and $\mathcal{R}$.

- **Setup**: upon receiving $(\textsc{Setup}, \mathsf{M}, \ell_{\mathsf{v}})$ from $\mathcal{S}$ and $(\textsc{Setup})$ from $\mathcal{R}$, store M.
- **Select**: upon receiving $(\textsc{Select}, \langle \mathsf{idx} \rangle_{\mathsf{s}})$ from $\mathcal{S}$ and $(\textsc{Select}, \langle \mathsf{idx} \rangle_{\mathsf{r}})$ from $\mathcal{R}$:
  - recover $\mathsf{idx} \leftarrow \langle \mathsf{idx} \rangle_{\mathsf{s}} \oplus \langle \mathsf{idx} \rangle_{\mathsf{r}}$
  - $e \xleftarrow{\$} \mathbb{Z}_2^{\ell_v}$, send e to $\mathcal{S}$ and $e \oplus \mathsf{M}[\mathsf{idx}]$ to $\mathcal{R}$

Figure 3: The Shared Oblivious Selection Functionality $\mathcal{F}_{\mathsf{sos}}^{(\mathsf{M})}$

Indeed, this also ensures us to hide the length of the longest path by setting iteration number as d' where $d' \ge d$. Note that setting $d' \ge d$ also hides d. For simplicity, in this paper, we take $d' = d$.

Challenges and Solutions. Things are tricky when evaluating Algorithm 1 in the secret domain. A basic requirement is to seal all values from both parties. For this part, we observe that the involved computation including comparison (line.4, Algorithm 1) and 1-out-of-2 MUX operation (line.5, Algorithm 1). We choose boolean sharing for the underlying secret-sharing scheme because it matches well with performing bite-level secure computation. For secure comparison and secure MUX protocol, we use existing protocols [13] directly.

However, hiding intermediate values is not sufficient to get rid of all the leakages. Taking node selection as an example (line.3, Algorithm 1), if the secure computation leaks the memory access pattern during each iteration, the client can learn idx directly, then the client learns decision path, which is not allowed from our security requirement. The goal here is to obliviously share $\mathsf{A}_{\mathcal{T}}[\mathsf{idx}]$ between parties where $\mathsf{A}_{\mathcal{T}}$ is provided by the tree holder and idx is also secret-shared, but neither party learns idx or $\mathsf{A}_{\mathcal{T}}[\mathsf{idx}]$[1].

In Fig. 3, we formalize the above task as a Shared Oblivious Selection (SOS) functionality. A possible way to realize $\mathcal{F}_{\mathsf{sos}}^{(\mathsf{M})}$ is via generic ORAM-based secure computation. Tueno *et al.* [33] use Circuit ORAM [34] to design a sublinear-communication PDTE protocol. However, this approach requires the parties to evaluate

[1]In our protocol, idx is secret-shared from previous secure computation, which means neither party learns the underlying value. Our definition of SOS functionality also ensures $\mathsf{M}[\mathsf{idx}]$ is inherently randomly shared.

ORAM circuit *inside* secure computation, causing massive computation and communication overhead in practice. Our goal is to minimize the overhead by designing specialized SOS protocols. Our design follows from the observation that ORAM is overkill since PDTE protocols only need read operations. Therefore, what we need is a secure computation protocol over Oblivious Read-Only-Memory (OROM). Such simplification allows us to design specialized SOS protocols to compute $\mathcal{F}_{\text{sos}}^{(\mathsf{M})}$ more efficiently. We also propose many optimizations to improve efficiency both asymptotically and concretely, some of them are of independent interests.

Put All Together. We will use both SOS protocol and boolean-sharing based secure computation to evaluate our modified decision tree evaluation algorithm. Intuitively, when performing decision tree evaluation using secure computation, all intermediate values are secret-shared between the parties, and the parties run secure computation to traverse the decision tree obliviously. Note that the algorithm itself does not leak length information. We further use oblivious selection to conceal access pattern leakage. The parties cannot learn which decision path is taken since everything is evaluated in the secret domain. We also propose concrete optimizations in our PDTE protocol. We will discuss our techniques in detail in the next section.

4 PRIVATE DECISION TREE EVALUATION WITH SUBLINEAR COMMUNICATION

In this section, we give our PDTE protocol with sublinear communication. We first propose two SOS protocols with sublinear communication under different trade-offs. Leveraging the SOS functionality, we design our sublinear PDTE protocol.

Parameters: Two parties denoted as $\mathcal{S}$ and $\mathcal{R}$; array length m; index bit length ℓ; array element bit length ℓ_{v}.

[Setup] Upon receiving $(\textsc{Setup}, \mathsf{M}, \ell_{\mathsf{v}})$ from $\mathcal{S}$ and $(\textsc{Setup}, \perp)$ from $\mathcal{R}$, $\mathcal{S}$ stores M locally.

[Select] Upon receiving $(\textsc{Select}, \langle \mathsf{idx} \rangle_{\mathsf{s}})$ from $\mathcal{S}$ and $(\textsc{Select}, \langle \mathsf{idx} \rangle_{\mathsf{r}})$ from $\mathcal{R}$:

(1) $\mathcal{S}$ and $\mathcal{R}$ run a B2A conversion [13], transforming $\langle \mathsf{idx} \rangle$ to its arithmetic form $[\![\mathsf{idx}]\!]$ over $\mathbb{Z}_m$.
(2) $\mathcal{S}$ samples $r \xleftarrow{\$} \mathbb{Z}_2^{\ell_{\mathsf{v}}}$, computes m messages $\{E_i\}_{i\in[0,m)}$ such that $E_i = r \oplus \mathsf{M}[i + [\![\mathsf{idx}]\!]_{\mathsf{s}} \pmod m]$.
(3) $\mathcal{S}$ and $\mathcal{R}$ invoke 1-out-of-m OT functionality $\mathcal{F}_{\mathsf{ot}}$. $\mathcal{S}$ inputs $\{E_i\}_{i\in[0,m)}$ and $\mathcal{R}$ provides $[\![\mathsf{idx}]\!]_{\mathsf{r}}$ as choice input. By definition of OT, $\mathcal{R}$ receives $r \oplus \mathsf{M}[\mathsf{idx}]$.
(4) $\mathcal{S}$ outputs r and $\mathcal{R}$ outputs $r \oplus \mathsf{M}[\mathsf{idx}]$.

Figure 4: Linear-communication SOS Protocol from OT [33]

4.1 Shared Oblivious Selection Protocol

In Fig. 4, we first show an SOS protocol from 1-out-of-m OT as previously done in [33]. We will use OT-based SOS protocol over feature vectors that are usually with low dimension. Despite being conceptually efficient and straightforward for small arrays, OT-based construction requires $O(m)$ online communication; this is prohibitively high when m is large. For oblivious selection over tree nodes, we need communication-efficient SOS protocols since, usually, a tree contains thousands to millions of nodes [10].

Functionality $\mathcal{F}_{\text{pre}}$

Parameters: Two parties denoted as $\mathcal{S}$ and $\mathcal{R}$; weight-1 bit vector length m; BMT arithmetic module n; index bit length ℓ.

- **GenWBV**: upon receiving $(\textsc{GenWBV}, \ell, m)$ from all parties:
  - sample $\mathsf{rdx} \xleftarrow{\$} \mathbb{Z}_\ell$, compute S such that $\mathsf{S}[\mathsf{rdx}] = 1$ and $\mathsf{S}[j] = 0$ for all $j \neq \mathsf{rdx}$.
  - sample $e \xleftarrow{\$} \mathbb{Z}_2^\ell$, $r \xleftarrow{\$} \mathbb{Z}_2^m$, send (e, r) to $\mathcal{S}$ and $(e \oplus \mathsf{rdx}, \mathsf{S} \oplus r)$ to $\mathcal{R}$.
- **GenBMT**: upon receiving $(\textsc{GenBMT}, n, [\![a]\!]_{\mathsf{s}}, [\![b]\!]_{\mathsf{s}})$ from $\mathcal{S}$ and $(\textsc{GenBMT}, n, [\![a]\!]_{\mathsf{r}}, [\![b]\!]_{\mathsf{r}})$ from $\mathcal{R}$:
  - compute $c \leftarrow ([\![a]\!]_{\mathsf{s}} + [\![a]\!]_{\mathsf{r}}) \cdot ([\![b]\!]_{\mathsf{s}} + [\![b]\!]_{\mathsf{r}}) \pmod n$.
  - sample $r \xleftarrow{\$} \mathbb{Z}_n$, send $-r \pmod n$ to $\mathcal{S}$ and $r+c \pmod n$ to $\mathcal{R}$.

Figure 5: The Pre-processing Functionality $\mathcal{F}_{\text{pre}}$

Functionality $\mathcal{F}_{\text{sprf}}$

Parameters: Two parties denoted as $\mathcal{S}$ and $\mathcal{R}$; PRF $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{b}}}$.

- **Eval**: upon receiving $(\textsc{Eval}, \mathsf{sk}, \langle \mathsf{idx} \rangle_{\mathsf{s}})$ from $\mathcal{S}$ and $(\textsc{Eval}, \langle \mathsf{idx} \rangle_{\mathsf{r}})$ from $\mathcal{R}$, compute:
  - reconstruct $\mathsf{idx} \leftarrow \langle \mathsf{idx} \rangle_{\mathsf{s}} \oplus \langle \mathsf{idx} \rangle_{\mathsf{r}}$, compute $F(\mathsf{sk}, \mathsf{idx})$
  - sample $r \xleftarrow{\$} \mathbb{Z}_2^{\ell_{\mathsf{b}}}$, send r to $\mathcal{S}$ and $r \oplus F(\mathsf{sk}, \mathsf{idx})$ to $\mathcal{R}$

Figure 6: The Shared PRF Functionality $\mathcal{F}_{\text{sprf}}$

4.1.1 ***Available Ideal Functionalities.*** Before describing our protocol, we introduce two ideal functionalities. One is called pre-processing functionality $\mathcal{F}_{\text{pre}}$ defined in Fig. 5 that generates useful correlated randomnesses, including shared weight-1 bit vector (WBV) and Beaver multiplication triple (BMT). Another is called two-party shared PRF functionality $\mathcal{F}_{\text{sprf}}$ in Fig. 6. In particular, WBVs can be constructed from Function Secret Sharing (FSS) [6, 7] with sublinear communication, and BMTs can be generated from AHE or OT [13]. We summarize how to generate these correlated randomnesses in Appendix B. One can initialize $\mathcal{F}_{\text{sprf}}$ by evaluating a block cipher circuit using secure two-party computation. In the following, we will use these ideal functionalities directly; such approach, which is known as hybrid-model, is commonly used in designing secure computation protocols [16, 17].

4.1.2 ***The PRF-based SOS Protocol.*** Our PRF-based SOS protocol in Fig. 7 only needs a single two-party PRF invocation regardless of the array length. The protocol is inspired by Floram, but we propose new techniques to improve efficiency.

New Pre-processing Technique. We propose a new preprocessing technique for WBVs inspired by Beaver's circuit derandomization technique [3], moving all its generation work to the offline phase.

Let S be a WBV over a random index rdx, the parties can use S during the online phase to share an element at location idx. Specifically, the parties simply reveal $\delta = \mathsf{rdx} - \mathsf{idx} \pmod m$ and compute:

$$\langle \mathsf{C}[\mathsf{idx}] \rangle = \bigoplus_{i \in [0,m)} \langle \mathsf{S}[\delta + i \pmod m] \rangle \cdot \mathsf{C}[i],$$

then $\mathsf{C}[\mathsf{idx}]$ is shared between parties as required.[2] This derandomization technique enables the parties to pre-generate sufficient weight-1 bit vectors to trade an efficient online protocol.

Note that idx and rdx are shared in boolean form, for better efficiency, the parties first perform B2A conversion [13] before performing subtraction; this can be efficiently done by ℓ OTs in $O(1)$ round for ℓ-bit boolean sharing [13]. Indeed, we can get rid of B2A conversion using a slightly different technique. Specifically, when $m = 2^k$, the parties can simply reveal $\delta = \mathsf{idx} \oplus \mathsf{rdx}$ and compute:

$$\langle \mathsf{C}[\mathsf{idx}] \rangle = \bigoplus_{i \in [0,m)} \langle \mathsf{S}[\delta \oplus i] \rangle \cdot \mathsf{C}[i].$$

For generic cases where $2^{k-1} \leq m \leq 2^k$, $\mathcal{S}$ has to pad M of size 2^k and randomly places elements in the padded array before encryption; this certainly needs more storage space but only doubles the storage cost at most.

Reduce Overhead of Two-party PRF Evaluation. Our protocol only needs a single mask to hide the underlying message, instead of two used in Floram (see section 2). In short, Floram works for secret-shared data; it is necessary to use two masks, each for protecting a share from one party. For our setting, double-masking is overkill since $\mathcal{S}$ already knows M. We only need one mask to protect M from $\mathcal{R}$. As such, we change the original masking mechanism to $\mathsf{C}[i] \leftarrow \mathsf{M}[i] \oplus F(\mathsf{sk}_\mathsf{s}, i)$ for $i \in [0, m)$. Our optimization reduces half of the two-party PRF evaluations, which can significantly improve efficiency in practice since two-party PRF evaluation contributes the main overhead to the PRF-based protocol.

In addition, we provide implementation-level optimizations. In particular, Floram uses AES for instantiating F. However, AES contains many AND gates, *e.g.*, AES-128 needs 6,400 AND-gates per evaluation[3], which incurs significant overhead when evaluated by secure computation. We provide an optimized implementation from LowMC block cipher [1]. LowMC is MPC-friendly, designed with much fewer AND-gates. It provides tunable options between block size, evaluation round and security level; this allows us to choose the best parameters for different scenarios.

Optimized Multi-block Masking Strategy. We propose an MPC-friendly masking method for M with large-size elements. Specifically, suppose each element has a size of ℓ_v, and the PRF output is of size ℓ_b. Note that when $\ell_\mathsf{b} < \ell_\mathsf{v}$, a single PRF output cannot mask the whole element. To handle the issue, we divide the element into multiple blocks and generate a mask for each. In addition, we design a fixed-key masking strategy, which turns to be MPC-friendly. Specifically, denote $\mathsf{M}[i][j]$ as $\mathsf{M}[i]$'s j-th block, $\mathcal{S}$ simply encrypts the block as:

$$\mathsf{C}[i][j] \leftarrow \mathsf{M}[i][j] \oplus F(\mathsf{sk}_\mathsf{s}, i||j),$$

[2] $\langle \mathsf{S}[\delta + i \pmod m] \rangle \cdot \mathsf{C}[i]$ is computed in the secret-shared fashion, *i.e.*, $\mathcal{S}$ computes $\langle \mathsf{C}[\mathsf{idx}] \rangle_\mathsf{s} = \bigoplus_{i \in [0,m)} \langle \mathsf{S}[\delta + i \pmod m] \rangle_\mathsf{s} \cdot \mathsf{C}[i]$, and $\mathcal{R}$ computes $\langle \mathsf{C}[\mathsf{idx}] \rangle_\mathsf{r} = \bigoplus_{i \in [0,m)} \langle \mathsf{S}[\delta + i \pmod m] \rangle_\mathsf{r} \cdot \mathsf{C}[i]$.

[3] https://homes.esat.kuleuven.be/~nsmart/MPC/

Parameters: PRF $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^{\ell_\mathsf{b}}$; index bit length ℓ; bit length of PRF output ℓ_b; bit length of array element ℓ_v; number of PRF output for an element $B = \lceil \frac{\ell_\mathsf{v}}{\ell_\mathsf{b}} \rceil$; array length $m = |\mathsf{M}|$.

[Setup] Upon receiving $(\textsc{Setup}, \mathsf{M}, \ell_\mathsf{v})$ from $\mathcal{S}$ and $(\textsc{Setup}, \perp)$ from $\mathcal{R}$:

(1) $\mathcal{S}$ samples a secret key $\mathsf{sk}_\mathsf{s} \xleftarrow{\$} \{0,1\}^\kappa$ for F, and encrypts M to obtain ciphertext C such that $\mathsf{C}[i][j] = \mathsf{M}[i][j] \oplus F(\mathsf{sk}_\mathsf{s}, i||j)$ for $i \in [0, m)$ and $j \in [0, B)$.
(2) $\mathcal{S}$ sends C to $\mathcal{R}$. $\mathcal{S}$ stores $(\mathsf{sk}_\mathsf{s}, \mathsf{C})$ and $\mathcal{R}$ stores C.

[Select] Upon receiving $(\textsc{Select}, \langle \mathsf{idx} \rangle_\mathsf{s})$ from $\mathcal{S}$ and $(\textsc{Select}, \langle \mathsf{idx} \rangle_\mathsf{r})$ from $\mathcal{R}$:

(1) $\mathcal{S}$ and $\mathcal{R}$ send $(\textsc{GenWBV}, \ell, m)$ to $\mathcal{F}_\mathsf{pre}$, obtain $(\langle \mathsf{rdx} \rangle, \langle \mathsf{S} \rangle)$.
(2) $\mathcal{S}$ and $\mathcal{R}$ convert $\langle \mathsf{idx} \rangle$ and $\langle \mathsf{rdx} \rangle$ to arithmetic share form $[\![\mathsf{idx}]\!]$ and $[\![\mathsf{rdx}]\!]$ by B2A conversion [13].
(3) $\mathcal{S}$ and $\mathcal{R}$ compute $[\![\delta]\!] = [\![\mathsf{rdx}]\!] - [\![\mathsf{idx}]\!] \pmod m$, and reveal δ in clear. Then they compute $\langle e \rangle = \bigoplus_{i=0}^{m-1} \langle \mathsf{S}[i + \delta \pmod m] \rangle \cdot \mathsf{C}[i]$.
(4) $\mathcal{S}$ and $\mathcal{R}$ compute B shared indexes $\langle \mathsf{idx}||j \rangle = \langle \mathsf{idx} << \lceil \log B \rceil \rangle + j$ for $j \in [0, B)$, locally.
(5) $\mathcal{S}$ and $\mathcal{R}$ call $\mathcal{F}_\mathsf{sprf}$ for each of B blocks. For $j \in [0, B)$, $\mathcal{S}$ inputs $(\textsc{Eval}, \mathsf{sk}_\mathsf{s}, \langle \mathsf{idx}||j \rangle_\mathsf{s})$ and $\mathcal{R}$ inputs $(\textsc{Eval}, \langle \mathsf{idx}||j \rangle_\mathsf{r})$ to $\mathcal{F}_\mathsf{sprf}$, $\mathcal{F}_\mathsf{sprf}$ sends $\langle f_j \rangle_\mathsf{s}$ to $\mathcal{S}$ and $\langle f_j \rangle_\mathsf{r}$ to $\mathcal{R}$ such that $\langle f_j \rangle_\mathsf{s} \oplus \langle f_j \rangle_\mathsf{r} = F(\mathsf{sk}_\mathsf{s}, \mathsf{idx}||j)$.
(6) Let $\langle f \rangle$ be $\langle f_0 \rangle || \cdots || \langle f_{B-1} \rangle$, $\mathcal{S}$ and $\mathcal{R}$ locally compute $\langle m \rangle = \langle e \rangle \oplus \langle f \rangle$.

Figure 7: The PRF-based SOS Protocol

where $i||j = i \cdot 2^{\lceil \log_2 B \rceil} + j$, $B = \lceil \frac{\ell_\mathsf{v}}{\ell_\mathsf{b}} \rceil$. Since $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^{\ell_\mathsf{b}}$ is defined over ℓ-bits inputs, one should note that ℓ should be large enough, *i.e.*, $i \cdot 2^{\lceil \log_2 B \rceil} + j < 2^\ell$ for all $i \in [0, m)$ and $j \in [0, B)$, otherwise it is possible to encounter a wrap-around issue incurring $i_1||j_1 = i_2||j_2$ and $F(\mathsf{sk}_\mathsf{s}, i_1||j_1) = F(\mathsf{sk}_\mathsf{s}, i_2||j_2)$, then $\mathcal{R}$ can easily learn:

$$\mathsf{M}[i_1][j_1] \oplus \mathsf{M}[i_2][j_2] = \mathsf{C}[i_1][j_1] \oplus \mathsf{C}[i_2][j_2].$$

As such, we require all $i||j$ are unique for $i \in [0, n)$ and $j \in [0, B)$. Indeed, setting $\lceil \log_2 n \rceil + \lceil \log_2 B \rceil \leq \ell$ suffices for the goal. Taking a concrete example, when $\ell = 64$, $B = 5$, our indexing method can support oblivious selection on 2^{61} elements, which is already sufficient in practice.

Two benefits follow from the design. First, since i is boolean-shared bit-by-bit and j is public, sharing $i||j$ is essentially free: each party just cyclically left-shifts its share of i by $\lceil \log_2 B \rceil$ bits, and sets the lower $\lceil \log_2 B \rceil$ bits to be the share of j, *i.e.*, $i||j = (i << \lceil \log_2 B \rceil) + j$. Note that j is publicly known to both parties, hence sharing j is easy, *e.g.*, $j = 0 \oplus j$. As a result, the parties can non-interactively share $i||j$ for all $j \in [0, B)$ from the sharing of i. Second, since F uses a fixed key sk_s, we can implement two-party PRF evaluation in a SIMD mode, allowing the parties to perform PRF evaluation in parallel during oblivious selection, which improves efficiency by reducing rounds.

Complexity Analysis. Same as Floram, the parties need to perform linear memory scan over all encrypted array elements; this is relatively cheap given highly efficient hardware nowadays. Besides, our protocol requires both parties to store the encrypted array locally. We believe the price is desirable to trade a better online communication in scenarios where multiple invocations are frequently performed between the parties. However, concretely, the overhead for two-party PRF evaluation can be relatively high. In particular, even using LowMC PRF, the parties still have to evaluate thousands of AND gates using secure computation, which can cause massive time consumption over a high-latency network; this motivates us to design a round-efficient SOS protocol.

Security. We have Theorem 2 to capture security of the PRF-based SOS protocol. The proof can be found in Appendices A.1.

Theorem 2. *Let F be a secure PRF, the oblivious selection protocol in Fig. 7 securely computes the functionality $\mathcal{F}_{sos}^{(\mathsf{M})}$ in $(\mathcal{F}_{sprf}, \mathcal{F}_{pre})$-hybrid model under a semi-honest adversary.*

4.1.3 *The HE-based SOS protocol*.

The prior PRF-based SOS protocol has sublinear communication, but the parties must perform a two-party PRF evaluation per oblivious selection. Though we provide optimizations to reduce the overhead, its round complexity is relatively high given the intrinsic complexity of PRFs; this can incur considerable time consumption over a high-latency network.

New Solution. We design a sublinear SOS protocol with better round complexity by using Paillier's AHE [29]. The idea is similar to the PRF-based SOS protocol, but we explore the additive homomorphic property of Paillier encryption to eliminate two-party PRF evaluation. Our key technique is a new share conversion protocol between additive arithmetic sharing and multiplicative arithmetic sharing over $\mathbb{Z}_{N^2}$, which is of independent interest.

Like the PRF-based SOS protocol, $\mathcal{S}$ encrypts M, but uses its public key pk and sends ciphertext C to $\mathcal{R}$. When selecting an element indexed by a shared index idx among M, the parties still use the shared weight-1 bit vector to obliviously share $\langle \mathsf{C}[\mathsf{idx}] \rangle$ in boolean fashion, and then convert $\langle \mathsf{C}[\mathsf{idx}] \rangle$ to arithmetic sharing $[\![\mathsf{C}[\mathsf{idx}]]\!]$ over $\mathbb{Z}_{N^2}$ by B2A conversion. However, one should note that such operation can only *additively* share $\mathsf{C}[\mathsf{idx}]$:

$$[\![\mathsf{C}[\mathsf{idx}]]\!]_\mathrm{s} + [\![\mathsf{C}[\mathsf{idx}]]\!]_\mathrm{r} = \mathsf{C}[\mathsf{idx}] \pmod{N^2}.$$

In order to facilitate homomorphic property of Paillier encryption, we need the ciphertext to be shared *multiplicatively* over $\mathbb{Z}_{N^2}^*$:

$$[\![\mathsf{C}[\mathsf{idx}]]\!]_\mathrm{s}^* \cdot [\![\mathsf{C}[\mathsf{idx}]]\!]_\mathrm{r}^* = \mathsf{C}[\mathsf{idx}] \pmod{N^2}.$$

With such conversion, the parties can explore a shared decryption technique to share the encrypted message. Therefore, our first challenge is designing an efficient protocol to perform such conversion.

Additive to Multiplicative Sharing Conversion over $\mathbb{Z}_{N^2}$. Given an additive sharing $[\![x]\!]$ over $\mathbb{Z}_{N^2}$ where $\mathcal{S}$ holding $[\![x]\!]_\mathrm{s}$ and $\mathcal{R}$ holding $[\![x]\!]_\mathrm{r}$, we want to convert it to its multiplicative sharing form $[\![x]\!]^*$ satisfying $x = [\![x]\!]_\mathrm{s}^* \cdot [\![x]\!]_\mathrm{r}^* \pmod{N^2}$.

The idea is $\mathcal{S}$ can sample a random value $\gamma \xleftarrow{\$} \mathbb{Z}_{N^2}^*$, and $\mathcal{R}$ can recover $x \cdot \gamma^{-1} \pmod{N^2}$ by running a secure protocol with $\mathcal{S}$. Now the question is how to securely compute $x \cdot \gamma^{-1} \pmod{N^2}$. It is easy to see that:

$$x \cdot \gamma^{-1} = [\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + [\![x]\!]_\mathrm{r} \cdot \gamma^{-1} \pmod{N^2}.$$

Parameters: index bit length ℓ; array element bit size $\ell_\mathsf{v} \ll |N|$; array length $m = |\mathsf{M}|$; computational security parameter κ; statistical security parameter λ.

[Setup] Upon receiving $(\textsc{Setup}, \mathsf{M}, \ell_\mathsf{v})$ from $\mathcal{S}$ and $(\textsc{Setup}, \perp)$ from $\mathcal{R}$. Then:

(1) $\mathcal{S}$ generates Paillier public/secret key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$.
(2) $\mathcal{S}$ encrypts M to obtain C such that $\mathsf{C}[i] \leftarrow \mathsf{Enc}_\mathsf{pk}(\mathsf{M}[i])$ for $i \in [0, m)$.
(3) $\mathcal{S}$ sends $(\mathsf{pk}, \mathsf{C})$ to $\mathcal{R}$. Both $\mathcal{S}$ and $\mathcal{R}$ store C locally.

[Select] Upon receiving $(\textsc{Select}, \langle \mathsf{idx} \rangle_\mathrm{s})$ from $\mathcal{S}$ and $(\textsc{Select}, \langle \mathsf{idx} \rangle_\mathrm{r})$ from $\mathcal{R}$. Then:

(1) $\mathcal{S}$ and $\mathcal{R}$ send $(\textsc{GenWBV}, \sigma = \lceil \log_2 m \rceil, m)$ to $\mathcal{F}_\mathrm{pre}$, obtain $(\langle \mathsf{rdx} \rangle, \langle \mathsf{S} \rangle)$.
(2) $\mathcal{S}$ and $\mathcal{R}$ convert $\langle \mathsf{idx} \rangle$ and $\langle \mathsf{rdx} \rangle$ to arithmetic share form $[\![\mathsf{idx}]\!]$ and $[\![\mathsf{rdx}]\!]$ by B2A conversion [13].
(3) $\mathcal{S}$ and $\mathcal{R}$ compute $[\![\delta]\!] = [\![\mathsf{rdx}]\!] - [\![\mathsf{idx}]\!]$, and reveal δ in clear. They can share $\langle x \rangle = \langle \mathsf{C}[\mathsf{idx}] \rangle = \bigoplus_{i=0}^{m-1} \langle \mathsf{S}[i + \delta] \rangle \cdot \mathsf{C}[i]$.
(4) $\mathcal{S}$ and $\mathcal{R}$ convert $\langle x \rangle$ to $[\![x]\!]$ by B2A conversion.
(5) The parties convert $[\![x]\!]$ to its multiplicative sharing form $[\![x]\!]^*$ over $\mathbb{Z}_{N^2}$ as follows:
   (a) $\mathcal{S}$ samples $a \xleftarrow{\$} \mathbb{Z}_{N^2}$ and sends $(\textsc{GenBMT}, N^2, a, 0)$ to $\mathcal{F}_\mathrm{pre}$. $\mathcal{R}$ samples $b \xleftarrow{\$} \mathbb{Z}_{N^2}$ and sends $(\textsc{GenBMT}, N^2, 0, b)$ to $\mathcal{F}_\mathrm{pre}$. In the end, the parties obtain sharing $[\![c]\!]$ where $c = a \cdot b \pmod{N^2}$.
   (b) $\mathcal{S}$ samples $\gamma \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and sends $e \leftarrow \gamma^{-1} - a \pmod{N^2}$ to $\mathcal{R}$. $\mathcal{R}$ sends $f \leftarrow [\![x]\!]_\mathrm{r} - b \pmod{N^2}$ to $\mathcal{S}$.
   (c) $\mathcal{S}$ computes $[\![x \cdot \gamma^{-1}]\!]_\mathrm{s} \leftarrow [\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + a \cdot f + [\![c]\!]_\mathrm{s} \pmod{N^2}$ and $\mathcal{R}$ computes $[\![x \cdot \gamma^{-1}]\!]_\mathrm{r} \leftarrow e \cdot f + b \cdot e + [\![c]\!]_\mathrm{r} \pmod{N^2}$. $\mathcal{S}$ sends $[\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + [\![x \cdot \gamma^{-1}]\!]_\mathrm{s} \pmod{N^2}$ to $\mathcal{R}$.
   (d) $\mathcal{S}$ sets $[\![x]\!]_\mathrm{s}^* \leftarrow \gamma \pmod{N^2}$. $\mathcal{R}$ sets $[\![x]\!]_\mathrm{r}^* \leftarrow [\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + [\![x \cdot \gamma^{-1}]\!]_\mathrm{s} + [\![x \cdot \gamma^{-1}]\!]_\mathrm{r} \pmod{N^2}$.
(6) $\mathcal{R}$ samples $\beta \xleftarrow{\$} \mathbb{Z}_{2^{\ell_\mathsf{v}}}$, $\rho \xleftarrow{\$} [0, 2^\lambda)$ and computes $x_\beta \leftarrow [\![x]\!]_\mathrm{r}^* \cdot \mathsf{Enc}_\mathsf{pk}(\beta + \rho \cdot 2^{\ell_\mathsf{v}}) \pmod{N^2}$.
(7) $\mathcal{S}$ computes $[\![m]\!]_\mathrm{s} = \mathsf{Dec}_\mathsf{sk}([\![x]\!]_\mathrm{s}^* \cdot x_\beta) \pmod{2^{\ell_\mathsf{v}}}$, and $\mathcal{R}$ sets $[\![m]\!]_\mathrm{r} \leftarrow -\beta \pmod{2^{\ell_\mathsf{v}}}$.
(8) $\mathcal{S}$ and $\mathcal{R}$ run A2B conversion protocol to transform $[\![m]\!]$ over $\mathbb{Z}_{2^{\ell_\mathsf{v}}}$ to its boolean sharing form $\langle m \rangle$ over $\mathbb{Z}_2^{\ell_\mathsf{v}}$.

Figure 8: The HE-based SOS Protocol

Since $\mathcal{S}$ can compute $[\![x]\!]_\mathrm{s} \cdot \gamma^{-1} \pmod{N^2}$ by itself, the only issue is how to share the cross term $[\![x]\!]_\mathrm{r} \cdot \gamma^{-1} \pmod{N^2}$ securely. Indeed, this can be done with the help of a BMT of special form $a \cdot b = c \pmod{N^2}$ where $\mathcal{S}$ holds a, $\mathcal{R}$ holds b, and the parties share c.[4] With the BMT, $\mathcal{S}$ reveals $e \leftarrow \gamma^{-1} - a \pmod{N^2}$ and $\mathcal{R}$ reveals $f \leftarrow [\![x]\!]_\mathrm{r} - b \pmod{N^2}$, then $\mathcal{S}$ computes $[\![x \cdot \gamma^{-1}]\!]_\mathrm{s} \leftarrow [\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + a \cdot f + [\![c]\!]_\mathrm{s} \pmod{N^2}$ and $\mathcal{R}$ computes $[\![x \cdot \gamma^{-1}]\!]_\mathrm{r} \leftarrow e \cdot f + b \cdot e + [\![c]\!]_\mathrm{r} \pmod{N^2}$. $\mathcal{S}$ sends $[\![x]\!]_\mathrm{s} \cdot \gamma^{-1} + [\![x \cdot \gamma^{-1}]\!]_\mathrm{s} \pmod{N^2}$ to $\mathcal{R}$. In

[4] Any BMT $([\![a]\!], [\![b]\!], [\![c]\!])$ can be easily transformed to a special BMT by revealing a to $\mathcal{S}$ and b to $\mathcal{R}$, respectively. In $\mathcal{F}_\mathrm{pre}$, this can be simply done by letting $\mathcal{S}$ input $(a, 0)$ and $\mathcal{R}$ input $(0, b)$.

the end, $\mathcal{S}$ holds γ and $\mathcal{R}$ recovers $x \cdot \gamma^{-1} \pmod{N^2}$ by setting

$$x \cdot \gamma^{-1} = [\![x]\!]_s \cdot \gamma^{-1} + [\![x \cdot \gamma^{-1}]\!]_s + [\![x \cdot \gamma^{-1}]\!]_r \pmod{N^2}.$$

Now x is multiplicatively shared between parties over $\mathbb{Z}^*_{N^2}$.

Remark. Note that additive sharing is over $\mathbb{Z}_{N^2}$ whereas multiplicative sharing is over $\mathbb{Z}^*_{N^2}$, we show our conversion still works and give explanation. Specifically, $\mathbb{Z}^*_{N^2}$ includes all elements in $\mathbb{Z}_{N^2}$ except $\{0, p, 2p, \cdots, (p \cdot q^2 - 2) \cdot p, (p \cdot q^2 - 1) \cdot p, q, 2q, \cdots, (q \cdot p^2 - 2) \cdot q, (q \cdot p^2 - 1) \cdot q\}$. It is clear that if γ is accidentally sampled from $\mathbb{Z}_{N^2} \backslash \mathbb{Z}^*_{N^2}$, there will be no way to compute γ^{-1}. However, the bad probability of this accident is only $\frac{|\mathbb{Z}_{N^2} \backslash \mathbb{Z}^*_{N^2}|}{|\mathbb{Z}_{N^2}|} < \frac{p \cdot q^2 + q \cdot p^2 - 1}{N^2} < \frac{p \cdot q^2 + q \cdot p^2}{N^2} = \frac{1}{p} + \frac{1}{q}$, which is negligible. In our protocol, $\mathcal{S}$ knows p and q so can always select γ with inverse from $\mathbb{Z}^*_{N^2}$. This introduces indistinguishable difference following our prior argument. Therefore, our conversion works over $\mathbb{Z}_{N^2}$ correctly and securely except with negligible failing/distinguishable probability. Besides that, we do not differentiate $\mathbb{Z}_{N^2}$ and $\mathbb{Z}^*_{N^2}$.

Sharing Encrypted Message over $\mathbb{Z}_{2^{\ell_v}}$ Additively. For a ciphertext $x = \mathsf{Enc}_{\mathsf{pk}}(m)$ that is multiplicatively shared over $\mathbb{Z}_{N^2}$, *i.e.*, $\mathcal{S}$ has $[\![x]\!]^*_s$ and $\mathcal{R}$ has $[\![x]\!]^*_r$ such that $x = [\![x]\!]^*_s \cdot [\![x]\!]^*_r \pmod{N^2}$, the parties can explore homomorphic property of Paillier encryption to additively share $m \in \mathbb{Z}_{2^{\ell_v}}$. Note that $\mathcal{S}$ has Paillier public/secret key pair $(\mathsf{pk}, \mathsf{sk})$. Specifically, $\mathcal{R}$ computes and sends a randomized ciphertext $x_\beta \leftarrow [\![x]\!]^*_r \cdot \mathsf{Enc}_{\mathsf{pk}}(\beta + \rho \cdot 2^{\ell_v}) \pmod{N^2}$ to $\mathcal{S}$, where $\beta \xleftarrow{\$} \mathbb{Z}_{2^{\ell_v}}$ and $\rho \xleftarrow{\$} [0, 2^\lambda)$ are randomly sampled by $\mathcal{R}$.[5] $\mathcal{S}$ can compute $[\![x]\!]^*_s \cdot x_\beta = \mathsf{Enc}_{\mathsf{pk}}(m + \beta + \rho \cdot 2^{\ell_v})$ and decrypt it to learn $m + \beta \pmod{2^{\ell_v}}$. $\mathcal{R}$ has $-\beta \pmod{2^{\ell_v}}$. Obviously, $\mathcal{S}$ and $\mathcal{R}$ finally additively share m over $\mathbb{Z}_{2^{\ell_v}}$. The parties can perform A2B conversion to transform $[\![m]\!]$ over $\mathbb{Z}_{2^{\ell_v}}$ to boolean sharing $\langle m \rangle$ over $\mathbb{Z}^{\ell_v}_2$.

Security. We have Theorem 4 for the security of our HE-based SOS protocol. The proof can be found in Appendices A.2.

Theorem 3. *If Paillier encryption is semantically secure and N is computationally hard to factorize, the oblivious selection protocol in Fig. 8 securely computes the functionality $\mathcal{F}^{(\mathsf{M})}_{sos}$ in $(\mathcal{F}_{\mathrm{pre}})$-hybrid model under a semi-honest adversary.*

4.2 The Proposed PDTE Protocol

With our data structure, decision tree algorithm and SOS protocols, it is straightforward to design our PDTE protocol modularly. We show the protocol in Fig. 9 built on the top of an ideal SOS functionality.

PDTE Setup. P_0 and P_1 perform necessary work to setup SOS functionality for tree array $\mathsf{A}_\mathcal{T}$ and feature array $\mathcal{X}$. Moreover, P_0 shares $\mathsf{A}_\mathcal{T}[0]$ (*i.e.*, the root node) with P_1 as the evaluation starting node. From the property of boolean sharing, the parties can parse the root bit-by-bit to get the sharing of all attributes $\langle t \rangle$, $\langle l \rangle$, $\langle r \rangle$, $\langle v \rangle$, and $\langle c \rangle$.

PDTE Evaluation. In each iteration, P_0 and P_1 first call SOS functionality $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$ to share $\mathcal{X}[v]$. The parties then perform a secure comparison between $\langle \mathcal{X}[v] \rangle$ and $\langle t \rangle$ to compute a comparison result $\langle b \rangle$. The evaluation can then decide which child becomes the next evaluation node by employing a MUX computation: $\langle \mathsf{idx} \rangle \leftarrow \langle l \rangle \oplus \langle b \rangle \cdot (\langle l \rangle \oplus \langle r \rangle)$. That is, the parties share $\langle \mathsf{idx} \rangle = \langle l \rangle$ if $b = 1$, otherwise $\langle \mathsf{idx} \rangle = \langle r \rangle$.

From the shared index idx, the parities invoke $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$ to share $\mathsf{A}_\mathcal{T}[\mathsf{idx}]$. $\langle t \rangle$, $\langle l \rangle$, $\langle r \rangle$, $\langle v \rangle$, $\langle c \rangle$ are then updated correspondingly. Besides, $\langle c \rangle$ is stored in $\langle \mathsf{rst} \rangle$ where the final classification label will stay in. Note that we encode a self-loop for each leaf node, thus $\langle \mathsf{rst} \rangle$ will always hold a correct classification label once the evaluation reaches a leaf node. Moreover, it is easy to hide length information: P_0 and P_1 just run d iterations of evaluation. In the end, P_0 sends $\langle \mathsf{rst} \rangle_0$ to P_1, and P_1 recovers rst as classification result.

The protocol runs in $O(d)$ iterations with d secure comparison and MUX operations. If the OT-based SOS protocol is used over $\mathcal{X}$ and a sublinear SOS protocol is used over $\mathsf{A}_\mathcal{T}$, then the total communication complexity is $O(n\ell d)$.

Optimization 1 - Reduce SOS Invocations. We can reduce the number of SOS invocations by exploring a data locality property in decision tree evaluation. Our observation is that tree evaluation will only go from a parent to one of its children. Therefore, we can pack the parent with its children together as a bigger node to reduce invocations of oblivious selection. We call the packed node as a *cluster*. If the parent node is a leaf, $\mathcal{S}$ needs to allocate two dummy nodes to make the cluster's size indistinguishable from others. The parties then use oblivious selection to share the desired cluster

Parameters: Computational security parameter κ; P_0 provides a tree $\mathcal{T}$; P_1 provides a feature vector $\mathcal{X}$; the longest tree depth d.

[Setup]

(1) P_0 encodes its decision tree $\mathcal{T}$ to an array $\mathsf{A}_\mathcal{T}$.
(2) P_0 and P_1 invoke functionality $\mathcal{F}^{(\mathsf{A}_\mathcal{T})}_{\mathrm{SOS}}$. P_0 sends $(\textsc{Setup}, \mathsf{A}_\mathcal{T}, \ell_v)$ to $\mathcal{F}^{(\mathsf{A}_\mathcal{T})}_{\mathrm{SOS}}$ and P_1 sends $(\textsc{Setup})$ to $\mathcal{F}^{(\mathsf{A}_\mathcal{T})}_{\mathrm{SOS}}$.
(3) P_0 and P_1 invoke functionality $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$. P_1 sends $(\textsc{Setup}, \mathcal{X}, \ell)$ to $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$, and P_0 sends $(\textsc{Setup})$ to $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$.
(4) P_0 shares root node $\mathsf{A}_\mathcal{T}[0]$ with P_1. Both parties parse $\langle \mathsf{A}_\mathcal{T}[0] \rangle$ as $\langle t \rangle || \langle l \rangle || \langle r \rangle || \langle v \rangle || \langle c \rangle$.

[Evaluation]

(1) For $i \in [1, d]$
  (a) P_0 sends $(\textsc{Eval}, \langle v \rangle_s)$ and P_1 sends $(\textsc{Eval}, \langle v \rangle_r)$ to $\mathcal{F}^{(\mathcal{X})}_{\mathrm{SOS}}$. In the end, P_0 and P_1 share $\langle \mathcal{X}[v] \rangle$.
  (b) P_0 and P_1 run secure comparison protocol to compute $\langle b \rangle \leftarrow \langle \mathcal{X}[v] \rangle > \langle t \rangle$.
  (c) P_0 and P_1 compute index of next tree node $\langle \mathsf{idx} \rangle \leftarrow \langle l \rangle \oplus \langle b \rangle \cdot (\langle l \rangle \oplus \langle r \rangle)$.
  (d) P_0 sends $(\textsc{Eval} \langle \mathsf{idx} \rangle_0)$, and P_1 sends $(\textsc{Eval}, \langle \mathsf{idx} \rangle_1)$ to $\mathcal{F}^{(\mathsf{A}_\mathcal{T})}_{\mathrm{SOS}}$. In the end, P_0 and P_1 share $\langle \mathsf{A}_\mathcal{T}[\mathsf{idx}] \rangle$.
  (e) parse $\langle t \rangle || \langle l \rangle || \langle r \rangle || \langle v \rangle || \langle c \rangle \leftarrow \langle \mathsf{A}_\mathcal{T}[\mathsf{idx}] \rangle$.
  (f) set $\langle \mathsf{rst} \rangle \leftarrow \langle c \rangle$.
(2) P_0 and P_1 reveal rst to P_1 as output.

Figure 9: Our PDTE Protocol

[5] We use Paillier's plaintext domain $\mathbb{Z}_N$ to hold messages of length ℓ_v where $2^{\ell_v} \ll |N|$. Here $\rho \cdot 2^{\ell_v}$ is used for statically hiding $m + \beta$, meanwhile still allows $\mathcal{S}$ to compute $m + \beta \pmod{2^{\ell_v}}$ correctly.

between parties. Since our SOS protocol supports SIMD mode, the parties can share a cluster by only one invocation, whereas it requires two in the original protocol. In this way, we reduce PDTE evaluation invocation from d to $\lceil d/2 \rceil$. We can generalize the idea to pack a parent node with its descendants in the following q layers, reducing invocations from d to $\lceil d/q \rceil$.

The remaining issue is how to traverse within a cluster obliviously. This can be done by q MUX operations, and each is over two smaller sub-trees. However, the total communication for traversing within a cluster will be $O(2^q)$. In practice, we can set $q = 2$ or 3 to reduce 50% or 67% rounds from SOS protocol while not increasing communication too much.

Optimization 2 - Reduce Local Computation. In our PDTE protocol, each oblivious selection causes a linear scan over the whole decision tree, incurring $O(d \cdot m)$ computation in total. We can reduce the overhead when $\mathcal{T}$ is a complete tree. That is, instead of performing the scan over all nodes, the parties only need to run oblivious selection over i-th layer of $\mathcal{T}$ for the i-th iteration of evaluation. Therefore, the total local computation from SOS will only be $O(m)$. Note that we can not use this optimization directly over sparse trees; otherwise, $\mathcal{R}$ can learn the tree structure of each layer, *e.g.*, number of nodes of each layer. Nevertheless, we can always transform a non-complete tree into a complete one by padding dummy nodes and then we can optimize the padded tree. In practice, padding is cost-effective for those near-complete trees but not for sparse trees.

Security. We have Theorem 4 towards security of our PDTE protocol. The proof can be found in Appendices A.3.

Theorem 4. *The PDTE protocol in Fig. 9 securely computes the functionality* $\mathcal{F}_{\text{DT}}$ *in* $(\mathcal{F}_{\text{SOS}}^{(\mathsf{A}_{\mathcal{T}})}, \mathcal{F}_{\text{SOS}}^{(X)})$*-hybrid model against semi-honest adversary.*

5 EXPERIMENT

In this section, we report the concrete efficiency of our PDTE protocol. We implement the protocol in C++ under ABY framework [13].

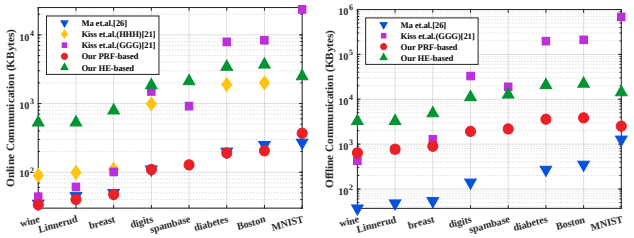
5.1 Experiment Setup

We run our experiment on a desktop PC equipped with Intel(R) Core™ i9-9900 CPU at 3.10 GHz × 16 running Ubuntu 20.04 LTS and 32 GB of memory. We use Linux tc tool to simulate local-area network (LAN, RTT: 0.1 ms, 1 Gbps), metropolitan-area network (MAN, RTT: 6 ms, 100 Mbps) and wide-area network (WAN, RTT: 80 ms, 40 Mbps). We set the computational security parameter $\kappa = 128$ and statistical security parameter $\lambda = 40$. As in prior work, we set the bit length to $\ell = 64$. For AHE, the plaintext module is $|N| = 2048$. We implement the involved secure computation using GMW [28] protocol over boolean sharing as default. The times reported are averaged over ten trials.

5.2 Tree Parameters

We evaluate our protocols on 8 representative datasets from UCI repository[6] as listed in Table 3. To compare with [21, 26], we directly use their used decision trees *wine, Linnerud, breast, digits, diabetes* and *Boston* which are trained using codes from [21]. We additionally train two trees, one is a deep-but-sparse tree *spmabase* and another is a density tree *MNIST* with a high-dimensional vector.

[6]https://archive.ics.uci.edu/ml



(a) Online Communication (b) Offline Communication

Figure 10: Online and Offline Communication Cost. Note that the y-axis is in logarithm scale.

Table 3: Tree Parameters

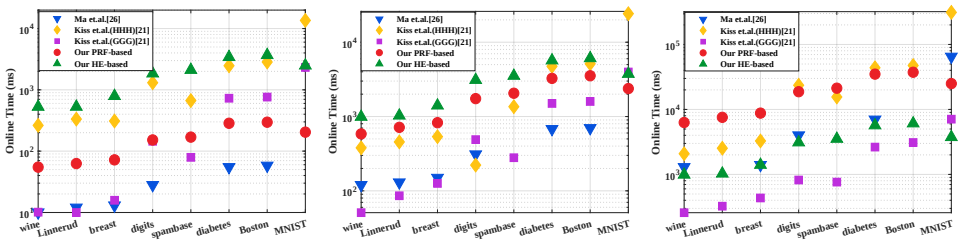
Decision Tree	Feature Dimension n	Depth d	#(Nodes) m
wine	7	5	23
Linnerud	3	6	39
breast	12	7	43
digits	47	15	337
spambase	57	17	171
diabetes	10	28	787
Boston	13	30	851
MNIST	784	20	4179

5.3 Performance Evaluation

In this section, we report the efficiency of our protocol. We first test PDTE protocols in communication and running time under different network settings and compare them with state-of-the-art PDTE protocols. Then we discuss trade-off by exploring the modular design of our PDTE design and give recommendations for different scenarios. Last we report performances over large synthetic deep trees to show the scalability of our PDTE protocols.

We mainly compare our protocols with three representative PDTE works [21, 26, 33]. Kiss *et al.* [21] divide a PDTE protocol into three sub-protocols: feature selection, comparison and path evaluation and use either Garbled Circuit (GC) or AHE to instantiate them. We select [21] because this work is the most summative in linear-cost PDTE protocols. We compare our work with their GGG and HHH since the former is computation-friendly and the latter is communication-friendly. Other two PDTE works are both sublinear-cost [26, 33] protocols, similar to ours. But only the ORAM-based PDTE protocol in [33] is truly sublinear-communication.

PDTE Communication. Fig. 10 details the communication consumption of our two PDTE protocols, and the comparison with [21, 26]. As we can see, our PRF-based construction (with LowMC as the PRF instantiation) requires the least communication among these PDTE protocols and is slightly better than [26]. Our HE-based PDTE protocol requires more communication than PRF-based one. The main reason is that the ciphertext size evaluated in each round in our PRF-based protocol is smaller than that in our HE-based protocol. In the latter, for example, the ciphertext size is set to



(a) UCI datasets on LAN (1Gbps/0.1ms) (b) UCI datasets on MAN (100Mbps/6ms) (c) UCI datasets on WAN (40Mbps/80ms)

Figure 11: Online Runtime in LAN/MAN/WAN Setting. Note that the y-axis is in logarithm scale.

be $|N^2| = 4096$. Although they all enjoy constant communication complexity per selection, the constant factor is much higher in HE-based protocol. GGG also shows better online communication performance than our HE-based protocol when trees are small. It is reasonable because GGG shifts all GC generation to the offline phase, resulting in efficient online efficiency independent of the tree size. However, GGG needs $n\ell$ 1-out-of-2 OT to perform oblivious feature selection. Thus, compared with GGG, our HE-based protocol shows less online communication cost when it comes to trees with a high-dimensional feature vector, like MNIST.

Towards offline communication, our two constructions both lie between GGG and the construction of Ma *et al.* [26]. The protocol in [26] enjoys the lowest offline communication cost for small trees. However, their offline communication cost increases with the size of the tree since the tree in their protocol should be re-sent before each evaluation. Thus, as the tree size m increases, our offline communication overhead will be outpaced by [26]. GGG costs the most even when the tree is medium-sized in Table 3, *i.e.*, digits. As we discussed before, GGG can enjoy a better online communication cost by moving major communication to offline. However, as shown in Fig. 10(b), the price is high, referring to big trees.

PDTE Running Time. We report the online running time of our PDTE protocols under different network settings (LAN, MAN, WAN) in Fig. 11. The reported runtimes of protocol [26] are read from their paper. In this test, we use LowMC to instantiate the involved PRF in our PRF-based protocol.

In the LAN setting, our HE-based protocol needs the most running time except for MNIST. It is clear to see that the running time of GGG and HHH grow with the tree size. Our HE-based protocol shows less computation when treating MNIST ($m = 4179$) than Boston ($m = 851$). This is because the depth of MNIST is smaller than Boston. Our PRF-based protocol lies between Ma *et al.* [26] protocol and HHH protocol. There is no doubt that protocol in [26] is the most efficient protocol to date under the LAN setting. This is because the most expensive operations in their protocol are OT and GC. Yet, in our PRF-based protocol, the most costly are secure LowMC evaluation. However, our PRF-based protocol still outruns linear protocols GGG and HHH by 24× to 65×, respectively, for MNIST.

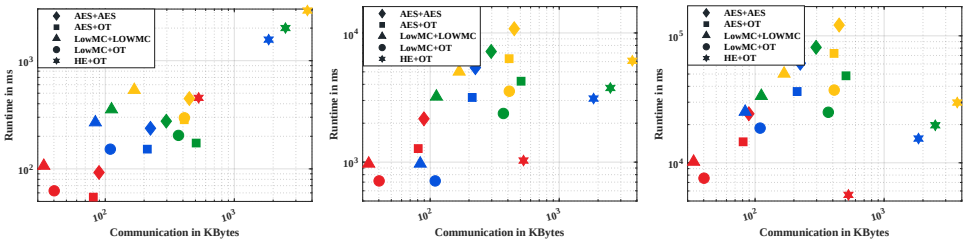
Our two protocols are slightly less efficient than HHH and require around 100× costs than GGG [26], especially when the tree is tiny. One reason is that our work is based on GMW who is highly influenced by latency. Thus our PRF-based protocol runs an order slower when the latency increases from 0.1ms to 6ms. This "negative" property appears on our HE-based protocol as well but in a mild influence since our HE-based protocol removes the costly LowMC operation. One should notice that our two protocols become slightly better than GGG and save at least 5× running times than HHH, for MNIST. With the increase of size/depth of the tree, our sublinear protocols will be more competitive.

We can also observe that as the network latency increases, our HE-based PDTE protocol gradually surpasses PRF-based one, *i.e.*, 6× to 7× faster in the WAN setting. The reason is that two-party PRF evaluation in the PRF-based protocol involves higher rounds than the HE-based construction, which causes significant time consumption over the high-latency network.

Trade-off. We report the concrete trade-off between our PRF-based protocol and HE-based protocol. Since OT/PRF/HE can be used to instantiate SOS protocol, we investigate the differences when using different SOS protocols and give the corresponding experiments. To thoroughly examine the PRF-based protocol, we also implement AES as the underlying PRF. Specifically, we use A+B to denote the tree node is selected by A-based SOS protocol, and the attribute is obliviously shared by B-based SOS protocol.

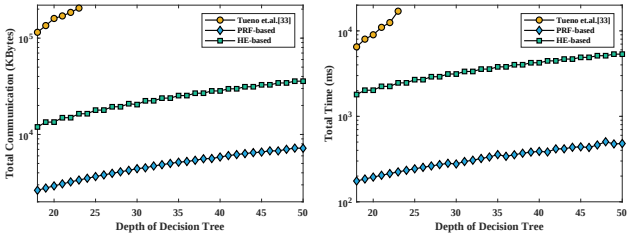
We remove wine, breast, spambase and diabetes, which share similar parameters as the selected trees. Fig. 12 shows the trade-off in three different network settings. In the scenarios with low network latency (*e.g.*, IoT), LowMC+OT can efficiently handle small trees like Linnerud. In the cases of deep trees with thousands of nodes, LowMC+LowMC shows less communication cost while LowMC+OT saves roughly 50% runtime. Thus, in the situation where the computation performance matters more, LowMC+OT can be adopted to provide reasonable online computation overhead and communication cost. The reported runtime/communication of HE+OT under the LAN setting is significantly high than other PRF-based protocols. With latency increasing, AES-based protocols suffer more than LowMC involved protocols, while HE+OT shows its advantages in runtime, as seen from Fig. 12(b). Under WAN with 80ms high network latency, the online running time for small trees by LowMC+LowMC is around 50% higher than LowMC combining OT. This gap becomes progressively smaller as the tree depth increases. Under this setting, HE+OT is the first choice when considering time-consuming.

Scalability for High-depth Trees. We report the scalability of our PDTE protocols and compare them with the ORAM-based PDTE protocol [33] in LAN setting. Unfortunately, Tueno *et al.* [33] only



(a) Trade-off under LAN (1Gbps/0.1ms) (b) Trade-off under MAN (100Mbps/6ms) (c) Trade-off under WAN (40Mbps/80ms)

Figure 12: Trade-offs between Online Communication (x-axis) and Online Runtime (y-axis). Each figure shows the online complexities including communication and runtime. Note that diamond, square, triangle, round and six-pointed star represent AES+AES, AES+OT, LowMC+LowMC, LowMC+OT and HE+OT protocol, respectively. The shape filled by red, blue, yellow and green are Linnerud, digits, Boston and MNIST, respectively. Both x-axis and y-axis are in logarithm scale.



(a) Total Communication (b) Total Running Time

Figure 13: Total Communication and Runtime Cost (LAN setting). Note that y-axis is in logarithm scale.

report results for trees with depth up to 23 since their experiments ran out of memory. The experiments of our protocols are done for trees with the depth ranging from 18 to 50. For tree's size, we follow the same setting of [33] by letting $m = 25d$. We perform the evaluation in the LAN setting as shown in Fig. 13. As we can see, our HE-based protocol requires 9× to 16× less communication and 4× to 7× less running time compared with [33]. Our PRF-based protocol requires 43× to 60× less communication and 26× to 54× less running time compared with [33]. Therefore, our PDTE protocols are more scalable for evaluation over large trees.

6 RELATED WORK

There are many two-party PDTE protocols in the literature [2, 5, 9, 21, 32, 35]. Among them, a few works [20, 26, 33] achieve sublinear cost. In the following, we summarize these works separately.

Linear-cost Protocols. Brickell *et al.* [9] pack each tree node into a GC circuit and then transmit the encrypted tree itself to the feature provider. With the help of HE and OT, the feature provider can perform oblivious evaluation by herself. However, the security requires that the encrypted tree should be refreshed for each tree evaluation, incurring linear costs. This work was later optimized by Barni *et al.* [2]. The communication cost is saved by only sending encrypted internal nodes rather than the whole tree. However, it is still linear to the tree size. Bost *et al.* [5] express the decision tree as a high-degree polynomial and encrypt it using an expensive Leveled Fully Homomorphic Encryption (Leveled-FHE) scheme. Wu *et al.* [35] propose a cheaper protocol by only relying on OT and AHE. Yet, the construction requires the tree holder to pad the tree to be complete in order to hide the tree structure. This padding strategy incurs massive communication and computation overhead, especially for deep-but-sparse trees. Subsequent PDTE protocols [12, 21, 25] follow the same approach. To avoid padding, Tai *et al.* [32] propose a novel path cost mechanism using AHE. This protocol performs better for sparse trees but still runs at linear cost. Kiss *et al.* [21] systematically compare existing PDTE protocols. They mix HE and GC in different evaluation phases and report their concrete efficiency. Same as previous works, they trade efficiency for privacy by doing comparisons for all decision nodes, resulting in linear computation/communication. Ideally, the best solution is to perform only the necessary comparisons meanwhile hiding the decision path.

Sublinear-cost Protocols. Three up-to-date work [20, 26, 33] consider sub-linear decision tree protocols. Joye and Salehi [20] reduce the number of secure comparisons to d. The comparison is based on DGK protocol [11] using AHE. In tree level l, they employ 1-out-of-2^l OT to obliviously select an AHE encrypted tree node. In the end, the involved OT incurs $O(2^d - 1)$ communication in total. Thus, this PDTE protocol is only sublinear in computation. Tueno *et al.* [33] organize a decision tree as an array and build an oblivious array indexing (OAI). Such an interactive OAI allows the participants to pick the desired tree node and its corresponding attribute obliviously. OAI can be instantiated utilizing GC, OT or Oblivious RAM (ORAM). The first two OAIs can only realize sublinear complexity on the feature provider side since they also require OT to transfer among 2^d nodes, same as [20]. If employing ORAM, it takes $O(d^4)$ communication cost and requires d^2 (*e.g.*, complete tree) rounds.

In order to further reduce communication cost, in protocol [26], the tree holder encrypts the tree and sends it to the feature provider. In each tree level, there is an OT and a comparison between both parties. The feature provider searches local encrypted tree for the next node after comparison. Since the authors move the most expensive oblivious selection operations to feature provider's local computation, their protocol is very efficient in terms of computation. Nevertheless, we notice that this searching property means this tree cannot be reused across evaluations because the feature provider can learn some information from memory access patterns during

different evaluations. Accordingly, to reach the genuine PDTE when using [26], toward every evaluation, the decision tree model should be re-randomized and transmitted to the feature provider, causing linear complexity in terms of communication and computation.

Outsourced Protocols. Some also try to gain more efficiency with the help of cloud servers, which is called outsourced PDTE protocols [19, 24, 26, 36]. They aim to use outsourced cloud servers to release the heavy burden from the tree holder and the feature provider. However, most of them suffer linear complexities [24, 36] or leak more information [24] than prior two-party protocols.

7 CONCLUSION

In this paper, we study how to design sublinear-communication PDTE protocols with improved efficiency. We first propose two communication-efficient shared oblivious selection (SOS) protocols with different trade-offs. By combining these SOS protocols with secure computation and a tree encoding strategy, we propose two PDTE protocols both with sublinear communication efficiency. Our experiments show our protocols are efficient and practical. As future research, we will extend our techniques to other privacy-preserving machine learning protocols.

ACKNOWLEDGMENT

We thank the anonymous reviewers for insightful comments and suggestions. Bai and Russello would like to acknowledge the MBIE-funded programme STRATUS (UOWX1503) for its support and inspiration for this research. This research is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 430–454.

[2] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2009. Secure evaluation of private linear branching programs with medical applications. In *European symposium on research in computer security*. Springer, 424–439.

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

[4] Donald Beaver. 1995. Precomputing oblivious transfer. In *Annual International Cryptology Conference*. Springer, 97–109.

[5] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS)*, Vol. 4324. 4325.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 337–367.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1292–1303.

[8] Andrej Bratko, Bogdan Filipič, Gordon V Cormack, Thomas R Lynam, and Blaž Zupan. 2006. Spam filtering using statistical data compression models. *The Journal of Machine Learning Research* 7 (2006), 2673–2698.

[9] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. 2007. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM conference on Computer and communications security*. 498–507.

[10] Jason Catlett. 1991. Overprvning Large Decision Trees.. In *International Joint Conferences on Artificial Intelligence*. Citeseer, 764–769.

[11] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. 2007. Efficient and secure comparison for on-line auctions. In *Australasian conference on information security and privacy*. Springer, 416–430.

[12] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson CA Nascimento, Wing-Sea Poon, and Stacey Truex. 2017. 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 (2017), 217–230.

[13] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *Network and Distributed System Security Symposium (NDSS)*.

[14] Jack Doerner and Abhi Shelat. 2017. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 523–535.

[15] Niv Gilboa and Yuval Ishai. 2014. Distributed point functions and their applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 640–658.

[16] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[17] Carmit Hazay and Yehuda Lindell. 2010. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media.

[18] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*. Springer, 145–161.

[19] Keyu Ji, Bingsheng Zhang, Tianpei Lu, Lichun Li, and Kui Ren. 2021. UC Secure Private Branching Program and Decision Tree Evaluation. *Cryptology ePrint Archive* (2021).

[20] Marc Joye and Fariborz Salehi. 2018. Private yet efficient decision tree evaluation. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 243–259.

[21] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider. 2019. Sok: Modular and efficient private decision tree evaluation. *Proceedings on Privacy Enhancing Technologies* 2019, 2 (2019), 187–208.

[22] Hian Chye Koh, Wei Chin Tan, and Chwee Peng Goh. 2006. A two-step method to construct credit scoring models with data mining techniques. *International Journal of Business and Information* 1, 1 (2006), 96–118.

[23] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT extension for transferring short secrets. In *Annual Cryptology Conference*. Springer, 54–70.

[24] Jinwen Liang, Zheng Qin, Sheng Xiao, Lu Ou, and Xiaodong Lin. 2019. Efficient and secure decision tree classification for cloud-assisted online diagnosis services. *IEEE Transactions on Dependable and Secure Computing* 18, 4 (2019), 1632–1644.

[25] Lin Liu, Jinshu Su, Rongmao Chen, Jinrong Chen, Guangliang Sun, and Jie Li. 2019. Secure and fast decision tree evaluation on outsourced cloud data. In *International Conference on Machine Learning for Cyber Security*. Springer, 361–377.

[26] Jack PK Ma, Raymond KH Tai, Yongjun Zhao, and Sherman SM Chow. 2021. Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In *Network and Distributed System Security Symposium (NDSS)*.

[27] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, and Ximeng Liu. 2019. Privacy-preserving and high-accurate outsourced disease predictor on random forest. *Information Sciences* 496 (2019), 225–241.

[28] Silvio Micali, Oded Goldreich, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM, 218–229.

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

[30] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *30th USENIX Security Symposium (USENIX Security 21)*. 2165–2182.

[31] Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. 2002. Decision trees: an overview and their use in medicine. *Journal of medical systems* 26, 5 (2002), 445–463.

[32] Raymond KH Tai, Jack PK Ma, Yongjun Zhao, and Sherman SM Chow. 2017. Privacy-preserving decision trees evaluation via linear functions. In *European Symposium on Research in Computer Security*. Springer, 494–512.

[33] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. 2019. Private Evaluation of Decision Trees using Sublinear Cost. *Proc. Priv. Enhancing Technol.* 2019, 1 (2019), 266–286.

[34] Xiao Wang, Hubert Chan, and Elaine Shi. 2015. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 850–861.

[35] David J Wu, Tony Feng, Michael Naehrig, and Kristin E Lauter. 2016. Privately Evaluating Decision Trees and Random Forests. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 335–355.

[36] Yifeng Zheng, Huayi Duan, and Cong Wang. 2019. Towards secure and efficient outsourcing of machine learning classification. In *European Symposium on Research in Computer Security*. Springer, 22–40.

A SECURITY PROOFS

A.1 Proof of Theorem 2

Security against corrupted $\mathcal{S}$. We construct a simulator $\mathsf{Sim_s}$ as follows. During setup, $\mathsf{Sim_s}$ runs $\mathcal{S}$'s setup protocol except with the following exception: $\mathsf{Sim_s}$ randomly samples $\mathsf{C}[i] \xleftarrow{\$} \{0,1\}^{\ell_v}$ for all $i \in [0,m)$. As for selection protocol, $\mathsf{Sim_s}$ is provided with input $\langle \mathsf{idx} \rangle_\mathsf{s}$ and output e (*i.e.*, a PRF output share). It runs $\mathcal{S}$'s selection protocol except with the following exceptions: $\mathsf{Sim_s}$ runs simulator for $\mathcal{F}_{\mathsf{pre}}$ to simulate the view of weight-1 bit vector generation. Also, $\mathsf{Sim_s}$ runs simulator for $\mathcal{F}_{\mathsf{pre}}$ to simulate the view of shared PRF protocol. Finally, $\mathsf{Sim_s}$ outputs $\mathcal{S}$'s view. We prove the simulated view is indistinguishable from real-world execution via a sequence of hybrid games.

- H_0: Outputs $\mathcal{S}$'s view in the real-world protocol.
- H_1: Same as H_0 except $\mathsf{Sim_s}$ randomly samples $\mathsf{C}[i] \xleftarrow{\$} \{0,1\}^{\ell_v}$ for all $i \in [0,m)$. By the security of F, H_1 is computationally indistinguishable from H_0.
- H_2: Same as H_1 except that $\mathsf{Sim_s}$ runs simulator for $\mathcal{F}_{\mathsf{pre}}$ to generate $\mathcal{S}$'s view in the shared weight-1 bit vector protocol. In particular, the simulator for $\mathcal{F}_{\mathsf{pre}}$ is provided with input n and output $\langle \mathsf{S} \rangle_\mathsf{s}$, $\langle \mathsf{idx} \rangle_\mathsf{s}$, and in the end it generates a simulated view. By the security of shared weight-1 bit vector protocol, H_2 and H_1 are indistinguishable.
- H_3: Same as H_2 except that $\mathsf{Sim_s}$ runs simulator to generate $\mathcal{S}$'s view in shared PRF protocol. Note that for all other view from running secret-shared computation, $\mathsf{Sim_s}$ can simulate it by randomly sampling shares, which is identically distributed in both worlds. However, $\mathsf{Sim_s}$ must ensure all prior simulated view be consistent with the final output, which is $\langle \mathsf{M}[\mathsf{idx}][j] \rangle_\mathsf{s}$. As such, $\mathsf{Sim_s}$ computes $f_j^* \leftarrow \langle \mathsf{M}[\mathsf{idx}][j] \rangle \oplus \langle \mathsf{C}[\mathsf{idx}][j] \rangle^*$ for $j \in [0,B)$, it then invokes the simulator for $\mathcal{F}_{\mathsf{pre}}$ over $(\langle \{\mathsf{idx}||j\rangle_\mathsf{s}^*\}_{j\in[0,B-1)}, \{f_j^*\}_{j\in[0,B)})$ to simulate $\mathcal{S}$'s view in shared PRF protocol. Since we are working in a hybrid model, such simulator must exist[7]. H_3 and H_2 are indistinguishable by the security of shared PRF protocol.

Security against corrupted $\mathcal{R}$. Operations of $\mathcal{S}$ and $\mathcal{R}$ are almost symmetric during the protocol, so our strategy for proving a corrupted $\mathcal{R}$ is almost same as the one we construct against corrupted $\mathcal{S}$. The proof for corrupted $\mathcal{R}$ is thus omitted.

A.2 Proof of Theorem 3

Security against corrupted $\mathcal{S}$. We construct a simulator Sim for corrupted $\mathcal{S}$ as follows. During setup, Sim runs $\mathcal{S}$'s setup protocol except with the following exception: Sim randomly samples $\mathsf{C}[i] \xleftarrow{\$} \mathbb{Z}_{N^2}$ for all $i \in [0,m)$. Given semantic security of Paillier encryption, the simulated ciphertexts are indistinguishable from the ones in real-world protocol.

In the selection protocol, for the view from generating weight-1 bit vector, Sim calls the simulator for $\mathcal{F}_{\mathsf{pre}}$ to simulate corresponding view over randomly sampled $\langle \mathsf{S} \rangle_\mathsf{s}$ and $\langle \mathsf{rdx} \rangle_\mathsf{s}$[8]; the simulation is perfect in hybrid mode. The view of B2A conversion can also be simulated using existing simulator for secure computation, such a simulator always exists given the security of B2A protocol. Next, Sim picks a random $\delta \xleftarrow{\$} \mathbb{Z}_m$, the simulation is also indistinguishable: in the real-world protocol, idx and rdx are all random, then $\delta = \mathsf{rdx} - \mathsf{idx} \pmod m$ is also a random number in $\mathbb{Z}_m$. The simulator Sim performs simulation for the next B2A conversion protocol, similarly as it previously does for idx and rdx. The simulated view of B2A is indistinguishable from real-world protocol. For the view of share conversion from additive sharing to multiplicative sharing, Sim first invokes the simulator for BMT generation in $\mathcal{F}_{\mathsf{sprf}}$ over a randomly sampled $a \xleftarrow{\$} \mathbb{Z}_{N^2}$ and the share $[\![c]\!] \xleftarrow{\$} \mathbb{Z}_{N^2}$. Next, Sim samples γ, e, f and $[\![x \cdot \gamma^{-1}]\!]$ randomly from $\mathbb{Z}_{N^2}$. Note that in the real-world protocol, $e \leftarrow \gamma^{-1} - a \pmod{N^2}$ and $f \leftarrow [\![x]\!]_\mathrm{r} - b \pmod{N^2}$. Given that a, b, γ and $[\![x]\!]_\mathrm{r}$ are all random, then e and f are random elements in $\mathbb{Z}_{N^2}$ as well. However, there is a difference since the simulator samples γ from $\mathbb{Z}_{N^2}$ rather from $\mathbb{Z}_{N^2}^*$, the simulated γ can be accidentally sampled from $\mathbb{Z}_{N^2} \backslash \mathbb{Z}_{N^2}^*$. However, this bad probability can only happens with probability of $\frac{1}{p} + \frac{1}{q}$, which is negligible in κ. Next, Sim randomly samples $x_\beta \xleftarrow{\$} \mathbb{Z}_{N^2}$ rather than $x_\beta \xleftarrow{\$} \mathbb{Z}_{N^2}^*$, the probability to distinguish is negligible as we argued before. For $[\![m]\!]_\mathsf{s}$, Sim will randomly sample it from $[0, 2^{\ell_v+\lambda+1}]$, this is statically closed to the real-world view for statistical parameter λ.

Security against corrupted $\mathcal{R}$. Simulator for a corrupted $\mathcal{R}$ can be constructed using the similar strategy as we constructed for $\mathcal{S}$ since all operations are symmetric between $\mathcal{S}$ and $\mathcal{R}$. Therefore, we omit the simulation for corrupted $\mathcal{R}$ in our proof.

A.3 Proof of Theorem 4

Simulator for corrupted P_0. The simulator Sim invokes the simulator of SOS setup protocol over $\mathsf{A}_\mathcal{T}$. Similarly, Sim invokes the simulator of SOS setup protocol over $\mathcal{X}$. The simulation is perfect in the hybrid model. Sim randomly samples $\langle t \rangle_0, \langle l \rangle_0 \langle r \rangle_0, \langle v \rangle_0$ and $\langle c \rangle_0$ from $\mathbb{Z}_{2^\ell}$. It is straightforward to see that the simulated view is indistinguishable from the real-world execution.

As for evaluation protocol, the simulator in the beginning does not need to simulate $\langle \mathsf{idx} \rangle$ since it shares 0, which is locally done by the parties. Then Sim randomly samples $\langle \mathsf{rst} \rangle_0$. The above simulation is indistinguishable from the real-world view. Then Sim calls the simulator of SOS selection protocol over $\mathcal{X}$. In particular, Sim samples $\langle v \rangle_0$ and $\langle \mathcal{X}[v] \rangle$ randomly, and invokes $\mathcal{F}_{\mathsf{sos}}^{(\mathcal{X})}$ where $\langle v \rangle_0$ and $\langle \mathcal{X}[v] \rangle$ are the input and the output, respectively. Afterwards, Sim simulates the view of secure comparison, the view can be simulated as the secure computation is well-studied in existing work [13]. Then Sim randomly samples $\langle \mathsf{idx} \rangle \in \mathbb{Z}_{2^\ell}$, the simulation is perfect since the share is randomly computed from secure computation. Sim invokes the simulator for SOS selection protocol over $\mathsf{A}_\mathcal{T}$ with $\langle \mathsf{idx} \rangle_\mathsf{s}$ as the input and a random number $r \xleftarrow{\$} (\mathbb{Z}_{2^{\ell_v}})^5$ as the output. In particular, r is used to simulate $\mathcal{S}$'s share of $\mathsf{A}_\mathcal{T}[\mathsf{idx}]$. The simulated view is indistinguishable from real-world view due to the security of SOS protocol. For all other operations that can be

[7] A simulator who can simulate secret-shared XOR and AND computation will suffice to simulate any secure computation task.

[8] Note that a secret share can be regards as a random number over its sharing domain. Then, the sampled random numbers as the simulated shares are indistinguishable from the real-world protocol.

Parameters: BMT module n, computational security parameter κ; statistical security parameter λ; Paillier plaintext module $N = p \cdot q$, where p and q are primes.

(1) P_0 generates a pair of Paillier public/private key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and sends pk to P_1.
(2) P_0 chooses $a \xleftarrow{\$} \mathbb{Z}_n$ and sends $x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a)$ to P_1.
(3) P_1 chooses $b \xleftarrow{\$} \mathbb{Z}_n$ and $\rho \xleftarrow{\$} [0, 2^\lambda)$, sends $x' \leftarrow x^b \cdot \mathsf{Enc}_{\mathsf{pk}}(r + \rho \cdot n) \pmod{N^2}$ to P_0.
(4) P_0 decrypts to get $[\![c]\!]_0 \leftarrow \mathsf{Dec}_{\mathsf{sk}}(x') \pmod{n}$.
(5) P_1 sets $[\![c]\!]_1 \leftarrow -r \pmod{n}$.

Figure 14: BMT from AHE [13]

Parameters: BMT module n, n's bit size ℓ; computational security parameter κ.

(1) P_0 chooses $a \xleftarrow{\$} \mathbb{Z}_n$ and decomposes a to its boolean form $(a_{\ell-1}, \cdots, a_1, a_0)$ such that $a = \sum_{i=0}^{\ell-1} 2^i \cdot a_i$.
(2) For $0 \le i < \ell$:
   (a) P_1 chooses $r_i \xleftarrow{\$} \mathbb{Z}_n$ and computes two messages $(m_i^0 = r_i, m_i^1 = r_i + 2^i \cdot b \pmod{n})$;
   (b) P_0 and P_1 invoke 1-out-of-2 OT functionality $\mathcal{F}_{\text{OT}}$ where P_1 sends (m_i^0, m_i^1) and P_0 sends a_i. In the end, P_0 receives $m_i^{a_i}$.
(3) P_0 sets $[\![c]\!]_0 \leftarrow \sum_{i=0}^{\ell-1} m_i^{a_i} \pmod{n}$, and P_1 sets $[\![c]\!]_1 \leftarrow \sum_{i=0}^{\ell-1} -r_i \pmod{n}$.

Figure 15: BMT from OT [13]

done locally, Sim can simulate trivially (since no view involves in these operations).

Simulator for corrupted P_1. The idea of simulating view for a corrupted P_1 is almost the same as the simulation for P_0 because the PDTE protocol is essentially symmetric for the two parties. Therefore, we omit the proof for the corrupted P_1.

B CORRELATED RANDOMNESS GENERATION

B.1 BMT Generation

For a normal BMT $([\![a]\!], [\![b]\!], [\![c]\!])$, a, b and c are all secret-shared among the parties. In our setting, we want to generate BMT $(a, b, [\![c]\!])$ where P_0 holds $(a, [\![c]\!]_0)$ and P_1 holds $(b, [\![c]\!]_1)$. In the following, we summarize two ways for generating such special BMTs using either AHE or OT.

AHE-based approach [13]. BMTs can be generated from AHE, *e.g.*, Paillier encryption. The parties can explore the additive homomorphic property to share the multiplication result over $\mathbb{Z}_n$. In Fig. 14, we give a protocol for generating BMTs from Paillier encryption.

OT-based approach [13] A BMT $(a, b, [\![c]\!])$ can be generated by using OTs as shown in Fig. 15.

B.2 WBV Generation from FSS

Boyle *et al.* [6, 7, 15] formalize a new cryptographic primitive called *Function Secret Sharing* (FSS), and gave concrete constructions for useful functions. We begin with formally defining *Function Secret Sharing* for two parties.

Definition 5 (Function Secret Sharing). A two-party FSS scheme $\Pi_{\text{fss}} = (\text{Gen}, \text{Eval})$ consists of a pair PPT algorithms as follows:

- $\text{Gen}(1^\kappa, f)$ is a key generation algorithm, which takes as input a security parameter 1^κ and a function description f, outputs a tuple of keys $(k_0^{\text{fss}}, k_1^{\text{fss}})$, each for one party.
- $\text{Eval}(k_i^{\text{fss}}, x)$ is an evaluation algorithm, which on input a key k_i^{fss} for party P_i $(i \in \{0, 1\})$, and an evaluation point $x \in \{0, 1\}^\ell$, outputs a group element $y_i \in \mathbb{G}$ as the share of $f(x)$ for P_i.

Definition 6 (Security of FSS). A secure two-party FSS satisfies the following requirements:

- Correctness: for all function $f : \{0, 1\}^\ell \to \mathbb{G}$ and every $x \in \{0, 1\}^n$, if $(k_0^{\text{fss}}, k_1^{\text{fss}}) \leftarrow \text{Gen}(1^\kappa, f)$ then $\Pr[\text{Eval}(k_0^{\text{fss}}, x) + \text{Eval}(k_1^{\text{fss}}, x) = f(x)] = 1$.
- Secrecy: For every corrupted P_i and every sequence of function $f_1, f_2, \ldots$, there exists a PPT simulator Sim such that for $i \in \{0, 1\}$:

$$\{k_i^{\text{fss}} : (k_0^{\text{fss}}, k_1^{\text{fss}}) \leftarrow \text{Gen}(1^\kappa, f_\kappa)\}_{\kappa \in \mathbb{N}} \overset{c}{\equiv} \{\text{Sim}_i(1^\kappa, i, \mathbb{G})\}_{\kappa \in \mathbb{N}}$$

Definition 7 (Point Function). A point function is a function $f_{\alpha,\beta}(x) : \{0, 1\}^\ell \to \mathbb{G}$ where $\mathbb{G}$ is an abelian group such that

$$f_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{if } x = \alpha \\ 0, & \text{otherwize} \end{cases} \tag{1}$$

Our WBV is based on a point function defined in Definition 7, which can be shared using FSS with the key size of $O(\kappa \log m)$. A trusted dealer can generate the keys for a point function with $O(\log m)$ PRG evaluations. In [14], they use two-party computation to generate FSS keys, removing the trusted-dealer assumption. Indeed, the technique in [14] can be directly used for pre-processing WBV. Specifically, each party P_i just locally sample a random share $\langle \text{rdx} \rangle_i \in \mathbb{Z}_2^\ell$, and then run the two-party FSS key generation protocol of [14] to compute the keys. Each party will hold a FSS key after secure computation, and then each party can evaluate his key over $i \in [0, m)$ to generate his own share of a WBV. By definition of point function, the parties will only share 1 at rdx, and 0 for any $i \neq \text{rdx}$.