Let's Stride Blindfolded in a Forest: Sublinear Multi-Client Decision Trees Evaluation

Jack P. K. Ma
and Raymond K. H. Tai
Department of Information Engineering
The Chinese University of Hong Kong
{mpk016,tkh016}@ie.cuhk.edu.hk

Yongjun Zhao
Strategic Centre for Research in
Privacy-Preserving Technologies & Systems
Nanyang Technological University, Singapore
yongjun.zhao@ntu.edu.sg

Sherman S. M. Chow
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
sherman@ie.cuhk.edu.hk

***Abstract*—Decision trees are popular machine-learning classification models due to their simplicity and effectiveness. Tai *et al.* (ESORICS '17) propose a privacy-preserving decision-tree evaluation protocol purely based on additive homomorphic encryption, without introducing dummy nodes for hiding the tree structure, but it runs a secure comparison for each decision node, resulting in linear complexity. Later protocols (DBSEC '18, PETS '19) achieve sublinear (client-side) complexity, yet the server-side path evaluation requires oblivious transfer among 2^d real and dummy nodes even for a sparse tree of depth d to hide the tree structure.**

This paper aims for the best of both worlds and hence the most lightweight protocol to date. Our complete-tree protocol can be easily extended to the sparse-tree setting and the reusable outsourcing setting: a model owner (resp. client) can outsource the decision tree (resp. attributes) to two non-colluding servers for classifications. The outsourced extension supports multi-client joint evaluation, which is the first of its kind without using multi-key fully-homomorphic encryption (TDSC '19). We also extend our protocol for achieving privacy against malicious adversaries.

Our experiments compare in various network settings our offline and online communication costs and the online computation time with the prior sublinear protocol of Tueno *et al.* (PETS '19) and $O(1)$-round linear protocols of Kiss *et al.* (PETS '19), which can be seen as garbled circuit variants of Tai *et al.*'s. Our protocols are shown to be desirable for IoT-like scenarios with weak clients and big-data scenarios with high-dimensional feature vectors.

I. Introduction

Machine learning summarizes training data as a mathematical model that can make predictions on (unforeseen) queries. Clients can get prediction results, but the simple means of uploading their information to a prediction service provider forfeits their privacy. Meanwhile, the model is a valuable asset of the service provider, and it may leak information about the sensitive training data. Revealing the model for client-side prediction is not sensible and may even violate the law.

Privacy-preserving inference protects the privacy of both the client and the model owner. While generic cryptographic tools exist (*e.g.*, [28], [34], [2]), classifier-specific solutions can be more efficient. We focus on the decision-tree classifier [9], which is widely used [24], *e.g.*, for spam detection, multimedia protocol tunneling, and content classification. Tree evaluation starts from the root node as the first *decision node* of the tree. It compares a node-specific threshold with one of the *features* or *attributes* in the client query, and traverses to the left or right node based on the comparison. The process iterates across each level until reaching a *leaf node*, which stores a classification label as the tree evaluation result. The simple comparison-based design makes decision trees interpretable; on the other hand, it may also deter model owners from providing inference as a service if no privacy protection over the model is in place.

Corresponding authors: Yongjun Zhao and Sherman Chow. Part of the work was done while Yongjun Zhao was with the Chinese University of Hong Kong.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.23166
www.ndss-symposium.org

Note that the comparisons along the evaluation path suffice to derive the result, *i.e.*, the number of comparisons is linear in the *tree depth* d, or *sublinear* in the *tree size* denoted by m (*cf.*, the number of decision nodes). Ideally, the complexity of a privacy-preserving decision-tree evaluation protocol should also be sublinear, without being blown up by any artifacts introduced for privacy, *e.g.*, padding a tree until it has 2^d nodes.

We also consider outsourcing computation. For scenarios such as Internet-of-things (IoT) and smart metering, weak devices often periodically report data to a cloud platform, which can run decision-tree classification over the data (Fig. 1). We observe that outsourcing not only helps in offloading but also enriches the functionality, such as *multi-client* evaluation, in which each client contributes a part of the feature vector (*e.g.*, multiple hospitals hold different records of a patient). Trivial approaches leak the interested feature of each decision node (*e.g.*, a specific hospital is contacted directly for the attribute of interest it holds). With secure outsourcing, a single cloud platform can run classification over a composite feature vector collected from different clients, without learning the attributes and without further interacting with any clients. Similarly, secure outsourcing also supports evaluations of *random forests* contributed by *multiple decision-tree owners*.

This work aims to make privacy-preserving inference more usable in practice. Technically, we propose a baseline protocol that achieves the best of two existing paradigms, namely, sublinear client-side computation complexity, without incurring an exponential blow-up in server-side complexity. Not only this improves the state-of-the-art on the foundational level, but this design is also easily extensible to the outsourced setting, saving us the ad-hoc efforts in customizing yet another protocol. We also extend it to achieve privacy against malicious adversaries.

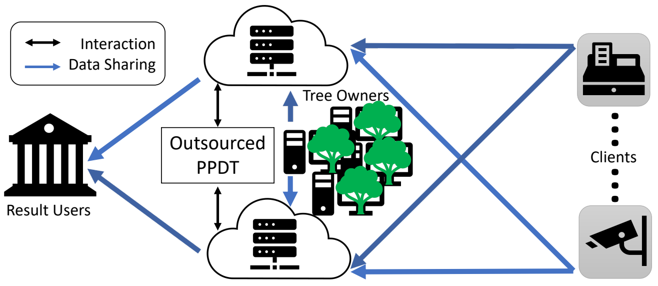


Fig. 1: An outsourced scenario: Cameras and cashiers (the clients) secret-share the photographs and purchase records to the cloud servers, which can then carry out private classification. Multiple tree owners can also delegate their respective models. All (queries, models, and results) are in secret shares.

A. Related Works

Foundational protocols. We first discuss five works with foundational contributions [11] and a systematization study [24]. Bost *et al.* [8] treat a decision tree as a high-degree polynomial and use fully-homomorphic encryption (FHE) to evaluate it. Wu *et al.* [46] get rid of FHE by adding dummy nodes to form a complete tree for hiding the tree structure. In many cases, *the decision tree is sparse* and not complete (*e.g.*, [40]). Padding a deep-but-sparse tree into a complete tree results in a huge ($O(2^d)$) waste in bandwidth and computation. The protocols of Tai *et al.* [40] do not require such tree transformation and are more efficient for sparse trees. Their novelty lies in the re-interpretation of path evaluation as "path cost" computation that is doable via *additive HE* (AHE).

Kiss *et al.* [24] abstract many constructions in the literature and propose a modular construction from three sub-protocols: (oblivious) *feature selection*, (secure) *comparison*, and *path evaluation*, which can be instantiated by either HE or *garbled circuit* (GC) with *secret sharing* (SS). Their detailed comparison suggests that using GC for comparison and HE with the path-cost trick [40] for path evaluation (*i.e.*, GGH or HGH) is the best-fit for *a weak client with limited storage.* Their study covered only protocols prior to their work of at least $O(m)$ complexity.[1] The boundary of the above three tasks may become unclear for sublinear protocols [22], [42].

Joye and Salehi [22] reduce the number of *secure comparisons*[2] to $(d-1)$, *i.e.*, one comparison, each taking 2 rounds, at each tree level. Since each decision node works on a different feature, for feature selection at each level ℓ, the client uses $\binom{2^\ell}{1}$ or 1-out-of-2^ℓ oblivious transfer (OT) protocol, which takes 2 rounds too. The client then decrypts the retrieved ciphertext, and runs secure comparison with the server to get a (shared) bit indicating which node to traverse next. The total number of rounds is linear in d. Unlike [40], this protocol requires the server to transform the tree into a complete tree.

Concurrent to [22], Tueno *et al.* [42] propose a *sublinear* (in m) decision-tree protocol (*i.e.*, only $(d-1)$ comparisons as [22]) using oblivious array indexing (OAI), which can be instantiated by GC+OT, OT alone, or oblivious RAM (ORAM). The tasks involved, feature selection and tree indexing, have complexity $O(n)$ and $O(2^d)$. OAI from OT requires only symmetric-key operations. OAI from (efficient) ORAM requires GC, with complexity sublinear in the number of nodes, but has higher computation and communication costs for small trees. At each level, the server and the client run OAI over the client's attributes and run GC for comparison[3]. They also invoke another OAI over *the* 2^ℓ *decision nodes*[4] at each level ℓ to obtain shares of the index of the next node, followed by a GC to traverse. These take up to 4 rounds per level and $O(2^d)$ communication for processing a randomized complete tree (or $O(d^4)$ communication but $O(d^2)$ rounds if ORAM is used). Table I compares the major protocols in the literature[5].

Outsourced extensions. Aloufi *et al.* [3] consider a client and multiple model owners outsource collaborative evaluation of a random forest to a cloud server via multi-key somewhat-homomorphic encryption (SHE). The model owners send the decision trees encrypted under (pre-generated) joint SHE keys while the client sends its attributes encrypted under its SHE key to the cloud. The cloud then evaluates over the ciphertexts extended under both client and model owners' keys. As such, everyone needs to interact to decrypt the final result.

Two recent (concurrent) works [31], [49] also consider outsourced extension with two non-colluding semi-honest clouds[6]. Liu *et al.* [31] just consider query outsourcing *without hiding the attribute index from the clouds*. The tree owner needs to stay online as one of the clouds. Their comparison utilizes homomorphism for addition and Beaver triplets [7] for multiplication. Its evaluation cost is *linear* in m (plus the number of dummy nodes), and it still leaks an upper bound of m.

Zheng *et al.* [49] replace the "pure AHE" approach [24] of Tai *et al.* [40] with additive-sharing counterparts (and Beaver triplets) using known secure computation tricks [14]. With the client attribute secret-shared to the clouds, feature selection can be done with the naïve 2-server private information retrieval by matrix-vector multiplication with vectorization as in a prior $O(m)$ decision-tree evaluation protocol [14], incurring $O(m)$ online communication[7]. Comparisons use an $O(t)$-round bitwise protocol [14] followed by a standard modulus conversion, such that they still work with the path-cost computation [40].

While these two works [31], [49] aim for outsourcing, their results are the worst of both worlds according to our criteria. Namely, it spoiled the $O(1)$-round benefit of the underlying protocols [8], [40], but without achieving the sublinear (in m) complexity of existing $O(d)$-round protocols [22], [42]. Both works also confine to "regular" outsourcing benefits, without identifying the possible benefits of multi-client or multi-server support. Table II summarizes the outsourcing protocols.

[1]Kiss *et al.* [24] briefly mentioned (without implementation) $O(d)$-round variants of GGG/HGG and "Joye and Salehi present a similar protocol in [22]."

[2]Joye and Salehi [22] also improved the efficiency of the secure comparison protocol of Damgård, Geisler, and Krøigaard (DGK) [16].

[3]The GC contains $3t$ AND-gates for comparison and multiplexing (for selecting the next node to traverse after comparison).

[4]Tueno *et al.* [42] suggested handling sparse trees by OAI over fewer (than 2^ℓ) nodes; however, this leaks the tree structure, so this paper treats it as 2^ℓ.

[5]Table I omits "non-interactive" (*i.e.*, round-optimal) protocols [33], [41], which require public-key somewhat HE and heavier computation. For example, Lu *et al.* [33] transform the 4-round protocol of Tai *et al.* [40] into 2-round.

[6]This computation model was also used for securely computing/outsourcing other tasks (*e.g.*, private set intersection [13] and linear mean classifiers [44]).

[7]The tree owner needs to compute and share an $m \times n$ selection matrix. A trusted initializer is assumed to prepare large vectorized multiplicative triplets of size $m \times n$. The clouds process them with $O(mn)$ communication.

TABLE I: Summary of Private Decision-Tree Evaluation Protocols: n: the number of t-bit client features, m: the number of decision nodes, $\bar{m}$: the number of depth-padded nodes ($O(m^2)$), d: the tree depth, DGK: Damgård–Geisler–Krøigaard protocol, GC: garbled circuit, HE: homomorphic enc/decryption/operation, OT: oblivious transfer, SKE: secret-key enc/decryption, SS: secret sharing. Sel, Comp, Path: feature selection, comparison, and path evaluation. OT or GC not in $\{\}$ is for the whole tree.

	Sel $\{$at level $\ell\}$	Comp	Path $\{$at level $\ell\}$	Client Computation	Server Computation	Round	Leakage
[46]	HE	2^d DGK	HE, $\binom{2^d}{1}$-OT	$O((n+2^d)t$ HE	$O(2^d)$ DGK + $O(2^d)$ HE	$O(1)$	d
[40] (HHH)	HE	m DGK	HE	$O(m+nt)$ HE + $O(m)$ DGK	$O(mt)$	$O(1)$	m
[6] (GGG)	GC, $nt \times \binom{2}{1}$-OT	$O(\bar{m})$ GC, garbled decision tree		$O(\bar{m}t \log \bar{m})$	$O(\bar{m}t \log \bar{m})$	$O(1)$	$d, \bar{m}$
[24] (GGH)	GC, $nt \times \binom{2}{1}$-OT	m GC	HE	$O(m)$ HE + $O(mt \log m)$ SKE	$O(md)$ HE + $O(mt \log m)$ SKE	$O(1)$	m
[24] (HGH)	HE	m GC	HE	$O(m+n)$ HE + $O(mt)$ SKE	$O(md)$ HE + $O(mt)$ SKE	$O(1)$	m
[22]	HE, $\{\binom{2^\ell}{1}$-OT$\}$	d HE	$\binom{2^d}{1}$-OT	$O(n)$ HE + $O(d)$ DGK	$O(2^d+dt)$	$2d$	d
[42] (OT)	SS, $\{\binom{n}{1}$-OT$\}$	d GC[3]	GC, $\{\binom{2^\ell}{1}$-OT$\}$	$O((n+t)d)$	$O(2^d+dt)$	$4d$	d
Ours (Complete)	SS, $\{\binom{n}{1}$-OT$\}$	d GC, $\{\binom{2}{1}$-OT$\}$		$O((n+t)d)$	$O(2^d+dt)$	$2d-1$	d
Ours (Sparse)	SS, $\{\binom{n}{1}$-OT$\}$	d GC, $\{\binom{2}{1}$-OT$\}$		$O((n+t)d)$	$O(m+dt)$	$2d-1$	d, m

TABLE II: Comparison of Two-Server Outsourced Protocols (MT stands for the number of Beaver's multiplication triplets.)

	Selection	Comparison	Path Evaluation	Round	Leakage	Trusted Initializer
Liu *et al.* [31]	no obliviousness	m HE + m MT	dm MT	$d+3$	$d, m, \{\mathsf{ix}_i\}_{i=1}^m$	Required
Zheng *et al.* [49]	mn MT	mt MT	SS	$O(t+d)$	d, m	Required
Ours (Outsourced)	$d \times \binom{n}{1}$-OT	d GC	$d \times \binom{2}{1}$-OT	$2d$	d, m	No

B. Technical Challenges and Our Contribution

We propose semi-honest secure protocols that only perform a sublinear number of secure comparisons. We start with one for complete trees and extend it for sparse trees. The building blocks include additive secret sharing, OT, and GC merely for comparison. Our design is arguably optimal considering feature selection, comparison, and path evaluation.

I) Our *selection* uses 1-out-of-n OT over n client attributes, instead of the garbled decision tree [6] or garbled selection network of GGG/GGH [24], which uses GC and nt OTs [28], incurring an online communication cost of $2\lambda \cdot tn$, where t is the *bit-length of an attribute* (say, 64) and λ is the *security parameter* (say, 128). Looking ahead, online communication for our protocol is dominated by $d \cdot tn$. Usually, decision-tree training limits the depth d if possible, which motivates the pursuit of $O(d)$-round sublinear protocols [22], [42].

Our improvement in communication is more prominent for large n (and t), which suits "big-data" analytics over high-dimensional feature vectors. Particularly, our experiments show a decision tree over MNIST for recognizing handwritten digits, a task that is commonly realized by neural networks nowadays.

II) Linear-time protocols can perform comparisons of *all* nodes. An optimal design performs *only the necessary comparisons*. Note that each decision node compares a different client attribute, denoted by an *index*. If the server learns the index, it can infer the values of the client attributes since it knew where the evaluation is up to. If the client learns the index, partial information about the decision node is revealed. This explains why prior protocols [22], [42] hide the decision nodes among all possible nodes, including dummy ones, at each level during tree evaluation, totaling in $O(2^d)$ complexity. It is unclear how sublinear comparison can be supported without this overhead.

The first novelty of our design is that it uses the "bare minimum" of 1-out-of-2 OT to determine which index to compare at each level. This is achieved by our key management techniques (which, interestingly, are inspired by OT protocols).

III) To hide the tree topology, we introduce an arguably optimal design that uses *at most one* dummy node for each level (to make all paths take $O(d)$ time). The prior best approach (for linear-time GGG [24]) uses depth-padding, which separately pads each leaf with dummies until reaching the tree depth d. This increases the number of tree nodes to $\bar{m} = O(m^2)$.

Reusing dummy nodes for multiple evaluation paths might further complicate the *path evaluation*. The sparse-tree protocol of Tueno *et al.* pushes the evaluation complexity to the GC, *e.g.*, extra AND-gates for multiplexing[3], leaf-node checking, or extra resharing steps. Notably, the GC we need just uses a single comparison gate built from t AND-gates [26], which is much simpler (while Table I simply marked both by d GC).

In summary, metaphorically, the server and client help each other run through a forest blindfolded, yet with confidence, *i.e.*, only deciding among two directions at each step.

IV) We also integrated a few techniques for making our protocol *more efficient* and *extensible*. Our secret-sharing-based design avoids expensive public-key operations (*cf.* [22]). Apart from reducing time and communication, it also enables a conceptually-easy extension for *secure outsourcing*.

Our outsourcing extension shifts almost all workloads to two non-colluding clouds. It also allows *multiple clients*, each holds part of the feature vector, to jointly evaluate a decision tree without leaking their inputs to others. This is non-trivial [12] since the clouds need to pinpoint the required attribute from different data sources. Our protocol is also *reusable*. Clients just need to share their features to the clouds once for many different tree evaluations. This is especially convenient when our protocol can also support random forest evaluation from multiple tree owners. To our knowledge, this is the first such protocol without heavyweight FHE/SHE [3].

V) Our experiments show that our protocols are twice as fast as the prior best $O(d)$-round protocol [42]. We also compare our performance with existing $O(1)$-round protocols for different trees over different networks. Even being $O(d)$-

round protocols, ours remain competitive for deep trees or high-dimensional feature vectors even in WAN.

VI) Finally, for privacy against malicious adversaries, we extend our complete-tree protocol as a showcase (Appendix C), by upgrading the sub-protocols to maliciously-secure variants and supplementing appropriate zero-knowledge proofs.

II. Preliminaries

A. Decision-Tree Classifiers

Let the client input be an n-dimensional feature vector $\vec{x} = (x_1, \ldots, x_n) \in (\mathbb{Z}^+)^n$, and $\mathcal{T}$ be a decision tree with m decision nodes. $|a|$ denotes the bit length of the value a. We denote the decision-tree evaluation result by $v = \mathcal{T}(\vec{x})$. We assume every node of the tree has either 0 or 2 children. Each node has a unique *index* i. Node i refers to a node with index i. The *root node*'s index i is 1. For each node i, $\mathsf{kid}_{0,i} = 2i$ denotes the *index of its left child*, $\mathsf{kid}_{1,i} = 2i+1$ denotes the index of its *right* child. Since we traverse the tree based on a bit denoting the comparison result, we assume a bit is implicitly cast to an integer 0 or 1 when operated by $\{+, -, \times\}$. Each non-leaf node is also associated with a *threshold* y_i, and a *client attribute* (to be compared with y_i) indexed by $\mathsf{ix}_i \in [1, n]$. These form the *node content*. Evaluation starts from the root, descends to the left or right branch based on the test of a node (*e.g.*, comparing the threshold y_i and the attribute x_{ix_i}) until arriving at some leaf node storing $\mathcal{T}(\vec{x})$.

Definition 1. *Given the client's input $\boldsymbol{x}$ and the server's decision tree $\mathcal{T}$, a decision-tree evaluation protocol is correct if the client always learns $v = \mathcal{T}(\vec{x})$ after the protocol execution.*

B. Secret Sharing (SS) and Oblivious Transfer (OT)

We use $(2,2)$-additive secret sharing and denote the secret-shared input by (x^C, x^S), where party C or S gets its share x^C or x^S such that $x^C + x^S = x$ over $\mathbb{Z}_N$ for some integer N. Each share itself does not reveal any information about x.

1-out-of-n OT, denoted by $\binom{n}{1}$-OT, is a protocol in which a server/sender inputs n messages $(m_0, \ldots, m_{n-1})$ and a client/receiver inputs an index i. The client learns only m_i and the server learns nothing about i. $\binom{n}{1}$-OT can be done by using $\log n$ $\binom{2}{1}$-OT [35], [25]. OT extension [20], [5] "extends" a small number of base OT's requiring public-key operations to a larger number of OT's by using symmetric-key operations. The base OT can be pre-computed to reduce the online cost. The complexity of $\binom{n}{1}$-OT of an ℓ-bit string with the improved OT extension of Kolesnikov and Kumaresan [25] is $O(\lambda + n\ell)$.

C. Garbled Circuit (GC) and Conditional OT (COT)

Yao's garbled circuit [47] allows an evaluator with input x to learn $f(x, y)$ for a public function f represented as a boolean circuit, while y is the input of the garbler, who learns nothing. A key ingredient in GC is to let the evaluator obtains (via OT) the input wire key. Several optimization techniques such as free XOR [27] and half gates [48] are proposed to reduce the communication and computational complexities.

A conditional OT (COT) considers a client with x and a server with y and two messages (m_0, m_1). The client can only get m_b where b is the output by the condition $Q(x, y)$. We will set Q to be the "less than" predicate: $Q(x, y) : b \leftarrow (x < y)$. The client can obtain $m_b = m_1$ if $x < y$; m_0 otherwise, but neither y, $Q(x, y)$, nor m_{1-b}. The server learns nothing. We consider the following variant of COT (instantiated in Appendix D, which may be of independent interest) on secret shares: the two parties both hold *secret shares* of x *and* y instead of x and y. Formally, suppose the server/sender inputs are $x_\mathsf{s}, y_\mathsf{s}, m_0, m_1$, and the client/receiver inputs are $x_\mathsf{r}, y_\mathsf{r}$. The protocol outputs $\perp$ to the sender and $m_{((x_\mathsf{s}+x_\mathsf{r})<(y_\mathsf{s}+y_\mathsf{r}))}$ to the receiver, where $x_\mathsf{s} + x_\mathsf{r}$ and $y_\mathsf{s} + y_\mathsf{r}$ refers to modular addition that recovers the original values of x and y, respectively.

Our protocols only embed a GC-based secure comparison and a $\binom{2}{1}$-OT in the COT. The $\binom{2}{1}$-OT can be run in parallel with the GC. We use a variant of GC in which the evaluator obtains the key of the output wire (decoding information is not sent to the evaluator) without knowing the plaintext result.

III. Security Model

Our privacy notion for private decision-tree evaluation is similar to previous works [46], [42], [40]. We first consider *semi-honest* adversaries. The adversary is assumed to follow the protocol specification exactly but may try to learn more information by extra local computation[8]. Throughout the paper, we assume the adversary is taking *static* corruption strategy, corrupting one of the two parties before protocol execution, and learn its private input. We present the standard definition of static semi-honest security in Appendix B-A. Like most applied cryptography papers, we assume the honest party's hardware and software are not compromised. There are no other trust assumptions, say, trusted hardware.

We also consider protocols in a stronger security model, in which the adversary may deviate from the protocol specification. See Section IV-D and Appendix C for details. Finally, our outsourced protocol assumes two non-colluding servers.

The $\mathcal{F}$-hybrid model. We describe our protocols in a "hybrid model" where the two parties interact and use some secure two-party protocols as sub-protocols. When constructing a protocol Π that uses a sub-protocol for securely computing some functionality $\mathcal{F}$, we consider that the parties run Π and use "ideal calls" to a trusted party to compute $\mathcal{F}$. Upon receiving the inputs from the parties, the trusted party computes $\mathcal{F}$ and sends all parties the corresponding output. After receiving these outputs from the trusted party, the protocol Π continues. By the sequential composition theorem [10], any protocol that securely implements $\mathcal{F}$ can replace the ideal calls to $\mathcal{F}$.

We consider two ideal functionalities: The ideal functionality $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$ receives the sender input $(m_0, \ldots, m_{n-1})$ and the receiver input i. It outputs $\perp$ to the sender and m_i to the receiver. The ideal functionality $\mathcal{F}_\mathsf{COT}$ receives the sender inputs $x_\mathsf{s}, y_\mathsf{s}, m_0, m_1$ and the receiver inputs $x_\mathsf{r}, y_\mathsf{r}$. It outputs $\perp$ to the sender and $m_{((x_\mathsf{s}+x_\mathsf{r})<(y_\mathsf{s}+y_\mathsf{r}))}$ to the receiver.

Leakage and Public Parameter. Same as previous protocols, our protocol reveals some public parameters: dimension n of the feature vector, an upper bound t on the bit-length to

[8]Model-inversion attacks are outside of our scope since they can be mitigated by various (non-cryptographic) orthogonal methods, *e.g.*, rate-limiting.

TABLE III: Notations for (Privacy-Preserving) Tree Evaluation

$\vec{x}$	client's n attributes $x_1, \ldots, x_n$
$\mathcal{T}(\vec{x})$	decision-tree evaluation on $\vec{x}$
$\mathcal{T}_i$	node i's information with y_i, ix_i, and $\mathsf{kid}_{b,i}$ or v_i (and msk_i)
m	number of decision nodes
$y_i \in \mathbb{Z}_{2^t}$	threshold for node $i \in [1, m]$ to be compared with x_{ix_i}
$\mathsf{ix}_i \in [1, n]$	index of attribute at node i
v_i	label of a leaf node i (a possible classification result $\mathcal{T}(\vec{x})$)
d	depth of the tree $\mathcal{T}$
$b_{j \in [d]}$	decision bit $(x_{\mathsf{ix}_i} < y_i)$ for some node i at level j
$\mathsf{kid}_{b,i} \in \mathbb{Z}_{2^d}$	index of left/right ($b = 0/1$) child of node i
b'_{dp}	permutation bit at level dp, $1 \leq \mathsf{dp} < d$ (see Fig. 4)
$\mathsf{dsk}_{b,\mathsf{dp}}$	λ-bit keying material at level dp for the left/right node
msk_i	λ-bit secret key for node i at level d' (see Fig. 3)
$\mathcal{T}'_i$	shared (and encrypted) $\mathcal{T}_i$
a^C or a^S	client (C)'s or server (S)'s share of a
	(*e.g.*, $y^S_{\mathsf{dp}} \in \mathbb{Z}_{2^{t+1}}$ is server's share (masking) of y_{dp} at level dp)
$[a^C]^S$	S's share of a^C (for our outsourced extension, see Section V-A)

represent an attribute, and an upper bound d on the tree depth. Our sparse-tree protocol also leaks the upper bound on the number of non-dummy internal nodes of the whole tree (see Fig. 6), but not that of each level (*cf.*, [42]).

IV. Semi-Honest Two-Party Protocol

Table III lists our notations. $\vec{x} = (x_1, \ldots, x_n)$ is the client's attribute vector, y_i is the threshold of node i, and $v = \mathcal{T}(\vec{x})$ is the decision-tree evaluation result. We suppose $|x_1| = \cdots = |x_n| = |y| = t$ and $|v| = d$. We hide the attribute index ix in the OT, so it is operated in $\mathbb{Z}_n$ (shifted by 1), where $|\mathsf{ix}| = \lceil \log n \rceil$. Node index $\mathsf{kid}_{b,i}$ should be in $\mathbb{Z}_{2^d}$, so $|\mathsf{kid}| = d$.

$\mathcal{H}$ is a *key-derivation function* (KDF), which outputs a pseudorandom string of the desired length, and can be instantiated using AES. We require $\mathcal{H}(k_0 \oplus k_1)$ to be indistinguishable from random given $\mathcal{H}(k_0), \mathcal{H}(k_1)$ for randomly chosen k_0, k_1, which holds if we model $\mathcal{H}$ as a random oracle in the proof.

A. Complete-Tree Protocol

We first construct a protocol for a complete decision tree. A non-binary or incomplete tree can be transformed into a complete binary tree by increasing the depth and adding dummy nodes. All leaf nodes in the subtree of dummy decision nodes have the same dummy classification label.

We start with some technical highlights of our protocol (with forward pointers to the relevant steps detailed in Section IV-A5). Fig. 2 is a flow diagram of our approach.

1) Same level, same share: For each decision node i, the server $(2, 2)$-secret-shares to the client threshold y_i and the index ix_i of the client attribute to be compared (or "*the index*"). The server-side shares y^S_{dp} and $\mathsf{ix}^S_{\mathsf{dp}}$ are the same for each node across the same level dp. Thus, the server simply knows the (constant) shares "corresponding to" the path without knowing the path taken. However, we need to ensure that the client can *only* get the client-side share for *one node per level* on the evaluation path, to be discussed in Section IV-A3.

At each level, the client rotates each attribute of the feature vector by $\mathsf{ix}^C_{\mathsf{dp}}$, the server then retrieves the $\mathsf{ix}^S_{\mathsf{dp}}$-th attribute via $\binom{n}{1}$-OT (Fig. 5: Step 2b). Intuitively, the server will get the $\mathsf{ix}_{\mathsf{dp}}$-th attribute of the feature vector. For server privacy, we use the trick of reusing shares again, *i.e.*, the client-side share of

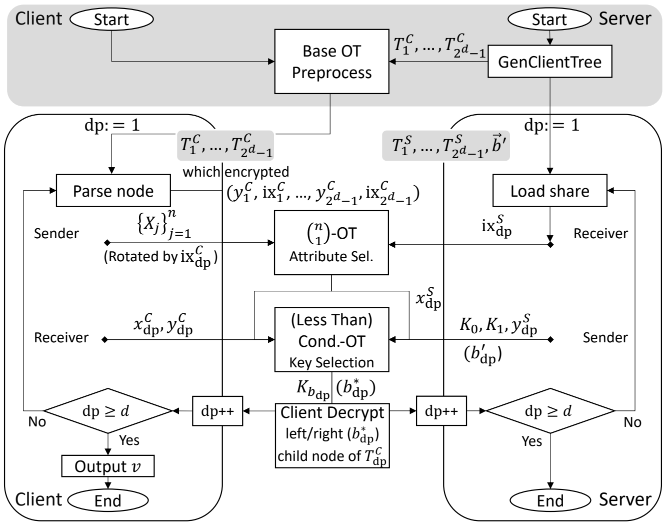


Fig. 2: Overview of Our (Complete-Tree) Protocol (the node content and input/outputs are different for the sparse-tree case)

all attributes are the same for each level of traversal. The client simply knows the client-side share of the attribute transferred, without knowing which index the server is interested in.

2) Compare and Evaluate Together: From both shares of the attribute and the threshold, both parties can then compare them via COT (Fig. 5: Step 2c), which transfers a decision bit (possibly flipped, see Section IV-A5) and one of the two keying materials of the current level to the client. The client can then locally derive the key to recover (the share of) the next node. This merged comparison with path evaluation, while they are separate sub-protocols in the framework of Kiss *et al.* [24].

3) Tree-Node Encryption and its Key Management: For each node i, the server uses a one-time pad (OTP) to encrypt the client share of $\mathcal{T}_i$, which is denoted by $\mathcal{T}'_i$, via $\mathcal{T}'_i \oplus \mathcal{H}(\mathsf{msk}_i)$, where $\mathcal{H}$ is a KDF of $|\mathcal{T}'_i|$ bits. We call msk_i the *node key*. This encrypted tree (share) can be sent to the client in advance. Again, this deviates from Kiss *et al.* [24].

We define $\mathsf{msk}_1 \leftarrow 0^\lambda$, *i.e.*, all zeros for the root ($i = 1$), and assume $\mathcal{T}'_1$ is not encrypted by $\mathcal{H}(0^\lambda)$. At each level dp, the server picks two random bitstrings $\mathsf{dsk}_{0,\mathsf{dp}}$ and $\mathsf{dsk}_{1,\mathsf{dp}}$ as the *keying materials*. Collectively, they suffice for deriving a unique key for each node. Specifically, we define $\mathsf{msk}_{\mathsf{kid}_{b,i}} \leftarrow \mathsf{msk}_i \oplus \mathsf{dsk}_{b,\mathsf{dp}}$ for all node i at level dp, $b \in \{0, 1\}$, *i.e.*, $\mathsf{dsk}_{b,\mathsf{dp}}$ is re-used across half of all nodes at level dp. If node i at level d' is reachable by the evaluation path determined by decision bits $b_1, \ldots, b_{d'-1}$, it is encrypted by $\mathcal{H}(\bigoplus^{d'-1}_{\mathsf{dp}=1} \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}})$.

Fig. 3 depicts the keying materials. In each level, the client can only obtain $\mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$ through conditional OT where b_{dp} is the decision bit a level dp. Throughout the protocol, the client obtains $\mathsf{dsk}_{b_1,1}, \ldots, \mathsf{dsk}_{b_{d-1},d-1}$. Since the keying material for each node key is uniquely determined by its path from the root, all the one-time pads, *i.e.*, the outputs of $\mathcal{H}(\cdot)$, are all different. Thus the client can only decrypt the nodes along the path determined by $b_1, \ldots, b_{d-1}$, where d is the tree depth.

4) Summarizing Our Tree Traversal/Evaluation: Each decision node i consists of the node-specific threshold y_i and

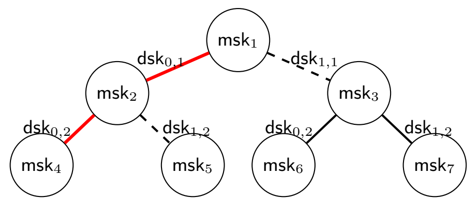
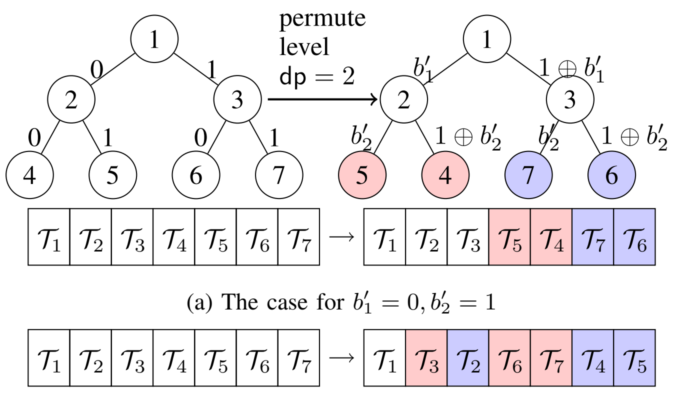


Fig. 3: Key Management (for the leftmost path): We set $\mathsf{msk}_1 \leftarrow 0^\lambda$, $\mathsf{msk}_2 \leftarrow \mathsf{msk}_1 \oplus \mathsf{dsk}_{0,1}$, and $\mathsf{msk}_4 \leftarrow \mathsf{msk}_2 \oplus \mathsf{dsk}_{0,2}$. The client can only obtain the keying material for a branch to advance to the next level along the same branch, *e.g.*, without $\mathsf{dsk}_{1,1}$, node 6 is inaccessible even with $\mathsf{dsk}_{0,2}$.



permute level $\mathsf{dp} = 2$
1
0
1
2
3
0
1
0
1
4
5
6
7
b'_1
$1 \oplus b'_1$
b'_2
$1 \oplus b'_2$
b'_2
$1 \oplus b'_2$
5
4
7
6
$\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_3$ $\mathcal{T}_4$ $\mathcal{T}_5$ $\mathcal{T}_6$ $\mathcal{T}_7$ $\to$ $\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_3$ $\mathcal{T}_5$ $\mathcal{T}_4$ $\mathcal{T}_7$ $\mathcal{T}_6$
$\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_3$ $\mathcal{T}_4$ $\mathcal{T}_5$ $\mathcal{T}_6$ $\mathcal{T}_7$ $\to$ $\mathcal{T}_1$ $\mathcal{T}_3$ $\mathcal{T}_2$ $\mathcal{T}_6$ $\mathcal{T}_7$ $\mathcal{T}_4$ $\mathcal{T}_5$

(a) The case for $b'_1 = 0, b'_2 = 1$

(b) $b'_1 = 1, b'_2 = 0$: nodes at level 1 (and their subtrees) are permuted.

Fig. 4: At each level, permutation bit b'_{dp} decides to flip or not.

Algorithm 1: PermuteTree

```
Input  : $\mathcal{T}$, Permutation bits $\vec{b} = (b'_1, \ldots, b'_d)$
Output: Permutation $\pi(\cdot)$
Set $\pi(1) = 1$
for dp $\leftarrow 1$ to $d$ do
    foreach node $i$ at level dp do
        $\pi(\mathsf{kid}_{0,i}) \leftarrow 2\pi(i) + b'_{\mathsf{dp}}$        // update $\pi$ at $\mathsf{kid}_{0,i}$
        $\pi(\mathsf{kid}_{1,i}) \leftarrow 2\pi(i) + 1 - b'_{\mathsf{dp}}$    // update $\pi$ at $\mathsf{kid}_{1,i}$
return $\pi : [1, 2^{d+1} - 1] \to [1, 2^{d+1} - 1]$
```

Algorithm 2: GenClientTree

```
Input  : Server tree $\mathcal{T}$
Output: Encrypted client share of the tree $\mathcal{T}'$
        and all random choices of the server
$(b'_1, \ldots, b'_d) \leftarrow\$ \{0,1\}^d$                  // server random choices
$\pi \leftarrow \mathsf{PermuteTree}(\mathcal{T}, (b'_1, \ldots, b'_d))$       // Algorithm 1
$\mathsf{msk}_1 \leftarrow 0^\lambda$                         // must traverse the root (no privacy)
for dp $= 1$ to $d - 1$ do                      // set $\mathcal{T}'_i$, derive key material
    $y^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_{2^{t+1}}$, $\mathsf{ix}^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_n$          // server share at lv. dp
    $\mathsf{dsk}_{0,\mathsf{dp}}, \mathsf{dsk}_{1,\mathsf{dp}} \leftarrow\$ \{0,1\}^\lambda$               // keying material
    foreach node $i$ at level dp do     // same lv. same share
        $\mathcal{T}'_i \leftarrow (y_i - y^S_{\mathsf{dp}}) || (\mathsf{ix}_i - \mathsf{ix}^S_{\mathsf{dp}})$       // client share
        $\mathsf{msk}_{\mathsf{kid}_{0,i}} \leftarrow \mathsf{msk}_i \oplus \mathsf{dsk}_{0,\mathsf{dp}}$            // left child key
        $\mathsf{msk}_{\mathsf{kid}_{1,i}} \leftarrow \mathsf{msk}_i \oplus \mathsf{dsk}_{1,\mathsf{dp}}$            // right child key
foreach node $i$ at level $d$ do $\mathcal{T}'_i \leftarrow v_i$                   // label
foreach node $i \neq 1$ do $\mathcal{T}'_i \leftarrow \mathcal{T}'_i \oplus \mathcal{H}(\mathsf{msk}_i)$           // OTP
Send $(\mathcal{T}'_1, \ldots, \mathcal{T}'_{2^d - 1})$ after applying $\pi$
```

the attribute index ix_i. After receiving the encrypted nodes $\mathcal{T}'_1, \ldots, \mathcal{T}'_{2^d-1}$ from the server, the COT compares x_{ix_1} and y_1 and let the client obtain the first decision bit b_1 and $\mathsf{dsk}_{b_1,1}$. The client can then "traverse" by decrypting the next node. This process continues until reaching a leaf node. In this way, the server learns neither the attribute nor the attribute index used in the COT (which leaks information about the evaluated decision nodes), but the client learns the evaluation path from the decision bits. Below we discuss how to protect the latter.

5) Detailed Construction: Before the online inference protocol, the server runs GenClientTree (Algorithm 2) to generate the encrypted decision tree share (and sends it to the client).

Randomizing the Decision Tree. To avoid leaking the evaluation path, the server picks random bits $\{b'_{\mathsf{dp}}\}$ for PermuteTree (Algorithm 1), which permutes all subtrees at level dp where $b'_{\mathsf{dp}} = 1$. The bits $(b'_1, \ldots, b'_d)$ solely held by the server determine the permuted index (Fig. 4) and whether the comparison decision bit output by the conditional OT needs to be flipped.

Sharing the Decision Tree. The server picks random secret shares[9] y^S_{dp}, $\mathsf{ix}^S_{\mathsf{dp}}$, and computes $y_i - y^S_{\mathsf{dp}}$ and $\mathsf{ix}_i - \mathsf{ix}^S_{\mathsf{dp}}$ for each decision node i at each level dp (except the leaves). The server thus *knows the inputs* for the COT without knowing which node is traversed. For the client share, the server encrypts it as in Algorithm 2 (overviewed in Section IV-A3) such that the client can only decrypt one node and hence one secret-shared content per level, after obtaining the keys from the COT.

Selecting the Attribute at Decision Nodes. In the online phase in Fig. 5, the client starts from $\mathsf{dp} = 1$, obtains the shares of threshold and attribute index y^C_{dp}, $\mathsf{ix}^C_{\mathsf{dp}}$ (after decryption if it is not the root), and secret-shares the attribute at $\mathsf{ix}_{\mathsf{dp}}$ to the server through OT in Step 2b. Here, the client's shares for all attributes are identical (x^C_{dp}), so it does not need to know which share is obtained by the server. Namely, the client computes $X_j \leftarrow x_{j+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}}$ for $1 \leq j \leq n$ in Step 2a, where x_j is the j-th client attribute. The inputs to the OT are then all rotated by $\mathsf{ix}^C_{\mathsf{dp}}$. The server inputs $\mathsf{ix}^S_{\mathsf{dp}}$ to get $x^S_{\mathsf{dp}} \leftarrow X_{\mathsf{ix}^S_{\mathsf{dp}}}$. Note that $X_{\mathsf{ix}^S_{\mathsf{dp}}} = x_{\mathsf{ix}^S_{\mathsf{dp}} + \mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}} = x_{\mathsf{ix}_{\mathsf{dp}}} - x^C_{\mathsf{dp}}$, thus $x^C_{\mathsf{dp}} + x^S_{\mathsf{dp}} = x_{\mathsf{ix}_{\mathsf{dp}}}$.

Traversing the Decision Tree. Both parties holding the secret shares of the threshold and the attribute now engage in the COT in Step 2c for $K_e \leftarrow b^* || \mathsf{dsk}_{e,\mathsf{dp}}$ where $b^* = e \oplus b'_{\mathsf{dp}}$, which is $((x^C_{\mathsf{dp}} + x^S_{\mathsf{dp}}) < (y^C_{\mathsf{dp}} + y^S_{\mathsf{dp}}))$ but possibly flipped by b'_{dp}. The client then traverses to the node $\mathsf{kid}_{\mathsf{dp}+1} \leftarrow 2\mathsf{kid}_{\mathsf{dp}} + b^*$, and decrypts its secret-shared content using the key $\mathsf{msk}_{\mathsf{dp}+1} \leftarrow \mathsf{msk}_{\mathsf{dp}} \oplus \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$ (Fig. 5: Step 2d). The process is iterated until a leaf node is reached, and the client outputs the result.

Theorem 1 (Correctness)**.** *Assuming the correctness of the underlying OT and conditional OT, if both the client and server follow our two-party complete-tree protocol (Fig. 5), the client*

[9]The operation over the secret shares of x, y is taken modulo by 2^{t+1}, due to our conditional OT instantiation (Lemma 1). For ix, the operation is carried out over $[1, n]$, similar to modulus operation.

Complete-Tree Online Protocol:

1) The client sets $\mathsf{msk}_1 \leftarrow 0^\lambda$ and $\mathsf{kid}_1 \leftarrow 1$.
2) **for** $\mathsf{dp} = 1$ **to** $d - 1$ **do**
   a) The client parses $y^C_{\mathsf{dp}} || \mathsf{ix}^C_{\mathsf{dp}} \leftarrow \mathcal{T}'_{\mathsf{dp}}$.
      The client picks $x^C_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_{2^{t+1}}$,
      - **for** $j = 1$ **to** n **do** sets $X_j \leftarrow x_{j+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}}$.
   b) The client and the server invoke $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$:
      - The client inputs $(X_1, \ldots, X_n)$.
      - The server inputs $\mathsf{ix}^S_{\mathsf{dp}}$ and gets $X_{\mathsf{ix}^S_{\mathsf{dp}}} = x^S_{\mathsf{dp}}$.
   c) Both parties invoke $\mathcal{F}_{\mathsf{COT}}$:
      - The server inputs $(x^S_{\mathsf{dp}}, y^S_{\mathsf{dp}}, K_0, K_1)$, where $K_0 \leftarrow b'_{\mathsf{dp}} || \mathsf{dsk}_{0,\mathsf{dp}}$, $K_1 \leftarrow (1 \oplus b'_{\mathsf{dp}}) || \mathsf{dsk}_{1,\mathsf{dp}}$.
      - The client inputs $(x^C_{\mathsf{dp}}, y^C_{\mathsf{dp}})$.
   d) The client parses $K_{b_{\mathsf{dp}}}$ as $b^* || \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$ and sets
      - $\mathsf{kid}_{\mathsf{dp}+1} \leftarrow 2\mathsf{kid}_{\mathsf{dp}} + b^*$,
      - $\mathsf{msk}_{\mathsf{dp}+1} \leftarrow \mathsf{msk}_{\mathsf{dp}} \oplus \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$,

      and decrypts $\mathcal{T}'_{\mathsf{dp}+1} \leftarrow \mathcal{T}'_{\mathsf{kid}_{\mathsf{dp}+1}} \oplus \mathcal{H}(\mathsf{msk}_{\mathsf{dp}+1})$.
3) At last, the client outputs the classification $v \leftarrow \mathcal{T}'_d$.

Fig. 5: Online Phase for Complete-Tree Evaluation

learns the classification results $\mathcal{T}(x)$ at the end.

Theorem 2 (Semi-honest Security)**.** *In the random oracle model, the protocol in Fig. 5 securely implements $\mathcal{F}_{\mathsf{DT}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}})$-hybrid model.*

The proofs can be found in Appendices A-A and B-B.

B. Sparse-Tree Protocol

Our sparse-tree protocol is obtained from slightly modifying our complete-tree one (with the same flow in Fig. 2).

1) Hiding the Sparse Structure: A decision tree may not be a complete tree and has a leaf at level $d' < d$. We introduce *one* dummy node at each level from $\mathsf{dp} = 3$ to d (Fig. 6) to hide the tree structure. If a leaf node is reached at some level $d' < d$ during the evaluation, the client continues to traverse the dummy nodes at level $d' + 1$ to d. Note that we "reuse" the dummy nodes instead of padding the tree everywhere. This appears to inherently minimal, or one can observe that a path is shorter than the depth d if the traversal finishes "too early." However, the resulting data structure is not a tree, so we need to modify our complete-tree protocol accordingly.

Our new padding trick seems to require us to "sink" the classification result to the deepest dummy node, but it may be a descendant of more than one real leaf nodes. Instead, we come up with a new way to get the classification result by storing v_i in all nodes, which is 0 for non-leaf and dummy nodes, or the real classification value for leaf nodes. Tree evaluation adds up all v_i's along the path, which leads to the classification result. As other node content, these v_i's are revealed in secret shares. We also set the threshold y_i to 0 for all non-decision nodes, so the dummy traversal after the real leaf always goes left.

To prevent the client from distinguishing the decision node, leaf node, and dummy, each node i should have the same structure. Table IV illustrates the content of each kind of node. (A tree with m decision nodes has at most $m+1$ leaf nodes.)

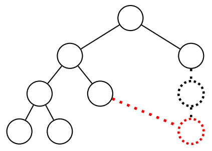


Fig. 6: (Dotted) Dummy Nodes in the Sparse-Tree Protocol

TABLE IV: Node Structure for Our Sparse-Tree Protocol (before secret sharing, msk^C_i is omitted): dp is the level of i

Node i	label	threshold	index	left child	right child
Decision	0	y_i	ix_i	$\mathsf{kid}_{0,i}$	$\mathsf{kid}_{1,i}$
(Real) Leaf	v_i	0	1	$2m + \mathsf{dp}$	0
Dummy	0	0	1	$i + 1$	0

Algorithm 3: $\mathsf{GenClientSparseTree}$

```
Input  : Server Tree $\mathcal{T}$, with $v_i$ if $i$ is a leaf, or $y_i$,
         $\mathsf{ix}_i$, $\{\mathsf{kid}_{b,i}\}_{b=0}^{1}$ for a decision node
Output: Encrypted client share of the tree $\mathcal{T}'$
         and all random choices of the server
1  $(b'_1, \ldots, b'_d) \leftarrow\$ \{0,1\}^d$                    // to be used in Line 18
2  Sample a random permutation subject to $\pi(0) = 0$
     $\pi : \{0, 1, \ldots, 2m+d-1\} \to \{0, 1, \ldots, 2m+d-1\}$
3  for $\mathsf{dp} = 1$ to $d - 1$ do                // set up keys and materials
4      $\mathsf{dsk}_{0,\mathsf{dp}} \leftarrow\$ \{0,1\}^\lambda$, $\mathsf{dsk}_{1,\mathsf{dp}} \leftarrow\$ \{0,1\}^\lambda$
5      foreach node $i$ at level dp do
6          if $\mathsf{msk}_{\mathsf{kid}_{0,i}}$ is not null then    // undefined left child
7              $\mathsf{msk}_{\mathsf{kid}_{0,i}} \leftarrow\$ \{0,1\}^\lambda$              // define node key
8              $\mathsf{msk}_{\mathsf{kid}_{1,i}} \leftarrow \mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus \mathsf{dsk}_{0,\mathsf{dp}} \oplus \mathsf{dsk}_{1,\mathsf{dp}}$
9  for $i = 2m+2$ to $2m+d-1$ do             // connect dummies
10     $\mathcal{T}_i := (v_i, y_i, \mathsf{ix}_i, \mathsf{kid}_{0,i}, \mathsf{kid}_{1,i}) \leftarrow (0, 0, 1, i+1, 0)$
11 for $\mathsf{dp} = 1$ to $d$ do   // prepare nodes of the same structure
12     $y^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_{2^{t+1}}$, $\mathsf{ix}^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_n$    // pick server share of $y$, ix,
13     $v^S_{\mathsf{dp}}, \mathsf{kid}^S_{0,\mathsf{dp}}, \mathsf{kid}^S_{1,\mathsf{dp}} \leftarrow\$ \mathbb{Z}_{2^d}$     // of label, child node IDs
       foreach node $i$ at level dp do
14         if $i$ is a decision node then
15             Prepend $v_i = 0$ to $\mathcal{T}_i$      // no label for non-leaf
16         else if $\mathcal{T}_i$ is a non-dummy leaf node then
17             Append $(0, 1, 2m + \mathsf{dp}, 0)$ to $\mathcal{T}_i$      // after $v_i$
18         Update $\mathcal{T}_i \leftarrow (v_i - v^S_{\mathsf{dp}} || y_i - y^S_{\mathsf{dp}} || \mathsf{ix}_i - \mathsf{ix}^S_{\mathsf{dp}} ||$
             $\pi(\mathsf{kid}_{b'_{\mathsf{dp}},i}) - \mathsf{kid}^S_{b'_{\mathsf{dp}},\mathsf{dp}} ||$
             $\pi(\mathsf{kid}_{1 \oplus b'_{\mathsf{dp}},i}) - \mathsf{kid}^S_{1 \oplus b'_{\mathsf{dp}},\mathsf{dp}} || \mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus \mathsf{dsk}_{0,\mathsf{dp}})$
19 Set $\mathcal{T}'_{\pi(1)} \leftarrow \mathcal{T}_1$          // reveal the root (and encrypt the rest)
20 for $i = 2$ to $2m+d-1$ do $\mathcal{T}'_{\pi(i)} \leftarrow \mathcal{T}_i \oplus \mathcal{H}(\mathsf{msk}_i)$
21 Send $(\pi(1), \mathcal{T}'_1, \ldots, \mathcal{T}'_{2m+d-1}, v^S \leftarrow \sum_{\mathsf{dp}=1}^{d} v^S_{\mathsf{dp}})$
```

Moreover, to hide any inference of the tree structure from the node indices, we apply a random permutation π on them. Correspondingly, each node i needs to include indices of the child nodes $\mathsf{kid}_{0,i}$ and $\mathsf{kid}_{1,i}$. Depending on the permutation bit b'_{dp}, the left and right child entries in Table IV would be switched after applying π. This flips the child nodes explicitly inside the node entry in contrast to the complete-tree case.

2) New Key Management: If we directly employ our key management for complete trees, since the key for a node is determined by all the keying materials on the path from the root to itself, a dummy node will end up with conflicting definitions of its key for being a child of two or more nodes.

Solving this conflict requires a slight change to our key derivation. Our $\mathsf{GenClientSparseTree}$ (Algorithm 3) adopts the following "hybrid" approach that additionally stores in each node i the *"helper" keying material* msk^C_i, which is determined by the node key $\mathsf{msk}_{\mathsf{kid}_{0,i}}$ of its left child. Namely, we first pick $\mathsf{msk}_{\mathsf{kid}_{0,i}}$, or we reuse the existing value if it has been set. Then we set msk^C_i of node i to $\mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus \mathsf{dsk}_{0,\mathsf{dp}}$. The same-level-same-share principle remains here, *i.e.*, the keying materials to be transferred by COT for all left nodes ($\mathsf{dsk}_{0,\mathsf{dp}}$), and respectively all right nodes ($\mathsf{dsk}_{1,\mathsf{dp}}$), are the same for a given level dp. Finally, we set the node key $\mathsf{msk}_{\mathsf{kid}_{1,i}}$ of the right child to $\mathsf{msk}^C_i \oplus \mathsf{dsk}_{1,\mathsf{dp}}$. The node key $\mathsf{msk}_{\mathsf{kid}_{b,i}}$ can be recovered from msk^C_i when COT transferred $\mathsf{dsk}_{b,\mathsf{dp}}$ since the node keys for left or right differ by $\Delta_{\mathsf{dp}} = \mathsf{dsk}_{0,\mathsf{dp}} \oplus \mathsf{dsk}_{1,\mathsf{dp}}$.

Since a right child can never be a dummy, it is never a (left) child of some node other than its single parent. Meanwhile, its left sibling cannot be a dummy since only one dummy node is created as a child of a leaf node having no child nodes originally. Consequently, the top-down assignment for the right child nodes is always possible. Also, the node key of a left child is always generated independently from other nodes. We thus still maintain the property that all node keys of the whole tree are unique. Even though the same helper material may be assigned to an original leaf node and some dummy node at the same level, they are OTP-ed by different node keys, the client cannot figure it out since the client can decrypt at most one node at each level as in our complete-tree protocol.

3) Construction: Algorithm 3 details the setup. Before the online phase, the server first adds a dummy node at each level that contains a decision node at its higher level. To hide the tree structure, a random permutation is applied over the indices. The encrypted content of a node also contains the secret share of permuted child node indices and the helper keying material.

The evaluation of our sparse-tree protocol (Fig. 7) mostly follows the complete-tree protocol, except it also utilizes the appended entries (shares of indices/pointers and helper keying materials). In COT, the client receives b^* and the secret-shared (permuted) index $\mathsf{kid}^S_{\mathsf{dp}}$. Combining $\mathsf{kid}^S_{\mathsf{dp}}$ with the client share $\mathsf{kid}^C_{b^*,\mathsf{dp}}$ in the current node enables the client to obtain the (permuted) index of the next node $\hat{\mathsf{kid}}$. Meanwhile, with $\mathsf{msk}^C_{\mathsf{dp}} = \mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus \mathsf{dsk}_{0,\mathsf{dp}}$ for the current node i and the keying material $\mathsf{dsk}_{b,\mathsf{dp}}$ from COT, the client can obtain $\mathsf{msk}_{\mathsf{kid}_{0,i}}$ or $\mathsf{msk}_{\mathsf{kid}_{1,i}}$ to decrypt the next node (Step 2d).

The sum of v_i in all the traversed nodes equals the sum of $(d-1)$ secret shares of 0 plus the secret-shared classification result in the leaf. With the sum v^S of all server shares of v_i from the offline phase, the client can recover the final result.

Theorem 3. *Assuming the correctness of the underlying OT and conditional OT, if both the client and server follow our two-party sparse-tree protocol (Fig. 7), the client learns the classification results $\mathcal{T}(x)$ at the end.*

Sparse-Tree Online Protocol:
(blue parts are different from the Complete-Tree Protocol)

1) The client sets $\mathcal{T}'_1 \leftarrow \mathcal{T}'_{\pi(1)}$.
2) **for** $\mathsf{dp} = 1$ **to** $d-1$ **do**
   a) The client parses
      $$v^C_{\mathsf{dp}}||y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}}||\mathsf{kid}^C_{0,\mathsf{dp}}||\mathsf{kid}^C_{1,\mathsf{dp}}||\mathsf{msk}^C_{\mathsf{dp}} \leftarrow \mathcal{T}'_{\mathsf{dp}}.$$
      The client picks $x^C_{\mathsf{dp}} \leftarrow_{\$} \mathbb{Z}_{2^{t+1}}$,
      - **for** $j = 1$ **to** n **do** sets $X_j \leftarrow x_{j+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}}$.
   b) The client and the server invoke $\mathcal{F}_{\binom{n}{1}\text{-OT}}$:
      - The client inputs $(X_1, \ldots, X_n)$.
      - The server inputs $\mathsf{ix}^S_{\mathsf{dp}}$ and gets x^S_{dp}.
   c) Both parties invoke $\mathcal{F}_{\mathsf{COT}}$:
      - The server inputs $(x^S_{\mathsf{dp}}, y^S_{\mathsf{dp}}, K_0, K_1)$, where the server sets $K_0 \leftarrow b'_{\mathsf{dp}}||\mathsf{kid}^S_{0,\mathsf{dp}}||\mathsf{dsk}_{0,\mathsf{dp}}$ and $K_1 \leftarrow (1 \oplus b'_{\mathsf{dp}})||\mathsf{kid}^S_{1,\mathsf{dp}}||\mathsf{dsk}_{1,\mathsf{dp}}$.
      - The client inputs $(x^C_{\mathsf{dp}}, y^C_{\mathsf{dp}})$.
   d) The client parses $K_{b_{\mathsf{dp}}}$ as $b^*||\mathsf{kid}^S_{\mathsf{dp}}||\mathsf{msk}^S_{\mathsf{dp}}$ and sets
      - $\hat{\mathsf{kid}} \leftarrow \mathsf{kid}^S_{\mathsf{dp}} + \mathsf{kid}^C_{b^*,\mathsf{dp}}$,
      - $\hat{\mathsf{msk}} \leftarrow \mathsf{msk}^C_{\mathsf{dp}} \oplus \mathsf{msk}^S_{\mathsf{dp}}$,

      and decrypts $\mathcal{T}'_{\mathsf{dp}+1} \leftarrow \mathcal{T}'_{\hat{\mathsf{kid}}} \oplus \mathcal{H}(\hat{\mathsf{msk}})$.
3) The client sets $v^C_d|| \cdots \leftarrow \mathcal{T}'_d$ and $v^C \leftarrow \sum_{\mathsf{dp}=1}^{d} v^C_{\mathsf{dp}}$. The classification result is $v \leftarrow v^S + v^C$.

Fig. 7: Online Phase for Sparse-Tree Evaluation

Theorem 4 (Semi-honest Security). *In the random oracle model, the protocol in Fig. 7 securely implements $\mathcal{F}_{\mathsf{DT}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\binom{n}{1}\text{-OT}})$-hybrid model.*

The proofs can be found in Appendices A-B and B-C.

C. Performance Analysis and Comparison

The complete-tree protocol requires $(2d-1)$ rounds. In our protocols, the communication and computation costs involve the encrypted tree and d instances of $\binom{n}{1}$-OT and COT. In more detail, the server needs to encrypt (with OTP from AES) and send $(2^d - 1)$ nodes to the client, each of length $(t + \log n)$. They can be sent in the offline phase. The client will decrypt d nodes with d calls to $\mathcal{H}(\cdot)$. At each level, they engage in $\binom{n}{1}$-OT with output length t, and COT with output length $(\lambda + 1)$. With (the public-key operations of) base-OT's preprocessed, $\binom{n}{1}$-OT's can be carried out efficiently online [25], [5].

More concretely, the communication cost of each $\binom{n}{1}$-OT with output length t is $(4\lambda + nt)$ bits. The computation cost for the client/server is $n/1$ hash function. A COT requires a GC with t AND-gates, one $\binom{2}{1}$-OT with output length λ, transmissions of two λ-bit hash values, and two $(\lambda + 1)$-bit encrypted messages. The communication cost of GC with t AND-gates and OT with output length λ is $7t\lambda$ bits and 2λ bits, respectively. The computation cost of GC with half-gate optimization is 2/4 hash function calls per AND-gate for the client/server. Thus the cost of COT is $(7t + 8)\lambda + 2$ bits.

In the sparse-tree protocol, the server sends $(2m + d - 1)$

TABLE V: Our Online/Offline Costs for Client/Server Side

	(Approximated) Communication Cost	Computation Cost
Online	Complete: $d(4\lambda + nt) + (7t + 8)\lambda$ Sparse: $d(4\lambda + nt) + (7t + 8)\lambda + 8d$	Client: $O((n + t)d)$ Server: $O(td)$
Offline	Complete: $2^{d+1}(t + \log n)$ Sparse: $(2m + 1)(t + \log n + \lambda + 3d)$	Client: $-$ Server: $O(2^d)/O(m)$

encrypted nodes, each $(\lambda + t + 3d + \log n)$ long, to the client, and the output length of the COT is $(\lambda + d + 1)$. The online and offline costs are listed in Table V with small terms omitted.

D. Upgrading the Sub-protocols

Prior $O(1)$-round linear-time protocols [40], [46] provided one-sided maliciously-secure versions, while the prior sublinear protocols [22], [42] only considered semi-honest security. Interactive evaluations [22], [42] may leak more information. A malicious client could alter the attribute to be supplied or the (secret-shared) tree it received in any round and possibly inferring information on what attribute was being selected.

Fortunately, we could achieve privacy against a malicious client/server by further replacing the GC with its maliciously-secure version [5]. We provide an analysis in Appendix C. Note that the "pure GC" approach [24] of Barni *et al.* [6] has already suggested a generic way to achieve malicious security for privacy by upgrading the OT (extension) to its maliciously-secure version. The efficiency of such a defense mechanism against a malicious attacker within GC will be much lower than those tailor-made approaches over HE or homomorphic commitments in general, *e.g.*, [46], [40]. The general design is routine, which extensively introduces zero-knowledge proofs (ZKPs) and (signed) commitments. The tailored part is for efficiency, *e.g.*, coming up or using "compatible" ZKPs.

To limit the malicious client's capability, the client should commit to its feature vector $\vec{x}$ first and later proves that it has been using the same committed $\vec{x}$ throughout the protocol execution. Furthermore, we use ZKPs and signatures as follows: the tree shares are signed via $\mathsf{Sig}_{\mathsf{sk}}(\cdot)$ by the server. In the online phase, if the decrypted tree shares are not in the right form, or the signature verification fails, the client aborts. To prove that all the tree shares fed into conditional OTs are indeed generated by the server, the client ZKP-proves that it possesses the corresponding server's signatures, *i.e.*, proving the statements described in Fig. 12 in Appendix C. Efficient instantiations of the corresponding ZKPs are known. For example, one can use the technique of verifiable rotation of HE ciphertexts [18] to instantiate the second statement.

The attribute selection and conditional OT can be instantiated using (maliciously-secure) garbled circuit separately with ZKP and cut-and-choose technique [21], [29]. One can also combine the attribute selection (including the feature vector rotation), conditional OT, and signature verification for each level in a single circuit. The garbled circuit takes as input $(\vec{x}, y^C_{\mathsf{dp}}, \mathsf{ix}^C_{\mathsf{dp}}, \mathsf{Sig}_{\mathsf{sk}}(y_{\mathsf{dp}}||\mathsf{ix}_{\mathsf{dp}}||\mathsf{dp}))$ from the client and $(y^S_{\mathsf{dp}}, \mathsf{ix}^S_{\mathsf{dp}}, b'_{\mathsf{dp}}, K_0, K_1, \mathsf{vk})$ from the server, where vk is the public verification key corresponding to the private signing key sk. It outputs the possibly flipped decision bit and K_b to the client as in the complete-tree protocol. The circuit for attribute selection is similar to that used by Tueno *et al.* [42], which consists of XOR-gates for reconstructing the shared index, and n equality gates and n multiplexers for picking the attribute. A comparison gate with multiplexer can then be used to return the permuted bit and the decryption key for the next node.

Looking ahead, in the outsourced setting, the clouds do not collude and only receive information-theoretically secure secret shares, which could also achieve privacy against malicious adversary formalized in [4]. The clouds cannot learn if a classification result changes because the leaf nodes containing the label are freshly shared each time, as long as the client does not collude with one of the clouds. Hence, the clouds cannot infer extra information about the feature vector nor the decision tree from information-theoretically secure secret shares.

V. Outsourced Protocol for Joint Inputs

Our basic protocols require the client and the model owner to interact with each other directly. We consider securely outsourcing evaluation for both parties as described below by using two non-colluding cloud servers. The clouds do not need to interact with the client or the model owner during traversal after obtained the corresponding private inputs (for once). Our outsourced protocol still hides which attribute is being considered at each tree level from either cloud. These features enable our extension to support multiple clients and multiple owners (or contacting a certain client during traversal already reveal something about the attribute index of interests).

The query, as a feature vector, can consist of attributes from different clients (*e.g.*, x_1, x_3 from client 1, and x_2, x_4, x_5 from client 2). We assume the index of each client attributes ($\{1, 3\}$ and $\{2, 4, 5\}$) in the "combined" feature vector ($\{x_i\}_{i=1}^5$), and the unique identifier connecting their vertically partitioned attributes (*e.g.*, when many hospitals hold different parts of patient records) are publicly known. This combination is conceptual. Each client can send the attributes in secret shares to the clouds independently. Similarly, multiple model owners can share their respective decision tree to the clouds independently, possibly forming a random forest acting on the same query.

Our secret-sharing-based design also allows simple arithmetics (additions and constant multiplications) over the shares (*e.g.*, the average of attributes for horizontally partitioned data) by the clouds. Our outsourced protocol does not use heavyweight tools such as multi-key homomorphic encryption.

A. Intuition: Secret-Sharing the Inputs to Two Clouds

In our base protocol (Algorithm 2), the client and the model owner split their respective private inputs into two shares. One may be tempted to use it as-is for outsourcing; namely, they send their respective client shares to the first cloud, and their respective model-owner shares to another cloud. The clouds can then run our base protocol, with one acting solely as the client and another acting solely as the model owner.

Such a trivial approach may work, but *at most for one time* due to linkability issues because the shares (*e.g.*, of the tree) and random choices of each cloud never change. If a cloud saw the same intermediate output again (*e.g.*, after OT) at a certain tree level in the second invocation, it can conclude that the current query shares similarity with the previous one. To ensure unlinkability even when one of the clouds colludes

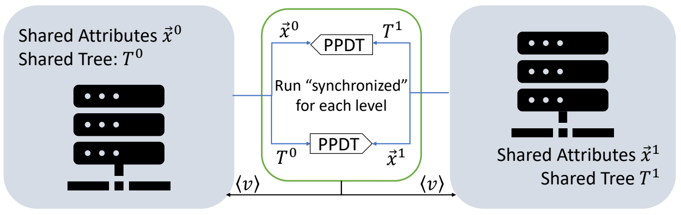


Fig. 8: Two clouds with shares of the tree and clients' attributes run in parallel 2 instances of the 2-party protocol playing as the tree-owner/server and the client, and receive a shared result.

with any client or model owner, both clouds must refresh their secret shares for each query, so they can never re-use any prior shares and see the same intermediate result after their interaction. That is, for each query, each cloud runs $\mathsf{GenClientTree}$ (Algorithm 2) over $(2, 2)$-secret sharing of $\mathcal{T}$ (y_i and ix_i for all nodes i). Each cloud also generates a share of the share of $\vec{x}$ to another cloud. Intuitively, due to the non-colluding assumption, at least one cloud would provide a new share to each invocation of the protocol, which breaks any linkage across different invocations. Also, the client and the model owner do not need to redistribute the shares every time.

For evaluation, the clouds run two instances of the two-party evaluation protocol (in Section IV-A5) synchronously (in level) but with opposite roles. Namely, each cloud executes the evaluation protocol in Fig. 5 as a "client" using the share of $\vec{x}$ with the other cloud as the "owner." For example, when a client shares x to the clouds as $[x]^{\mathcal{CS}_0}$ and $[x]^{\mathcal{CS}_1}$, cloud $\mathcal{CS}_0$ is going to further share it with $\mathcal{CS}_1$ in the form of $[[x]^{\mathcal{CS}_0}]^S$, denoting the owner role of $\mathcal{CS}_1$. We still keep the superscript ^{S} notation while we refer to the model owner. (Strictly speaking, both clouds are now servers of the prediction service). Even though such preprocessing further splits their shares of tree and attributes, they can obtain the required secret-shared result without learning the attributes nor the tree during the evaluation, a feature inherited from the stand-alone setting.

It would be nice if the outsourced extension can just run two independent invocations of our basic protocol, named as PPDT in Fig. 8, in a black-box manner. However, two parts in the evaluation prevent us from doing so in a similar manner.

The first part concerns the attribute index. If the two instances stay independent, each of them always works on a random index, which is never the real index indicating the attribute to be compared. This issue can be resolved if each cloud also takes the share from the other instance of the protocol as an input to make it "complete" (for the OT).

For example, when the first cloud $\mathcal{CS}_0$ plays a client, it has the client share for sure. Meanwhile, this cloud also plays as a server in the other instance and obtains a "server share" from $\mathcal{CS}_1$ who treats $\mathcal{CS}_0$ as a "server." Now, $\mathcal{CS}_0$ forms a single "client share" by adding up both shares. Similarly, $\mathcal{CS}_1$ also obtains one server share in an instance and one client share from the other. $\mathcal{CS}_1$ adds them up as a single "server share."

In short, each cloud forms the share expected by the underlying protocol by adding up the two shares across two instances. The second part concerns the inputs compared by the conditional OT in a similar way. The same treatment applies.

Outsourced Protocol:
($\mathcal{CS}_{1-s}$ and $\mathcal{CS}_s$ parallelly plays as client and server respectively. All the variables are local, except those marked in blue, which come from the other instance.)
The client sets $\mathsf{msk}_1^C \leftarrow 0^\lambda$, $\mathsf{kid}_1^C \leftarrow 1$, $\mathcal{T}_1'^C \leftarrow [\mathcal{T}_0'^{\mathcal{CS}_s}]^C$.
for $\mathsf{dp} = 1$ **to** $d-1$ **do**

1) The client parses $[y_{\mathsf{dp}}^{\mathcal{CS}_s}]^C || [\mathsf{ix}_{\mathsf{dp}}^{\mathcal{CS}_s}]^C \leftarrow \mathcal{T}_{\mathsf{dp}}'^{\,C}$.
2) The client picks $[x_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^C \xleftarrow{\$} \mathbb{Z}_{2^{t+1}}$ and sets $X_j^C \leftarrow x^{\mathcal{CS}_{1-s}}_{j+[\mathsf{ix}_{\mathsf{dp}}^{\mathcal{CS}_s}]^C + [\mathsf{ix}_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^S} - [x_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^C$, $\forall j \in [n]$.
3) The client and the server invoke $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$:
   - The client inputs $(X_1^C, \ldots, X_n^C)$.
   - The server inputs $[\mathsf{ix}_{\mathsf{dp}}^{\mathcal{CS}_s}]^S + [\mathsf{ix}_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^C$.

   In the end, the server outputs $[x_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^S$.
4) The client and server invoke $\mathcal{F}_{\mathsf{COT}}$:
   - The server inputs $([x_{\mathsf{dp}}^{\mathcal{CS}_s}]^C + [x_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^S$, $[y_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^S + [y_{\mathsf{dp}}^{\mathcal{CS}_s}]^S, K_0, K_1)$.
   - The client inputs $([x_{\mathsf{dp}}^{\mathcal{CS}_s}]^S + [x_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^C$, $[y_{\mathsf{dp}}^{\mathcal{CS}_{1-s}}]^C + [y_{\mathsf{dp}}^{\mathcal{CS}_s}]^C)$.

   where $K_0 \leftarrow b'_{\mathsf{dp}} || \mathsf{dsk}_{0,\mathsf{dp}}^{\mathcal{CS}_s}$, $K_1 \leftarrow (1 - b'_{\mathsf{dp}}) || \mathsf{dsk}_{1,\mathsf{dp}}^{\mathcal{CS}_s}$.
5) The client parses $b^* || \mathsf{dsk}_{b^*,\mathsf{dp}}^C \leftarrow K_{b_{\mathsf{dp}}}$ and sets
   - $\mathsf{kid}_{\mathsf{dp}+1}^C \leftarrow 2\mathsf{kid}_{\mathsf{dp}}^C + b^*$,
   - $\mathsf{msk}_{\mathsf{dp}+1}^C \leftarrow \mathsf{msk}_{\mathsf{dp}}^C \oplus \mathsf{dsk}_{b^*,\mathsf{dp}}^C$,

   and decrypts $\mathcal{T}_{\mathsf{dp}+1}^C \leftarrow [\mathcal{T}^{\mathcal{CS}_s}_{\mathsf{kid}_{\mathsf{dp}+1}^C}]^C \oplus \mathcal{H}(\mathsf{msk}_{\mathsf{dp}+1}^C)$.

At $\mathsf{dp} = d$, the client outputs $[v]^C \leftarrow \mathcal{T}_d^C$.

Fig. 9: Evaluation Phase of Our Outsourced Protocol

B. *Outsourced Complete-Tree Protocol*

Delegation. The tree owner secret-shares the tree $\mathcal{T}_i = (y_i || \mathsf{ix}_i)$ or v_i for all node i, while the client secret-shares its attributes $\vec{x} = (x_1, \ldots, x_n)$ to the clouds. The shares are computed by taking the respective modulo, *e.g.*, shares of a decision node $\mathcal{T}_i$ are $[\mathcal{T}_i]^{\mathcal{CS}_0} = [y_i]^{\mathcal{CS}_0} || [\mathsf{ix}_i]^{\mathcal{CS}_0} = (r_i || s_i)$ and $[\mathcal{T}_i]^{\mathcal{CS}_1} = [y_i]^{\mathcal{CS}_1} || [\mathsf{ix}_i]^{\mathcal{CS}_1} = (y_i - r_i || \mathsf{ix}_i - s_i)$ with $r_i \in \mathbb{Z}_{2^t}, s_i \in \mathbb{Z}_n$.

Cloud Preprocessing. Denote the clouds by $\mathcal{CS}_0, \mathcal{CS}_1$. They invoke two slightly modified instances of the two-party protocol and play as the client (C) and server (S), respectively, *i.e.*, $\mathcal{CS}_s$ plays as client in instance s and server in instance $(1-s)$. In the instance that $\mathcal{CS}_s$ plays as the server, $\mathcal{CS}_s$ first runs $\mathsf{GenClientTree}$ and sends to the counterpart (which plays as the client C) the permuted tree share $\{[[\mathcal{T}_i]^{\mathcal{CS}_s}]^C\}_{i=1}^{2m+1}$ encrypted.

Cloud Online Phase. Both clouds play as client and server (*in parallel*) in two instances of the online two-party evaluation protocol (Fig. 5). We highlight in blue the parts obtained from another instance (of $\mathsf{GenClientTree}$ or the evaluation protocol) in Fig. 9. In the end, both send their local share of v_i and result $[v]^C$ they obtain as "client" to the user for reconstruction.

C. *Security Analysis*

Our outsourced protocol guarantees that the semi-honest non-colluding clouds cannot learn extra information about the decision tree, the feature vector, or the classification result. At the end of the protocol, each cloud only outputs a secret share of the classification result. In the multi-client setting, it is up to the participants to decide whether the clouds should send the secret shares of the classification result to a designated client. With minimal client involvement, our outsourced protocol guarantees that only the designated client learns the classification result without learning any other information (*e.g.*, other clients' feature vectors, the decision tree model).

The security analysis of our outsourced protocol is similar to the analysis of our basic two-party protocol. In a bit more detail, the decision tree and client attributes are shared to two clouds in the setup phase. If the clouds do not collude, the distribution of the secret shares looks completely random to them. In our outsourced protocol, as in our basic protocol, the clouds only learn secret shares or permuted index. We can reduce the security of the outsourcing protocol to the sub-protocols ($\mathcal{F}_{\binom{n}{1}\text{-OT}}$, $\mathcal{F}_{\text{COT}}$) similar to our basic protocols.

For multi-client queries, two or more clients can separately contribute the shares of the client attributes. The clouds perform evaluation on freshly generated shares of the decision tree and client attributes. The client does not need to re-share its attribute for another round of evaluation and does not actively participate in the online protocol execution. Thus, we do not need to consider another client as an adversary especially. Even if it shares its "secret" information to one of the clouds, the leakage is confined to its own attributes since the cloud is protected from knowing which nodes are being traversed. The discussion for the security of multi-model evaluation is similar.

D. Performance Analysis

The online phase of the outsourced complete-tree protocol requires $2d$ rounds of interaction. The clouds computation and communication complexity is $O(2^d + (n+t)d)$. They work as both server and client of the base protocol in parallel and hence share similar workloads. The overall computation time is similar to that of Section IV (when both clouds are as powerful). Specifically, if $\lambda = 128$ and $t = 64$, the communication cost of the outsourced complete-tree protocol is around $2^{d-6} + (2^{-6}n + 14)d$ (1252) for $d = 16, n = 20, m = 25d$.

VI. Empirical Evaluation

We empirically compare our protocols with the state-of-the-art with different round-trip time (RTT) and bandwidth (in bit/second) over local-area network (LAN, RTT: 0.1ms, 1Gbps), metropolitan-area network (MAN, RTT: 6ms, 100Mbps), and wide-area network (WAN, RTT: 80ms, 40Mbps) settings. We run tests for all protocols on a desktop equipped with Ryzen 7 3700x running Ubuntu 18.04.4 LTS on VMware Workstation 15 allocated with 4 cores and 16GB of RAM. The times reported are averaged over 10 trials.

We set the security parameter λ to 128. We implement our protocols using emp-toolkit [45], with semi-honest OT extension and optimized GC. It achieves millions of 1-out-of-2 OTs in a second, which we only require $O(d(\log n + t))$ OTs. In the benchmark, an $\binom{n}{1}$-OT is instantiated by $\log n$ $\binom{2}{1}$-OT's. Our COT is based on GC (see Appendix D), which only requires a single "less-than" gate and a 1-out-of-2 OT; both could be run in 1 round simultaneously, yet it cannot be supported via the API of emp-toolkit. The library supports AES-NI acceleration. We use AES as the KDF and SHA-256 as the hash function. We chose elliptic-curve (lifted-)ElGamal with 514-bit ciphertexts to instantiate HE used by the existing protocols, *e.g.*, the one of Joye–Salehi's [22]. For the protocols of Kiss *et al.* [24], we used their code [23]. In particular, HE used by feature selection in HGH [24] is instantiated by either Paillier encryption with ciphertext packing or DGK encryption [17] with either 4096/2048-bit ciphertext.

TABLE VI: Online Time (ms) of Our Complete-Tree (CT) Protocol and Tuneo *et al.* in the MAN (100Mbps/6ms RTT) and WAN Settings (40Mbps/80ms RTT) for 64-bit attributes

Decision	n	d	m	Metropolitan		Wide-Area		Our
Tree				[42]	Our CT	[42]	Our CT	Cloud
Nursery	8	4	4	0.30	**0.14**	2.01	**0.97**	0.86
Cancer	9	8	12	0.58	**0.24**	4.12	**1.98**	1.63
Housing	13	13	92	1.02	**0.49**	6.25	**3.11**	2.75
Spambase	57	17	58	1.32	**0.61**	8.53	**4.12**	3.98
Contagio	13	13	52	0.91	0.487	6.26	3.03	-

TABLE VII: Tree Parameters in Our Experiments

Decision	#(Attributes)	Tree Depth	#(Nodes)	#(Padded Nodes)	
Tree	n	d	m	$\bar{m}$	2^d
WINE	7	5	11	26	32
LINNERUD	3	6	19	47	64
BREAST	12	7	21	66	128
SYNTHETIC	200	8	255	255	256
DIGITS	47	15	168	1161	32768
DIABETES	10	28	393	6432	$>2 \cdot 10^8$
BOSTON	13	30	425	6768	$>10 \cdot 10^8$
MNIST	784	20	4191	23073	1048576

A. Dataset and Decision-Tree Model

For comparing with the protocol of Tueno *et al.* [42], we use their used datasets from the UCI machine learning repository [19], which are nursery, cancer, housing, and spambase as listed in Table VI. We use 1-hot encoding or label encoding to represent categorical variables and scale up floating-point numbers by multiplying a suitable value while preserving accuracy. These information and n, d, t are known to the client.

For $O(1)$-round protocols surveyed by Kiss *et al.* [24], we adapted their trained decision trees using their code [23] and added two trees for higher-dimensional (n) vectors, one shallow (SYNTHETIC) and one deep (MNIST), Roughly, WINE (chemical analysis), LINNERUD (physical exercise performance), and BREAST (cancer) are small trees, and DIGITS, DIABETES, BOSTON (housing value), and MNIST are deep-but-sparse trees. Table VII lists the tree parameters. Our experiments aim to show their effects under different networks.

B. Comparison with $O(d)$-round Protocols

Both our protocols and Joye–Salehi's [22] take $2d$ online rounds. Our online communication cost (Fig. 10b) is comparable ($\sim$1KB) for small trees ($d \leq 7$) and is at least an order less for large trees ($\sim$50KB for deep trees with the sparse-tree protocol), without any exponential blow-up [22]. We did not evaluate the computation time empirically since Joye–Salehi's needs $O(2^d + dt)$ HE operation while ours are symmetric-key.

Compared to the $4d$-round protocol of Tueno *et al.* [42], ours reduce the online communication cost by at least $3\times$ for large trees, while our offline overhead being less than 30% of our online cost (using our sparse-tree or complete-tree protocol depending on the tree structure). This is because our offline tree sharing eliminated OAI for next-level nodes and GC-based tree traversal. The saving in communication is plotted in Fig. 10. Table VI listed the overall online computation time in detail, showing that ours enjoys $\sim$50% saving. Ours are thus clear winners among $O(d)$-round protocols. The last row and the last column will be discussed in Section VI-D.

C. Comparison with Constant-round Protocols

Communication. Fig. 10 details the comparison in communication[10], excluding OT-preprocessing of GGG/GGH and ours. We start describing GGG [6], which can shift the major communication cost to the offline phase, and hence features a low online communication cost *in general*. However, the feature-vector dimension dominates the online communication cost due to its garbled selection network (for its nt 1-out-of-2 OTs). So, it will be overtaken by ours as n increases.

Using HE for path evaluation [40], the total communication cost of GGH/HGH for deep-but-sparse trees is less than GGG, but it loses advantage for dense trees (for sending $O(m)$ ciphertexts online[11]). GGH costs slightly more in online communication but saves much more in offline cost than GGG [6].

Our sparse-tree protocol is a clear winner in offline communication. (Naturally, our complete-tree protocol suffers when the tree is deep but sparse.) More concretely, it is >2 orders of magnitude less than GGG/GGH ($\sim$1.5 for HGH). We also aim for a fast online phase (sublinear client-side computation with only symmetric-key operations). The price is $O(d)$ communication rounds. For instance, ours suffer from >60·RTT with $\sim$200KB online overhead for the BOSTON deep tree.

In general, our online cost lies between GGH and HGH, except for WINE (too small a tree for ours to be competitive). GGG is a clear winner in online communication, except for large n. We can *roughly* treat DIGITS ($n = 47$) in our datasets as a break-even point. A more precise treatment could be done by synthesizing trees for varying n and d. However, we stress that it is just for *online communication* but not the total online time evaluated below. When we also consider the computation, (n, d) alone does not determine who would be the winner.

Our advantages become prominent for trees processing high-dimensional feature vectors (*e.g.*, SYNTHETIC and MNIST), which incur the smallest total (offline and online) communication cost ($5\times$ saving for online). For tree depth beyond 7, ours also enjoy the lowest total communication cost.

Runtime. After the microscopic evaluation, we consider the online time for completing the protocol. Fig. 11 plots the online runtime in three different network settings[12]. For the slowest network, we removed LINNERUD, BREAST, and BOSTON, which share similar parameters as some others, and added SYNTHETIC and MNIST, both for higher-dimensional vectors. For small trees and small n, our protocols are less efficient than GGG/GGH and perform similarly to HGH. For deep sparse trees, the reported runtime for GGG increases significantly and overtakes GGH/HGH. Our sparse-tree protocol becomes on par with GGH and slightly better for DIGITS in the MAN setting.

In the LAN setting, we outrun all the others for almost all tested trees. Our protocols run roughly an order slower when the latency increases from 0.1ms to 6ms and from 6ms to 80ms. Even for WAN, we outrun GGG, GGH, HGH[13], and HGG when the trees operate over high-dimensional feature vectors (SYNTHETIC and MNIST). Notably, we are still the winner for DIABETES despite the low dimension ($n = 10$), the online latency ($d = 28$), and our relatively higher online communication cost. An explanation is that the effect of the quadratic number of padded nodes ($\bar{m}$) of GGG comes into play, making our saving in the online computation time matters much more. In short, ours are competitive for deep trees or high-dimensional feature vectors.

We stress again that a holistic evaluation also considering offline communication will make our advantages stand out.

Recommandations. The overall performance depends on the client storage capacity, computational power, tree parameters, and network environment.[14] In some scenarios where network latency is low (smart home, IoT), our protocol can save total communication cost while providing rapid on-demand decision tree classification (where the client devices are storage-limited and inefficient for HE operations). Our protocols can also efficiently handle trees with high-dimensional feature vectors and support multi-client evaluation in the outsourced setting.

High Dimensionality in Decision Trees. Due to the "curse of dimensionality," it requires an exponentially-large data set to train a good classification model. Feature selection techniques are usually applied to reduce the feature dimension before training the classifier. However, some data sets contain thousands to million features (*e.g.*, spam detection [30]), with hundreds of them remaining after reduction for a reasonable error rate [32][15]. Existing distributed decision-tree training frameworks train from a large volume of distributed high-dimensional data (*e.g.*, [1]) or produces (gradient boosted) decision trees with a high-dimensional and sparse output space (*e.g.*, [37]) with controlled depth (*e.g.*, 10).

A random-forest evaluation runs multiple (*e.g.*, up to 100) decision trees in parallel (where the RTT effect will not be scaled up). Existing constant-round protocols will impose a considerable storage requirement on the clients' devices (when their offline cost is too large). Our protocols strike a balance between offline and online communication costs.

[10]The focus is $O(1)$-round protocols. We included the online cost for $O(d)$-round Joye–Salehi [22], but DIABETES and BOSTON are too large (10^8) for it. We use our complete-tree protocol for SYNTHETIC since it is a complete tree.

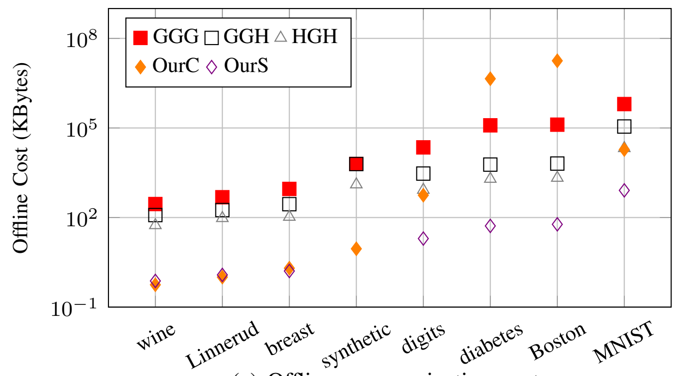
[11]While HE supports single-instruction-multiple-data (SIMD), it only reduces the computation time and ciphertext size when amortized over multiple invocations, but may not fit for prompt decision making over a single query.

[12]Offline times are mostly affected by the library and communication costs.

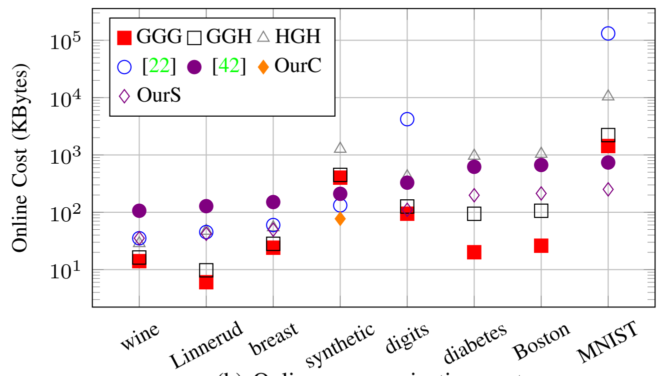
[13]HGH can reduce the offline communication cost ($5\times$), but it uses HE and incurs a larger online communication (>$2\times$) cost than us. Moreover, feature selection with HE requires $O(n + m)$ online HE operations in which HGH performs the worst in shallow trees operating over high-dimensional data.

[14]For instance, GGH performs better than HGH in MAN, and they have similar performance in WAN (except for SYNTHETIC). HGH offers lower total communication costs but has a higher online computation cost.

[15]This work [32] studies perceptron decision tree: each decision node is a linear threshold unit that compares over a weighted sum of multiple features.

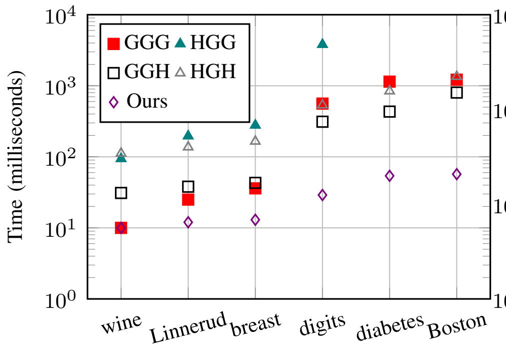


(a) Offline communication cost

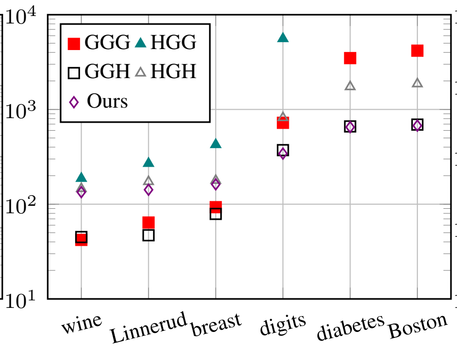


(b) Online communication cost

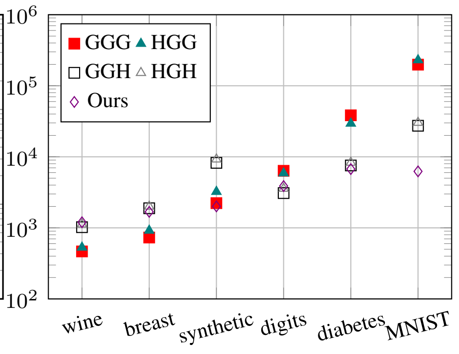
Fig. 10: Offline and Online Communication Cost (in different log scales): OurC/OurS refers to our complete-/sparse-tree protocol.



(a) UCI datasets on LAN (1Gbps/0.1ms)



(b) UCI datasets on MAN (100Mbps/6ms)



(c) Selected tests on WAN (40Mbps/80ms)

Fig. 11: Online Runtime (in different log scales) in LAN/MAN/WAN (bandwidth/RTT) of Kiss *et al.* [24] and Our Sparse-Tree Protocol

D. Further Experiments

Application in Security. While the UCI repository [19] collected many real-world data, they may not cover some of the latest research from specific communities outside data science. As a showcase, we implemented a recent decision-tree classifier for malicious PDF file detection [38], with $5000/5000$ benign/malicious PDF files using the feature extraction tool Mimicus [39] over the Contagio dataset [36]. The training is done with the 13 (out of 136) most important features. The average tree depth and the number of decision nodes are 13 and 52, respectively. For classification, the query is a feature vector extracted from the PDF file, using 64-bit attributes as our other experiments. We achieve 99.4% accuracy with 6-fold cross-validation. Table VI reports the time needed.

Cloud Experiment. To evaluate our outsourcing extension, we use Google Cloud Platform and create two e2.small (2GB RAM) instances located at us-east1 (South Carolina) and europe-west2 (London) with latency 88.5ms and bandwidth 220Mbps. While we assume the clouds have already received shares of the feature vector and tree model from the client and model owner, our experiment includes the preprocessing needed to create a fresh share (*e.g.*, the encrypted shared tree) for each protocol invocation (the green box of Fig. 8).

Table VI shows the online computation time, which is lower than that for the local execution (between the client and model owner) in the WAN setting but not the MAN setting, but it is not a failure because we support *secure* outsourcing without requiring any computation of the model owner or the client except initial sharing of their secret data (and recovering the query result), while prior work [49] may stress on the saving factor of the client computation without accounting for the time needed by the clouds and their network communication. We did not reimplement the existing works [31], [49] because they achieve "the worst of both worlds" as discussed in Section I-A.

VII. Concluding Remarks

Machine learning is becoming more pervasive. Beyond the pursuit of performance, the privacy of both the query and the model is essential. It still takes a long time to process massive neural networks in a privacy-preserving manner. The empirical results confined their predictive power to simpler tasks.

We propose privacy-preserving protocols using lightweight cryptographic tools for decision trees, which have been shown useful for many classification and prediction tasks, ranging from detecting cancer to detecting cyber attacks. Their complexities are sublinear in the number of decision nodes, with a much less online communication cost than previous ones, and take advantage of preprocessing. They can also be easily extended for secure outsourcing to two non-colluding clouds.

For future work, it would be interesting to consider security against malicious adversaries while maintaining the same efficiency level, say, by outsourcing to 3 or more servers.

Acknowledgment

The authors would like to thank the reviewers for the thoughtful comments on enriching performance evaluation and Marina Blanton for her comments on a related MPhil thesis.

This research is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative, and the Research Grant Council, University Grants Committee, Hong Kong under General Research Fund (CUHK 14210319). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore, and Research Grant Council, Hong Kong.

References

[1] F. Abuzaid, J. K. Bradley, F. T. Liang, A. Feng, L. Yang, M. Zaharia, and A. S. Talwalkar, "Yggdrasil: An optimized system for training deep decision trees at scale," in *NIPS*, 2016.

[2] M. Y. Alhassan, D. Günther, Á. Kiss, and T. Schneider, "Efficient and scalable universal circuits," *J. Cryptology*, vol. 33, no. 3, pp. 1216–1271, 2020.

[3] A. Aloufi, P. Hu, H. W. H. Wong, and S. S. M. Chow, "Blindfolded evaluation of random forests with multi-key homomorphic encryption," *IEEE Trans. Dependable Sec. Comput. (TDSC)*, 2019, early Access.

[4] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *CCS*, 2016.

[5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *CCS*, 2013.

[6] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A. Sadeghi, and T. Schneider, "Secure evaluation of private linear branching programs with medical applications," in *ESORICS*, 2009.

[7] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[8] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *NDSS*, 2015.

[9] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *CCS*, 2007.

[10] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[11] S. S. M. Chow, "Privacy-preserving machine learning," in *Frontiers in Cyber Security (FCS)*, 2018, invited paper for keynote talk.

[12] ——, "Can we securely outsource big data analytics with lightweight cryptography?" in *Security in Cloud Computing (SCC), co-located with AsiaCCS*, 2019, invited paper for keynote talk.

[13] S. S. M. Chow, J. Lee, and L. Subramanian, "Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases," in *NDSS*, 2009.

[14] M. D. Cock, R. Dowsley, C. Horst, R. S. Katti, A. C. A. Nascimento, W. Poon, and S. Truex, "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation," *IEEE TDSC*, vol. 16, no. 2, pp. 217–230, 2019.

[15] G. Couteau, "New protocols for secure equality test and comparison," in *ACNS*, 2018.

[16] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *ACISP*, 2007.

[17] ——, "Homomorphic encryption and secure comparison," *Int. J. Appl. Cryptogr.*, vol. 1, no. 1, pp. 22–31, 2008.

[18] S. de Hoogh, B. Schoenmakers, B. Skoric, and J. Villegas, "Verifiable rotation of homomorphic encryptions," in *PKC*, 2009.

[19] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[20] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, 2003.

[21] S. Jarecki and V. Shmatikov, "Efficient two-party secure computation on committed inputs," in *EUROCRYPT*, 2007.

[22] M. Joye and F. Salehi, "Private yet efficient decision tree evaluation," in *DBSec*, 2018.

[23] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, *Private Decision Tree Evaluation (PDTE) Protocols*, 2019, https://github.com/encryptogroup/PDTE, last accessed on Sep. 12, 2020.

[24] ——, "SoK: Modular and efficient private decision tree evaluation," *PoPETs*, no. 2, pp. 187–208, 2019.

[25] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *CRYPTO*, 2013.

[26] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, 2009.

[27] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008.

[28] ——, "A practical universal circuit construction and secure evaluation of private functions," in *FC*, 2008.

[29] B. Kreuter, abhi shelat, and C. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security*, 2012.

[30] C.-J. Lin, *LIBSVM Data: Classification (Binary Class)*, 2019, https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[31] L. Liu, J. Su, R. Chen, J. Chen, G. Sun, and J. Li, "Secure and fast decision tree evaluation on outsourced cloud data," in *ML4CS*, 2019.

[32] W. Liu and I. W. Tsang, "Making decision trees feasible in ultrahigh feature and label dimensions," *J. Mach. Learn. Res.*, vol. 18, pp. 81:1–81:36, 2017.

[33] W. Lu, J. Zhou, and J. Sakuma, "Non-interactive and output expressive private comparison from homomorphic encryption," in *AsiaCCS*, 2018.

[34] P. Mohassel and S. S. Sadeghian, "How to hide circuits in MPC an efficient framework for private function evaluation," in *EUROCRYPT*, 2013.

[35] M. Naor and B. Pinkas, "Computationally secure oblivious transfer," *J. Cryptology*, vol. 18, no. 1, pp. 1–35, 2005.

[36] M. Parkour, *Contagio Malware Dump*, 2013, http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html.

[37] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in *ICML*, 2017.

[38] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *ACSAC*, 2012.

[39] N. Srndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *IEEE S&P*, 2014.

[40] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *ESORICS Part II*, 2017.

[41] A. Tueno, Y. Boev, and F. Kerschbaum, "Non-interactive private decision tree evaluation," in *DBSec*, 2020.

[42] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," *PoPETs*, no. 1, pp. 266–286, 2019.

[43] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *PoPETs*, no. 3, pp. 26–49, 2019.

[44] B. Wang, M. Li, S. S. M. Chow, and H. Li, "A Tale of Two Clouds: Computing on Data Encrypted under Multiple Keys," in *CNS*, 2014.

[45] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[46] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests," *PoPETs*, no. 4, pp. 335–355, 2016.

[47] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *FOCS*, 1986.

[48] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT Part II*, 2015.

[49] Y. Zheng, H. Duan, and C. Wang, "Towards secure and efficient outsourcing of machine learning classification," in *ESORICS Part I*, 2019.

Appendix A
Correctness Analysis

A. The Complete-Tree Protocol (Theorem 1)

Proof: We traverse the tree at each level in the same way, so we just show that a single-step traversal is correct. At level dp, the client has the secret shares of the current node $(y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}})$. The server obtains via $\binom{n}{1}$-OT the share of the to-be-compared attribute x_{dp} from the rotated feature vector. Both parties hold the correct share of attribute x_{dp} and threshold y_{dp}. With a correct conditional OT, the client obtains b^*_{dp} such that $b^*_{\mathsf{dp}} \oplus b'_{\mathsf{dp}} = b_{\mathsf{dp}} = (x_{\mathsf{ix}_{\mathsf{dp}}} < y_{\mathsf{dp}})$, where b'_{dp} is a random bit picked by the server and used to permuted the tree, and the corresponding key $\mathsf{msk}_{b_{\mathsf{dp}},\mathsf{dp}}$. The client sets the index of the next node as $2\mathsf{kid}_{\mathsf{dp}} + b^*_{\mathsf{dp}}$. Since the left and right sub-trees are permuted using b'_v, the client can reach the correct node.

During traversal, the client can compute $\mathsf{msk}_{b_1,1}, \mathsf{msk}_{b_1,1} \oplus \mathsf{msk}_{b_2,2}, \ldots, \bigoplus_{\mathsf{dp}=1}^{d-1} \mathsf{msk}_{b_{\mathsf{dp}},\mathsf{dp}}$, which can decrypt the nodes along the evaluation path determined by $b_1, \ldots, b_{d-1}$. After d iterations, the client thus obtains the correct evaluation. ∎

B. The Sparse-Tree Protocol (Theorem 3)

Proof: The proof is similar to that of Theorem 1. We focus on the differences below. For $\mathsf{dp} = 1$, the client additionally obtains from the COT $\mathsf{kid}^S_1 = \mathsf{kid}^S_{b_1,1}$ and $\mathsf{msk}^S_1 = \mathsf{msk}_{\mathsf{kid}_{b_1,1}}$. In Step 2d, the client sets the (permuted) index of the next node as $\hat{\mathsf{kid}} \leftarrow \mathsf{kid}^S_1 + \mathsf{kid}^C_{b^*_1,1}$. Note that the (permuted) index of the left and right node is set to $\pi(\mathsf{kid}_{0,1})$ and $\pi(\mathsf{kid}_{1,1})$, respectively. When $b_1 = 0$, $\hat{\mathsf{kid}} = \mathsf{kid}'_{0,1} + \pi(\mathsf{kid}_{0,1}) - \mathsf{kid}^S_{0,1} = \pi(\mathsf{kid}_{0,1})$, which is the (permuted) index of the left node. When $b_1 = 1$, we have $\hat{\mathsf{kid}} = \mathsf{kid}'_{1,1} + \pi(\mathsf{kid}_{1,1}) - \mathsf{kid}^S_{1,1} = \pi(\mathsf{kid}_{1,1})$, which is the (permuted) index of the right node. In either case, the client traverses the decision tree correctly.

The node keys for the left and right child nodes of all nodes at level dp differ by $\Delta_{\mathsf{dp}} = \mathsf{dsk}_{0,\mathsf{dp}} \oplus \mathsf{dsk}_{1,\mathsf{dp}}$. Via COT, the client obtains $\mathsf{msk}^S_{\mathsf{dp}} = \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$ and can decrypt the next node using $(\mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus \mathsf{dsk}_{0,\mathsf{dp}}) \oplus \mathsf{msk}^S_{\mathsf{dp}} = \mathsf{msk}_{\mathsf{kid}_{0,i}} \oplus b'_{\mathsf{dp}}\Delta_{\mathsf{dp}}$.

Note that if the leaf node is located at level $d' < d$, the client will continue to traverse through the dummy nodes until reaching a node at level d.

In the last step, the client obtains $v = \sum_{\mathsf{dp}=1}^{d-1}(v^C_{\mathsf{dp}} + v^S_{\mathsf{dp}})$. If the leaf node is at level d', $v^C_{\mathsf{dp}} + v^S_{\mathsf{dp}} = 0$ for $\mathsf{dp} > d'$. Therefore, v is the desired classification result. ∎

Appendix B
Security Analysis

A. Security Definition

We use the simulation-based security definition for two-party computation (2PC). A 2PC protocol Π computes a function $f = (f_1, f_2) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. For every input pair (x, y), the output is $(f_1(x,y), f_2(x,y))$. The first (resp. second) party obtains $f_1(x,y)$ (resp. $f_2(x,y)$). Party i's output is empty if $f_i(x,y) = \perp$.

In the semi-honest model, a protocol Π is secure if whatever can be computed by a party in the protocol can be computed in probabilistic polynomial time (PPT) from its input and output only. This is formalized by requiring the *view* of a party in a protocol execution to be simulatable given only its input and output. We denote the view of the party i during an execution of Π on input (x,y) by $\mathsf{View}^{\Pi}_i(x,y) = (w, r^i, m^i_1, \ldots, m^i_t)$, where $w \in (x,y)$ is the input of i, r^i is i's internal random coin tosses, and m^i_j denotes the j-th message that i received for $1 \le j \le t$.

Definition 2 (Semi-honest Model)**.** *Protocol Π securely computes a deterministic function $f = (f_1, f_2)$ in the presence of static semi-honest adversaries if there exists PPT algorithms $\mathsf{Sim}_1, \mathsf{Sim}_2$ producing computationally-indistinguishable (denoted by $\overset{c}{\equiv}$) views without taking the input of the counterparty:*

$$\{\mathsf{Sim}_1(x, f_1(x,y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}^{\Pi}_1(x,y)\}_{x,y},$$
$$\{\mathsf{Sim}_2(y, f_2(x,y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}^{\Pi}_2(x,y)\}_{x,y}.$$

We treat the problem of privacy-preserving decision tree evaluation as a special case of 2PC as follows: the server input is the node content of the decision tree, and the client input is the attribute vector. The server's output is $\perp$ while the client's output is the decision tree evaluation on the attribute vector. According to Definition 2, the server should learn nothing about the client's attribute vector, and the client should learn nothing about the server's decision tree except the classification result. Unlike [28], [34], [2], we do not hide the fact that a decision tree evaluation is going on.

B. The Complete-Tree Protocol (Theorem 2)

In the following proof, we first describe the intuition of why our protocol is secure. Then, we describe how to construct a simulator in the ideal world. We conclude by arguing why the output of the simulator is indistinguishable from the output of the semi-honest adversary in the real world.

Security of Client. The information learned by the server at level dp is the secret share of a client attribute $x_j - x^C_{\mathsf{dp}}$ through $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$, which is independently and uniformly distributed because of x^C_{dp}. The other message received by the server is $\perp$ from $\mathcal{F}_{\mathsf{COT}}$ in Step 2c.

In more detail, let $\mathcal{A}_S$ be the semi-honest server in the real world; the ideal-world simulator $\mathcal{S}_S$ outputs a simulated view as $\mathsf{Sim}_S(\mathcal{T}, \perp) = (\mathcal{T}, r, ((r_1, \perp), \ldots, (r_{d-1}, \perp)))$, where $\mathcal{T}$ is the input decision tree, $r, r_1, \ldots, r_{d-1}$ are uniform and independent random strings from appropriate domains.

For $\mathsf{View}^{\Pi}_S(\vec{x}, \mathcal{T}) = (\mathcal{T}, r, ((x^S_1, \perp), \ldots, (x^S_{d-1}, \perp)))$, this real view is identically distributed as the simulated one. To see, $x^S_{\mathsf{dp}} = x_j - x^C_{\mathsf{dp}}$ is uniformly distributed and independent of the client's input $\vec{x}$, which is the same as r_{dp}, for all $\mathsf{dp} \in [1, d-1]$.

Security of Server. In the offline phase, the client receives "one-time-padded" secret shares of the decision tree $\mathcal{T}$. The one-time pads are derived via the random-oracle responses of XOR of a subset of secret keys $\{\mathsf{dsk}_{0,\mathsf{dp}}, \mathsf{dsk}_{1,\mathsf{dp}}\}_{\mathsf{dp} \in [1,d-1]}$. In the online phase, at level dp, the client obtains $\perp$ from $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$ and a secret key $\mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$ from $\mathcal{F}_{\mathsf{COT}}$. With all the secret keys obtained so far, the client can decrypt a single node of the encrypted decision tree at level $\mathsf{dp} + 1$. The client is not able to decrypt more than one node at the same level since it would require both the left and right keys in some previous parent nodes, but the client can get either the left key $\mathsf{dsk}_{0,\mathsf{dp}}$ or the right key $\mathsf{dsk}_{1,\mathsf{dp}}$ from $\mathcal{F}_{\mathsf{COT}}$ in level dp. After decryption, the client obtains secret shares of the node content, containing the permuted index of the traversed node, as well as the secret shared threshold, which leak no information about the decision tree $\mathcal{T}$. The client then obtains the secret share of the comparison result b^* from $\mathcal{F}_{\mathsf{COT}}$. With the share, the client is able to obtain the permuted index of the next node by traversing the randomized complete tree. Since the index is permuted, it does not leak the structure of the decision tree.

In more detail, the ideal-world simulator $\mathcal{S}_C$ outputs a simulated view $\mathsf{Sim}_C(\vec{x}, \mathcal{T}(\vec{x}))$ as follows.

Sim_C picks $(d-1)$ random secret keys $\hat{\mathsf{dsk}}_1, \ldots, \hat{\mathsf{dsk}}_{d-1}$, queries the random oracle on $\bigoplus_{i=1}^{\mathsf{dp}} \hat{\mathsf{dsk}}_i$, and records the response as r_{dp} for all $\mathsf{dp} \in [1, d-1]$. Sim_C simulates the encrypted secret shares of the decision tree by first setting all nodes to be uniformly and independently distributed random strings from the appropriate domain. Then, it chooses a random leaf v' and replaces it with $r_{d-1} \oplus \mathcal{T}(\vec{x})$. For the output of $\mathcal{F}_{\mathsf{COT}}$ at each level dp, Sim_C simulates it as $(b^*_{\mathsf{dp}} || \hat{\mathsf{dsk}}_{\mathsf{dp}})$, where b^*_{dp} is set according to the leaf-to-root path of the random leaf v'.

Let $\mathbb{Q} = \{\bigoplus_{\mathsf{dp}=1}^{d'} \mathsf{dsk}_{b,\mathsf{dp}}\}_{b \in \{0,1\}, d' \in [1,d-1]}$ be the set of random oracle queries made by the honest server in the real protocol when generating the encrypted client share of the tree $\mathcal{T}'$ in Algorithm 2. Let $\mathbf{Good} = \{\bigoplus_{\mathsf{dp}=1}^{d'} \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}\}_{d' \in [1,d-1]}$ be the set of random oracle queries made by the client when it follows our complete-tree protocol in Fig. 5. The queries in $\mathbf{Good}$ are the ones used by the server to encrypt the tree nodes along the root-to-leaf path of the leaf v'. If the client does not query the random oracle on inputs in $\mathbf{Bad} = \mathbb{Q} \setminus \mathbf{Good}$, then the encrypted client tree shares not on the root-to-leaf path of the leaf v' are uniformly random from the clients' view in the real protocol, which are the same as Sim_C's simulation above.

Conditioned on the above event, the rest of the simulated view is also identically distributed as the real one. This is because in the real protocol, at each level dp, the client learns the permuted comparison result from $\mathcal{F}_{\mathsf{COT}}$, which is independently and uniformly distributed. In the simulated transcript, the simulator chooses a uniformly random leaf v', which also guarantees that the decision bit $\{b^*_{\mathsf{dp}}\}$ at each level is uniformly distributed. Similarly, the secret-shared contents $\mathcal{T}^C_{\mathsf{dp}} = y^C_{\mathsf{dp}} || \mathsf{ix}^C_{\mathsf{dp}}$ in each level $\mathsf{dp} \in [1, d-1]$ are independently and uniformly distributed values and thus are distributed identically in both the real transcript and the simulated one.

Finally, the probability that the client queries in the real protocol elements in $\mathbf{Bad}$ is negligibly small if $\mathbf{Bad}$ is a negligibly small subset of $\{0,1\}^{|\mathsf{dsk}|}$. This can be ensured by setting the length of dsk to be long enough, say, $(|\mathsf{dsk}| - d) > \lambda$. Note that if a node i' is not in the root-to-leaf path of v', there must exists a level k that $\mathsf{dsk}_{\bar{b}_k,k} \notin \{\mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}\}_{\mathsf{dp} \in [1,d-1]}$, which is required for computing $\left(\bigoplus_{\mathsf{dp}=1}^{d-1} \mathsf{dsk}_{b'_{\mathsf{dp}},\mathsf{dp}}\right)$ as input to the random oracle for decrypting node i determined by $b'_1, \ldots, b'_k (= \bar{b}_k), \ldots, b'_{d-1}$. But $\mathsf{dsk}_{\bar{b}_k,k}$ is independently and uniformly distributed from elements in $\{\mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}\}_{\mathsf{dp} \in [1,d-1]}$. Hence, the client cannot compute $\bigoplus_{\mathsf{dp}=1}^{d-1} \mathsf{dsk}_{b'_{\mathsf{dp}},\mathsf{dp}}$ other than making random guesses.

C. The Two-Party Sparse-Tree Protocol (Theorem 4)

Security of Client. This proof is similar to that for the complete-tree case since the client contribution in the sparse-tree protocol is the same as the complete-tree protocol.

Security of Server. In the sparse-tree construction, the key required to decrypt a node is stored in the content of its parent node such that one is not able to decrypt a node without decrypting its parent node. The (base) decryption key of a node is solely linked to their parent (which is also linked to previous nodes). Here we applied random permutation on all nodes for topology hiding. The client learns one of the (permuted by π and b') child indices after running $\mathcal{F}_{\mathsf{COT}}$ while remains oblivious of the other index. The same-level same-server-share principle is also applied to the child indices separately so that the server can remain oblivious of the client's current node. $\pi(0)$ is used for padding leaf nodes or dummy nodes, for the never-reaching right child of sinking nodes, and is a freshly sampled node index. Furthermore, the client learns the number of total nodes (with our padding) in the real protocol.

Given the tree meta-parameter, the simulator Sim_C samples the designated number of nodes, $\mathcal{T}'_1, \ldots, \mathcal{T}'_{2m+d-1}$ (from the domain of each node entry), similar to the complete-tree case. It randomly chooses secret keys $\hat{\mathsf{msk}}_1, \ldots, \hat{\mathsf{msk}}_{d-1}$, queries the random oracle on $\hat{\mathsf{msk}}_{\mathsf{dp}}$, and records the response as r_{dp} for all $\mathsf{dp} \in [1, d-1]$. The simulator randomly samples distinct indices $\{\mathsf{kid}^*_1, \ldots, \mathsf{kid}^*_{d-2}, \mathsf{kid}^*_{d-1}\}$ (let $\pi(1) \leftarrow \mathsf{kid}^*_1$). Among $2m+d-1$ nodes, the simulator replaces the (kid^*_{d-1})-th node with $r_{d-1} \oplus (\mathcal{T}(\vec{x}) \oplus v^S || \mathsf{random\_bits})$ for some suitably-long string $\mathsf{random\_bits}$, which is now the target leaf node. So, $\{\mathsf{kid}^*_1, \ldots, \mathsf{kid}^*_{d-1}\}$ is the node indices from the root to the target leaf node. v^S is computed as the "decrypted" sum of all nodes' $v_{\mathsf{kid}^*_{\mathsf{dp}}}$ along the path using $(r_{\mathsf{dp}}, \mathsf{kid}^*_{\mathsf{dp}})$. Sim_C sends $(\pi(1), \mathcal{T}'_1, \ldots, \mathcal{T}'_{2m+d-1}, v^S)$ to the adversary.

Similar to the complete-tree protocol, the simulator can tweak $\mathsf{kid}, b_{\mathsf{dp}}$ and the key in $\mathcal{F}_{\mathsf{COT}}$ to lead the client to the target leaf. In more detail, the root node is in plaintext, the simulator supplies via the $\mathcal{F}_{\mathsf{COT}}$ $b||[(\mathsf{kid}^*_1||\hat{\mathsf{msk}}_1) \oplus (\mathsf{kid}^C_{b,1}||\mathsf{msk}^C_1)]$ where b is a random bit and $(\mathsf{kid}^C_{b,1}||\mathsf{msk}^C_1)$ is the respective part of $\mathcal{T}'_{\pi(1)}$. The simulator moves on to the next level and can "decrypt" the next node using $r_{\mathsf{dp}+1}, \mathsf{kid}^*_{\mathsf{dp}+1}$. Thus, the simulator can simulate the view of the adversary at each level. Let $\mathbf{Good} = \{\mathsf{msk}_{\mathsf{dp}}\}_{\mathsf{dp} \in [1,d-1]}$ be the set of random oracle queries made by the client when it follows our sparse-tree protocol in Fig. 7. By a similar argument as the proof of Theorem 2, the encrypted clients' out-of-path tree shares are uniformly random as the clients' view in the real protocol. The adversary's view for normal decision nodes is indistinguishable from the simulated view.

It remains to show that going into a dummy decision node is indistinguishable from a real decision node. The dummy (or leaf) nodes are padded to contain $v_i = 0, \mathsf{kid}_{0,i} = 2m + d, \mathsf{kid}_{1,i} = 0$. All are in shared form and encrypted by the node's key, which we have shown that the client can only learn one node at each level. The indices kid's are also permuted and switched according to π and b'_{dp}. The adversary cannot distinguish between two cases better than making a random guess on b'_{dp}.

Appendix C
Maliciously-Secure Version

Malicious adversaries can deviate from the protocol execution arbitrarily. The standard way of defining security in this setting is to formalize an ideal process that involves a trusted party who computes the protocol result directly. A protocol is said to be secure if any adversary in the real protocol execution can be simulated by a simulator in the ideal model.

Execution in the ideal model. In an *ideal execution*, the two parties submit their inputs to the trusted party, who will compute the desired output and send the outputs back. An honest party just directs its true input for the computation to the trusted party, while a malicious party may replace its input with any other value of the same length. Let $f = (f_1, f_2)$ be a deterministic function, and $\mathcal{A}_i$ be a PPT adversary that corrupts party $i \in \{1, 2\}$, the *ideal execution of* f on inputs (x, y) and auxiliary input z to $\mathcal{A}$, denoted by $\mathsf{Ideal}_{f,\mathcal{A}_i(z)}(x, y)$, is defined as the output pair of the honest party and $\mathcal{A}_i$.

Execution in the real model. In the *real model*, the honest party follows the instructions of the protocol Π to interact with $\mathcal{A}_i$, who can adopt any polynomial-time strategy. Let f and $\mathcal{A}_i$ to be the same as defined above, and let Π be a 2PC protocol for computing f. Then, the *real execution of* Π on inputs (x, y) and auxiliary input z to $\mathcal{A}_i$, denoted by $\mathsf{Real}_{\Pi,\mathcal{A}_i(z)}(x, y)$, is defined as the output vector of the honest party and $\mathcal{A}_i$.

Security. Security is defined by requiring that adversaries (often called simulators in this context) are able to simulate the protocol execution in the real world.

Definition 3 (Malicious Model)**.** *A protocol* Π *is said to securely compute* f *with abort in the presence of malicious adversaries if, for every PPT adversary* $\mathcal{A}_i$ *in the real model, there exists a PPT adversary* Sim_i *in the ideal model, where*

$$\{\mathsf{Ideal}_{f,\mathsf{Sim}_i}(x, y)\} \overset{c}{\equiv} \{\mathsf{Real}_{\Pi,\mathcal{A}_i}(x, y)\} \textit{ for } i \in \{1, 2\}.$$

We upgrade our base protocol to equip it with security against malicious adversaries. We highlight the parts for enforcing security against a malicious client in blue.

Algorithm 4 shows the modification to the encrypted tree share. Each node additionally stores a signature from the server on its level. The client can show it in the online evaluation protocol in Fig. 12 to show the possession of valid tree shares. The additional parts for malicious security for both are highlighted in blue.

During the online phase, the client commits to the feature vector and uses zero-knowledge proof to bind the two sub-protocols and the committed feature vector, so the client cannot modify its feature vector on-the-fly. The client also proves the knowledge of signature on the tree shares it supplied to the sub-protocols. By design, the client can only decrypt 1 node per level, which prevents the client from using different shares. The COT in which the client sends the first message as the receiver is right after the previous OT in which the client sends the last message as the sender. These two steps can be piggybacked with a single zero-knowledge proof confirming the consistency across these two sub-protocol invocations, but we separate it into two ZKPs in Fig. 12 for clarity.

Algorithm 4: (Maliciously-Secure) $\mathsf{GenClientTree}$

```
Input  : Server tree $\mathcal{T}$ and signing key $\mathsf{sk}$
Output: Encrypted client share of the tree $\mathcal{T}'$
        and all random choices of the server
1  $(b'_1, \ldots, b'_d) \leftarrow\$ \{0,1\}^d$                         // server random choices
2  $\pi \leftarrow \mathsf{PermuteTree}(\mathcal{T}, (b'_1, \ldots, b'_d))$              // Algorithm 1
3  $\mathsf{msk}_1 \leftarrow 0^\lambda$                  // must traverse the root (no privacy)
4  for $\mathsf{dp} = 1$ to $d-1$ do              // set $\mathcal{T}'_i$, derive key material
5      $y^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_{2^{t+1}}$, $\mathsf{ix}^S_{\mathsf{dp}} \leftarrow\$ \mathbb{Z}_n$            // server share at lv. dp
6      $\mathsf{dsk}_{0,\mathsf{dp}}, \mathsf{dsk}_{1,\mathsf{dp}} \leftarrow\$ \{0,1\}^\lambda$                  // keying material
7      foreach node $i$ at level $\mathsf{dp}$ do  // same lv. same share
8          $y^C_{\mathsf{dp}} \leftarrow y_i - y^S_{\mathsf{dp}}$, $\mathsf{ix}^C_{\mathsf{dp}} \leftarrow \mathsf{ix}_i - \mathsf{ix}^S_{\mathsf{dp}}$      // client share
9          $\mathcal{T}'_i \leftarrow (y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}})||\mathsf{Sig}_{\mathsf{sk}}(y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}}||\mathsf{dp})$
10         $\mathsf{msk}_{\mathsf{kid}_{0,i}} \leftarrow \mathsf{msk}_i \oplus \mathsf{dsk}_{0,\mathsf{dp}}$           // left child key
11         $\mathsf{msk}_{\mathsf{kid}_{1,i}} \leftarrow \mathsf{msk}_i \oplus \mathsf{dsk}_{1,\mathsf{dp}}$          // right child key
12 foreach node $i$ at level $d$ do $\mathcal{T}'_i \leftarrow v_i$                // label
13 foreach node $i \neq 1$ do $\mathcal{T}'_i \leftarrow \mathcal{T}'_i \oplus \mathcal{H}(\mathsf{msk}_i)$          // OTP
14 Send $(\mathcal{T}'_1, \ldots, \mathcal{T}'_{2^d-1})$ after applying $\pi$
```

Corrupted Server. The server's view consists of transcripts from $\binom{n}{1}$-OT (as a receiver) and conditional OT (as a sender). It does not receive any output. When the two sub-protocols are instantiated using maliciously-secure OT and GC, the views of the server can be indistinguishably simulated since it receives fresh randomness from $\binom{n}{1}$-OT. The server can only launch attacks on the decision tree share or the input to conditional OT, which the client obtains incorrect result.

Corrupted Client. The view of the client consists of the encrypted decision tree share and transcripts of $\binom{n}{1}$-OT (as a sender) and conditional OT (as a receiver). Similarly, the client cannot infer extra information from the sub-protocols. It obtains the permuted bit and a uniformly random key (one-time-pad) for decrypting the next node from conditional OT. The view is indistinguishable from that of the real protocol when a uniformly random value is used. The client can modify its attribute (inputs to the OT and conditional OT) or the decision tree share (input to the conditional OT) on-the-fly to alter the evaluation path. This only affects correctness. It can only obtain a single meaningful result (the path keys to a single leaf node) on its modified input by construction.

Correctness against Malicious Adversaries. Achieving correctness in the malicious setting without significantly changing the protocol design appears to be non-trivial. As an example, a recent study formalized the security model in the context of secure neural-network training protocol [43], which considered privacy against a single malicious adversary (in the 3-party setting) [4]. It argues that for any two inputs of the honest parties, the view of the adversary is indistinguishable. Even with privacy in the malicious setting, this notion does not provide correctness against malicious adversaries.

Appendix D
Less-than-Predicate Conditional OT

Suppose the sender inputs are $x_\mathsf{s}, y_\mathsf{s}, K_0, K_1$, and the receiver inputs are $x_\mathsf{r}, y_\mathsf{r}$ such that $x = x_\mathsf{r} + x_\mathsf{s} \mod 2^{t+1}$,

(Maliciously-Secure) Complete-Tree Online Protocol:

1) The client sets $\mathsf{msk}_1 \leftarrow 0^\lambda$ and $\mathsf{kid} \leftarrow 1$.
2) The client picks $x^C_{\mathsf{dp}} \leftarrow\$\ \mathbb{Z}_{2^{t+1}}$, $\mathsf{dp} \in [1, d-1]$. The client commits to its feature vector $\vec{c_x} \leftarrow \mathsf{Com}(\vec{x}; \vec{r})$ and the secret shares $\{c_{\mathsf{dp}} \leftarrow \mathsf{Com}(x^C_{\mathsf{dp}}; r_{\mathsf{dp}})\}_{\mathsf{dp}\in[1,d-1]}$.
3) **for** $\mathsf{dp} = 1$ **to** $d-1$ **do**
   a) The client parses $(y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}})||\sigma_{\mathsf{dp}} \leftarrow \mathcal{T}'_{\mathsf{dp}}$. If $\mathsf{Vf}(\mathsf{vk}, (y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}}||\mathsf{dp}), \sigma_{\mathsf{dp}}) = 0$, abort.
      - **for** $j = 1$ **to** n **do** sets $X_j \leftarrow x_{j+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}}$.

   b) Both invoke (maliciously-secure) $\mathcal{F}_{\binom{n}{1}\text{-}\mathsf{OT}}$.
      - The client inputs $(X_1, \ldots, X_n)$ as the sender and proves the knowledge of (σ_{dp}, $\mathsf{ix}^C_{\mathsf{dp}}$, dp, x^C_{dp}, r_{dp}, $\vec{x}$, $\vec{r}$) such that X_j's are computed honestly with respect to the commitments above, *i.e.*, (1) $X_j = x_{j+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}}, \forall j \in [1, n]$; (2) σ_{dp} signs on $(y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}}||\mathsf{dp})$.
      - The server inputs $j \leftarrow \mathsf{ix}^S_{\mathsf{dp}}$ as the receiver.

      The server thus gets $X_{\mathsf{ix}^S_{\mathsf{dp}}} = x_{\mathsf{ix}^S_{\mathsf{dp}}+\mathsf{ix}^C_{\mathsf{dp}}} - x^C_{\mathsf{dp}} = x_{\mathsf{ix}_{\mathsf{dp}}} - x^C_{\mathsf{dp}} = x^S_{\mathsf{dp}}$, where $\mathsf{ix}_{\mathsf{dp}} = \pi^{-1}(\mathsf{kid}_{\mathsf{dp}})$ is the index of the current node.
   c) Both parties invoke (maliciously-secure) $\mathcal{F}_{\mathsf{COT}}$:
      - The server inputs $(x^S_{\mathsf{dp}}, y^S_{\mathsf{dp}}, K_0, K_1)$ as the sender, where $K_0 \leftarrow b'_{\mathsf{dp}}||\mathsf{dsk}_{0,\mathsf{dp}}$ and $K_1 \leftarrow (1 \oplus b'_{\mathsf{dp}})||\mathsf{dsk}_{1,\mathsf{dp}}$.
      - The client inputs $(x^C_{\mathsf{dp}}, y^C_{\mathsf{dp}})$ as the receiver.

      The client proves in zero-knowledge that the two inputs $(x^C_{\mathsf{dp}}, y^C_{\mathsf{dp}})$ for this COT are consistent with respect to the prior OT and commitments, *i.e.*, (1) σ_{dp} signs on $(y^C_{\mathsf{dp}}||\mathsf{ix}^C_{\mathsf{dp}}||\mathsf{dp})$ and (2) x^C_{dp} was committed in Step 2.
   d) The client parses $K_{b_{\mathsf{dp}}}$ as $b^*||\mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$, where $b_{\mathsf{dp}} = (x^C_{\mathsf{dp}} + x^S_{\mathsf{dp}}) < (y^C_{\mathsf{dp}} + y^S_{\mathsf{dp}})$ and $b^* = b_{\mathsf{dp}} \oplus b'_{\mathsf{dp}}$, sets
      - $\mathsf{kid}_{\mathsf{dp}+1} \leftarrow 2\mathsf{kid}_{\mathsf{dp}} + b^*$,
      - $\mathsf{msk}_{\mathsf{dp}+1} \leftarrow \mathsf{msk}_{\mathsf{dp}} \oplus \mathsf{dsk}_{b_{\mathsf{dp}},\mathsf{dp}}$,

      and decrypts $\mathcal{T}'_{\mathsf{dp}+1} \leftarrow \mathcal{T}'_{\mathsf{kid}_{\mathsf{dp}+1}} \oplus \mathcal{H}(\mathsf{msk}_{\mathsf{dp}+1})$.
4) At last, the client outputs the classification $v \leftarrow \mathcal{T}'_d$.

Fig. 12: Online Phase for the Maliciously-Secure Version (with differences from the semi-honest protocol highlighted)

$y = y_r + y_s \mod 2^{t+1}$ and $x, y < 2^t$. The receiver will only learn K_b where $b = (x < y)$ while the sender learns nothing.

Lemma 1 ([15]). *Let $x = x_r + x_s \mod 2^{t+1}$, $y = y_r + y_s \mod 2^{t+1}$, $z_r = y_r - x_r \mod 2^{t+1}$, $z_s = y_s - x_s \mod 2^{t+1}$, s.t. $x, y < 2^t$. If $x \neq y$, $(x < y) = c_r \oplus c_s \oplus (w_r < 2^t - w_s)$, where $c_r = (z_r < 2^t)$, $c_s = (z_s < 2^t)$, $w_r = z_r \mod 2^t$, and $w_s = z_s \mod 2^t$.*

From Lemma 1, a secure comparison protocol with shared inputs can be constructed from a secure comparison protocol with plain inputs, similar to garbled circuits. Below, we use $\mathcal{F}_{\binom{2}{1}\text{-}\mathsf{OT}}$ to construct conditional OT for less-than predicate.

Let h be a hash function of λ-bit output and $\mathcal{H}$ be a KDF. The sender and the receiver interact as follows.

1) The sender computes $z_s \leftarrow y_s - x_s \mod 2^{t+1}$, $c_s \leftarrow (z_s < 2^t)$, $w_s \leftarrow z_s \mod 2^t$.
   The receiver computes $z_r \leftarrow y_r - x_r \mod 2^{t+1}$, $c_r \leftarrow (z_r < 2^t)$, $w_r \leftarrow z_r \mod 2^t$.
2) The sender and receiver use GC to perform secure integer comparison with inputs w_s and $2^t - w_r$. The receiver outputs the output label $k_{b'}$ corresponding to the output $b' = (w_s < 2^t - w_r)$. Define $k_1 = k_0 \oplus \Delta$.
3) The sender and the receiver invoke $\mathcal{F}_{\binom{2}{1}\text{-}\mathsf{OT}}$. The sender picks $k'_0 \leftarrow\$\ \{0,1\}^\lambda$, computes $k'_1 \leftarrow k'_0 \oplus \Delta$, and inputs (k'_{c_s}, k'_{1-c_s}). The receiver inputs c_r.
4) Let $k^*_0 = k_0 \oplus k'_0$, and $k^*_1 = k^*_0 \oplus \Delta$. The receiver computes $k^*_b \leftarrow k_{b'} \oplus k'_{c_s \oplus c_r}$, where $b = (x < y)$.
5) The sender picks $s \leftarrow \{0, 1\}$ and sets $s' \leftarrow s \oplus c_s$. The sender computes $H_0 \leftarrow h(k_{s'})$ and $H_1 \leftarrow h(k_{1-s'})$, $K'_0 \leftarrow \mathcal{H}(k^*_s) \oplus K_s$, and $K'_1 \leftarrow \mathcal{H}(k^*_{1-s}) \oplus K_{1-s}$. The sender sends H_0, H_1, K'_0, K'_1 to the receiver.
6) The receiver computes $h(k_{b'})$. If $h(k_{b'}) = H_0$, the receiver sets $s^* \leftarrow c_s$, else if $h(k_{b'}) = H_1$, the receiver sets $s^* \leftarrow 1 - c_s$. Note that $s^* = s \oplus c_r \oplus c_s \oplus b' = s \oplus b$. The receiver outputs $K_b \leftarrow \mathcal{H}(k^*_b) \oplus K'_{s^*}$.

The protocol requires 1 communication round as the OT, GC, and the transfer of H_0, H_1, K'_0, K'_1 can be done in parallel. The following theorem shows that our protocol is correct.

Theorem 5. *When the underlying GC is correct (i.e., the parties obtain the intended outputs), if both the client and the server follow our conditional OT protocol, the receiver learns K_b where $b = (x < y)$ at the end.*

Proof: In Step 2, the receiver learns $k_{b'}$ where $b' = (w_s < 2^t - w_r)$ from GC. In Step 4, the receiver learns $k'_{c_r \oplus c_s}$ through OT. By XOR-ing both values, it has $k^*_b = (k_0 \oplus k'_0) \oplus (b' \oplus c_r \oplus c_s)\Delta$. In Step 6, the receiver computes s^* based on $h(k_{b'})$, which equals $s \oplus b$; when $h(k_{b'}) = H_0$, we have $h(k_{b'}) = h(k_{s'}) = h(k_{s \oplus c_s})$, which means $b' \oplus s \oplus c_s = 0$. Thus $s^* = c_r = c_r \oplus b' \oplus s \oplus c_s = s \oplus b$. Similarly, if $h(k_{b'}) = H_1$, we have $s^* = 1 \oplus c_r = b' \oplus s \oplus c_s \oplus c_r = s \oplus b$. Therefore, $\mathcal{H}(k^*_b) \oplus K'_{s^*} = \mathcal{H}(k^*_b) \oplus K'_{s \oplus b} = \mathcal{H}(k^*_b) \oplus \mathcal{H}(k^*_b) \oplus K_b = K_b$. ■

Theorem 6. *Suppose the GC is secure in the semi-honest model, the above conditional OT protocol securely implements $\mathcal{F}_{\mathsf{COT}}$ against semi-honest adversaries in the $\mathcal{F}_{\binom{2}{1}\text{-}\mathsf{OT}}$-hybrid model, with h and $\mathcal{H}$ being a secure hash function and KDF.*

Proof: (Sketch) From Lemma 1, if $x \neq y$, it holds that $(x < y) = c_r \oplus c_s \oplus (w_r < 2^t - w_s)$. The partial key $k_{b'}$ obtained by the receiver from the GC reveals nothing about the other party's input, where $b' = (w_r < 2^t - w_s)$. The message $k'_{c_s \oplus c_r}$ obtained by the receiver in the OT is a random string (k'_0 is random) and reveals nothing about the other message. The two keys received by the receiver are then combined and used to decrypt and obtain the correct conditional OT message. Finally, the receiver can only "decrypt" one of the two messages (K'_0, K'_1) using the combined key (as input to the KDF). The sender in $\mathcal{F}_{\mathsf{COT}}$ acts as the garbler in GC and the sender in $\mathcal{F}_{\binom{2}{1}\text{-}\mathsf{OT}}$. It receives no output and hence can be easily simulated. ■