Constant-Round Private Decision Tree Evaluation for Secret Shared Data

Nan Cheng
University of St.Gallen

Naman Gupta
IIT Delhi

Aikaterini Mitrokotsa
University of St.Gallen

Hiraku Morita
Aarhus University
University of Copenhagen
hiraku@cs.au.dk

Kazunari Tozawa
The University of Tokyo
tozawa.kazunari@mail.u-tokyo.ac.jp

ABSTRACT

Decision tree evaluation is extensively used in machine learning to construct accurate classification models. Often in the cloud-assisted communication paradigm cloud servers execute remote evaluations of classification models using clients' data. In this setting, the need for *private decision tree evaluation* (PDTE) has emerged to guarantee no leakage of information for the client's input nor the service provider's trained model *i.e.,* decision tree. In this paper, we propose a private decision tree evaluation protocol based on the three-party replicated secret sharing (RSS) scheme. This enables us to securely classify inputs without any leakage of the provided input or the trained decision tree model. Our protocol only requires constant rounds of communication among servers, which is useful in a network with longer delays.

Ma *et al.* (NDSS 2021) presented a lightweight PDTE protocol with sublinear communication cost with linear round complexity in the size of the input data. This protocol works well in the low latency network such as LAN while its total execution time is unfavourably increased in the WAN setting. In contrast, Tsuchida *et al.* (ProvSec 2020) constructed a constant round PDTE protocol at the cost of communication complexity, which works well in the WAN setting. Although their construction still requires 25 rounds, it showed a possible direction on how to make constant round PDTE protocols. Ji *et al.* (IEEE Transactions on Dependable and Secure Computing) presented a simplified PDTE with constant rounds using the function secret sharing (FSS) at the cost of communication complexity.

Our proposed protocol only requires five rounds among the employed three servers executing secret sharing schemes, which is comparable to previously proposed protocols that are based on garbled circuits and homomorphic encryption. To further demonstrate the efficiency of our protocol, we evaluated it using real-world classification datasets. The evaluation results indicate that our protocol provides better concrete performance in the WAN setting that has a large network delay.

KEYWORDS

secure computation, secret sharing, private decision tree evaluation

Proceedings on Privacy Enhancing Technologies 2024(1), 397–412

https://doi.org/10.56553/popets-2024-0023

1 INTRODUCTION

1.1 Background

Privacy-preserving machine learning (PPML) enables us to analyze a large amount of data without revealing any sensitive information. More precisely, PPML allows service providers to compute statistics and learning models using sensitive datasets (*e.g.,* healthcare data or individual DNA information) and perform advanced data analytics while providing high privacy guarantees to the involved clients; making people feel safe and less hesitant to upload sensitive information to access specific services. Thus, PPML can support and allow large-scale data collection and advanced data analytics. PPML mainly focuses on the design of privacy-preserving machine learning models computed via deep neural networks (DNN), linear regression, logistic regression, support vector machine classification, as well as decision tree classification. One of the main challenges in PPML research is execution speed, so it is important to construct a practical and efficient PPML scheme.

Research on PPML has received significant attention employing different cryptographic primitives. Some of the most popular PPML approaches are based on secure multi-party computation (MPC) schemes such as fully homomorphic encryption (FHE) [24], garbled circuits [51], secret sharing, or a combination thereof. Some recent work in PPML includes privacy-preserving DNN such as Chameleon [43], Gazelle [29], and SecureML [38] in the two-party setting, ABY^3 [37], SecureNN [48], FALCON [49], and Adam in Private [4] in the three-party setting, as well as FLASH [13] and Trident [16] in the four-party setting. The main objective in this line of research is secure prediction (evaluation) and their feasibility has been demonstrated using small datasets such as the MNIST dataset. Another line of research has focused on achieving privacy-preserving and secure training [4, 13, 37, 38, 48].

The main focus of this paper is *Private Decision Tree Evaluation* (PDTE), which is one of the main tasks of PPML. Decision trees are widely used in machine learning and have many applications in medicine (*e.g.,* remote diagnosis) or finance. In the cloud-assisted communication paradigm, cloud servers allow remote evaluations of classification models. In this setting, remote evaluation requires that the model remains secret and known only to the service provider, while the service provider should not find out the client's input data. Thus, the need for PDTE has emerged to guarantee no information leakage for the client's input and service provider's decision tree.

In this study, we adopt a secret-sharing-based MPC scheme as the underlying system. Secret-sharing-based MPC allows efficient

computation of various functions with a small amount of communication. For instance, there have been many privacy-preserving protocols for basic operations such as less-than/equality check [14, 18, 20, 40, 41], division [8, 26, 39], shuffle [15], and database join [34], just to name a few. Among this line of work, [20, 40, 41] provided a constant-round less-than protocol over a field and [46] provided a constant-round less-than protocol over a ring. These works have shown that MPC can run at practical speeds since the main overhead of secret-sharing-based MPC arises from the communication delay depending on the number of communication rounds in many cases.

1.2 Related Work

Existing private decision tree evaluation (PDTE) protocols rely on homomorphic encryption (HE) and garbled circuits (GC) [6, 11, 44, 47, 50], secret sharing (SS) schemes [27, 36, 46] or a combination of these primitives. The best choice depends on the environment because these primitives often have a trade-off regarding the required computation cost, communication cost, the targeted computed circuit size and scalability of the threat models. We mainly consider environments where communication delays are dominant, such as WAN settings. An effective goal in such settings is to reduce the number of communication rounds, preferably regardless of datasize.

Most of the existing PDTE protocols that achieve constant communication rounds are based on HE or GC. However, instead of high round complexity, these schemes often require high computation costs, communication complexity, or the use of limited computation settings. For instance, Wu *et al.* [50] only considered the two-party *client-server setting* where the server holds a trained decision tree and the client holds a feature vector as input. The scheme of Wu *et al.* employs oblivious transfer (OT) and additive HE. Subsequently, Tai *et al.* [44] have improved Wu *et al.* [50]'s scheme significantly in the semi-honest setting, but the scheme still involves heavy cryptographic primitives, additive HE and OT.

As opposed to the above approach, another line of research has proposed PDTE protocols [27, 28, 36, 46] based on secret-sharing (SS) schemes. The advantage of SS-based protocols is their low computational cost and low communication volume, compared to HE-based and GC-based protocols, which generally require heavy computations. On the other hand, a general drawback of SS-based protocols is the large number of communication rounds. For instance, the PDTE protocols that were introduced in [27, 36] require $O(d)$ rounds for a decision tree with height d.

Recently, several SS-based constant-round PDTE protocols have been proposed. Tsuchida *et al.* [46] have performed a significant improvement in this line of research by proposing the first constant communication round (25 rounds) PDTE protocol, where the number of rounds is independent of the height of the tree model. Their proposed protocol is defined over a residue ring (contrary to all previous works relying on finite fields) providing important improvements in computation and communication cost while relying on a secret sharing scheme and three-party computation. Ji *et al.* [28] claimed that their PDTE protocol required only 4 communication rounds using a function secret sharing (FSS) technique. However, the protocol has some drawbacks; (i) its comparison phase allows only one type of comparison operation, (ii) the PDTE protocol cannot be deployed directly as a subprotocol unless it executes an additional share transformation operation since the final output of the servers is not in the same secret sharing format as the input, and (iii) it needs to store a lot of pre-computed randomness for FSS (*i.e.,* FSS keys). We highlight the difference with our proposed protocol in more detail in Appendix B.

In this paper, we investigate the question of whether it is possible to *further improve* the performance of a constant communication round PDTE protocol that relies on secret sharing and is defined over a residue ring. In particular, we aim to improve the communication complexity and computational cost compared to the SS-based schemes of Tsuchida *et al.* [46] and Ji *et al.* [28].

1.3 Contribution

In this paper, we present a private decision tree evaluation (PDTE) protocol that admits a *five-round* online phase. Our scheme works with a 2-out-of-3 replicated secret sharing ($(2,3)$-RSS) scheme over a ring, so it does not rely on heavy cryptographic primitives such as OT or HE. In particular, we focus on the *outsourced setting*, in which the computing servers cannot know or use any sensitive information about the input and output data. The outsourced setting provides stronger security than the client-server setting and enables a wider range of applications such as federated learning. For those who are interested in the PDTE protocols in the non-outsourced setting, please refer [21, 36].

Fig. 1 shows a possible scenario of the outsourced setting using $(2,3)$-RSS. As shown in Table 1, the round complexity of our scheme is 5 times more efficient than the current state-of-the-art constant-round protocol [46].

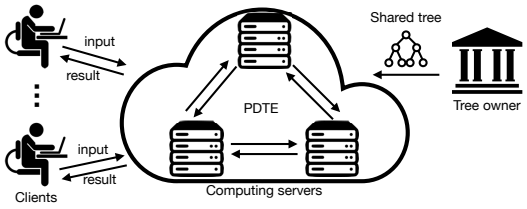


Figure 1: Possible scenario: The tree owner has a binary decision tree as a trained model, while each client has an attribute vector as input to the service. The computing servers only obtain shares of all values from the clients and the tree (model) owner. The classification result obtained via the decision tree can be seen only by the client who sent a query.

Our proposed PDTE protocol is secure in the presence of semi-honest adversaries in the honest majority setting and is composed of three novel sub-protocols: *(i)* a one-round feature selection protocol, *(ii)* a two-round comparison protocol, and *(iii)* a two-round path evaluation protocol. Table 1 shows an overview of our protocol and highlights the difference with previous work.

1.3.1 Feature Selection Protocol. Our feature selection protocol uses a randomize-then-reveal technique for a secret shared vector, more specifically, a roulette-then-reveal technique, instead of using relatively heavy public key cryptographic primitives such as FHE.

The same functionality as the feature selection protocol has also been achieved by circuit-based constructions using general

Table 1: Comparison of Outsourced PDTE Protocols

		Required Operations			Round
		FeatSelect	Compare	PathEval	
2PC	[35]	No obliviousness	μ HE + μ MT	$h\mu$ MT	$h+3$
	[52]	$n\mu$ MT	$m\mu$ MT	SS	$O(m+h)$
	[36]	$h \times (1,n)$-OT	h GC	$h \times (1,2)$-OT	$2h$
3PC	[46]	$(2,2)$-SS, $(2,3)$-RSS	$(2,3)$-RSS	$(2,3)$-RSS	25
	[28]	FSS	FSS	FSS	4*
	Ours	$(2,3)$-RSS	FSS, $(2,3)$-RSS	$(2,3)$-RSS	5

* Ji *et al.* [28] does not achieve the same functionalities as in [46] and ours. A straightforward solution requires two more rounds. See Appendix B for more details.

* h is the depth of a decision tree. μ is the number of the inner nodes, *i.e.*, , $\mu = 2^h - 1$. n is the size of a feature vector. 'HE' means a number of operations of homomorphic encryption, 'MT' means a number of multiplication triples, 'SS' means secret sharing, 'GC' means a garbled circuit, and 'OT' means an oblivious transfer. '$(2,2)$-SS' stands for 2-out-of-2 additive secret sharing and '$(2,3)$-RSS' stands for 2-out-of-3 replicated secret sharing. 'FSS' stands for function secret sharing. Note that $(2,3)$-RSS helps us to remove the relatively heavy public key crypto techniques such as OT and generation of MT.

MPC framework [7, 30, 33] and oblivious random access machines (ORAM) [12, 23, 32].

Laud [33] presented a feature selection protocol using general MPC, which achieved $O(m)$ round offline phase and a constant round online phase, where m is an array size. Blanton *et al.* [7] also introduced a general construction of the feature selection protocol for any number of computing parties under the Shamir secret sharing, which achieved constant online/offline rounds. Our result will be put among these constructions as a special case of the three-party setting that achieves a constant round online/offline phase. Please see Table 2 for comparison.

Faber *et al.* [23] use a "data-rotation" operation, which is similar to our roulette operation. However, their binary-tree ORAM approach proceeds a layer by layer in a tree, which takes a logarithmic order of communication rounds for an input size, while our feature selection protocol only requires one (constant) communication round. Bunn *et al.* [12] improved such an ORAM approach to obtain a constant round 3-party ORAM by using a distributed point function (DPF). However, their construction requires the client to generate input-dependent DPF keys, which we would like to avoid because the input to the feature selection protocol is not always from clients or other entities who know the input value itself. In other words, the input to the feature selection can be a secret shared value that nobody knows its original value.

Roulette To construct an efficient feature selection protocol, we introduce the Roulette protocol that computes the circular shifted value from input in the shared form in 3PC, *i.e.,* an input $(\llbracket x_0 \rrbracket^N, \llbracket x_1 \rrbracket^N, \ldots, \llbracket x_{m-1} \rrbracket^N)$ will be $(\llbracket x_n \rrbracket^N, \llbracket x_{n+1} \rrbracket^N, \ldots, \llbracket x_{m-1} \rrbracket^N, \llbracket x_0 \rrbracket^N, \ldots, \llbracket x_{n-1} \rrbracket^N)$ for a shared randomness $\llbracket n \rrbracket^N$. Although a roulette protocol in 2PC was introduced in [5], it was not trivial to achieve the same functionality in the $(2,3)$-RSS setting.

The key idea to construct it in the RSS setting is to generate appropriate correlated randomness by Rouletteprep. Well-structured randomness helps servers to reduce the number of communications between each other by cancelling out an accumulated term. Both Roulette and its offline sub-protocol Rouletteprep help to construct an efficient feature selection protocol and are of independent interest since they are related to applications such as a secure database search or an oblivious array read.

Note that Araki *et al.* [3] constructed an efficient secure shuffle protocol, which can be seen as a general case of our roulette protocol. However, our roulette protocol takes the best advantage of its simpler functionality than shuffle protocols and it only requires one communication round while the secure shuffle protocol requires two communication rounds.

1.3.2 Comparison Protocol. To construct an efficient comparison protocol, we use the existing less-than protocol and equality check protocol constructed in the 2 + 1 server function secret sharing (FSS) setting [9]. These protocols cannot be smoothly adopted in our setting because of the difference in a share type, where the FSS setting uses a $(2,2)$-secret sharing (SS) scheme, while ours uses a $(2,3)$-RSS scheme.

SC-AND Thus, we introduce the SC-AND protocol that enables us to execute share conversion from $(2,2)$-SS to $(2,3)$-RSS and execute the AND protocol at the same time, which only requires 1 round.

1.3.3 Path Evaluation Protocol. We introduce a new MPC primitive for randomizing tree models, *oblivious tree shuffle*, to achieve a two-round protocol for path evaluation.

Oblivious Tree Shuffle. Oblivious tree shuffling takes a shared tree as input and outputs a shared shuffled tree with node-wise random flip, which preserves the relations between a parent node and its child node. Thus, it allows us to use the shuffle-and-reveal technique for a binary tree to obtain a required leaf value.

Our tree permutation technique is inspired by the "PermuteTree" protocol that has a logarithmic round complexity by Ma *et al.* [36]. We improved it to realize a constant round tree permutation. Now, we highlight the difference between our construction and Ma *et al.*'s.

First of all, Ma *et al.*'s PermuteTree protocol prepares h random bits to randomize a decision tree for the tree's height h, where we consider $h \approx \log N$ for a bit length of inputs, N. Using such randomness, their PermuteTree randomizes all nodes in the same layer using the same flip-bit. This technique is essentially the same as the XOR permutation in [5]. In contrast, our tree permutation prepares 2^h random bits and every node is randomized independently using a different random bit, which enables the algorithm to execute within constant communication rounds. Although this increases storage for pre-computed randomness, the reduction of round complexity helps to shorten the execution time.

Secondly, a path evaluation protocol in [36] evaluates node labels from a root node to a leaf node one by one. Thus, its communication cost in terms of data amount is small but it requires $O(h)$ rounds. In such an algorithm, using XOR permutation is enough to hide intermediate information. In contrast, our path evaluation protocol executes a randomize-then-reveal for all node labels. In our setting, an XOR permutation will not generate uniformly random labels. Therefore, we invented the tree permutation that helps not to leak any information even after revealing (randomized) labels.

1.3.4 Overview. We now give an overview of our results.

Constant Round Protocols in Secret Sharing. To the best of our knowledge, Tsuchida *et al.* [46] introduced the first constant-round PDTE protocol using a secret sharing scheme over a residue ring. Our PDTE protocol requires only 5 online rounds, while [46]'s

Table 2: Comparison of Feature Selection (One execution)

Protocol	Share of Array	Share of Index	Round		Communication [bits/all parties]	
			Online	Offline	Online	Offline
Laud *et al.* [33]	$(t+1,n)$-SSS on $\mathbb{F}_p$	$(t+1,n)$-SSS on $\mathbb{F}_p$	2	$m-2$	$2n(n-1)\log p$	$(m-2)n(n-1)\log p$
Blanton *et al.* [7] in MPC	$(t+1,n)$-SSS on $\mathbb{F}_p$	$(t+1,n)$-SSS on $\mathbb{F}_p$	5	-	$(4m\log\log m+\log m+2)n(n-1)\log p$	-
Blanton *et al.* [7] in 3PC	$(2,2)$-ASS on $\mathbb{Z}_{2^k}$	$(3,3)$-ASS on $\mathbb{Z}_m$	2	-	$4mk$	-
Tsuchida *et al.* [46]	$(2,3)$-RSS on $\mathbb{Z}_{2^k}$	$(2,3)$-RSS on $\mathbb{Z}_{2^k}$	11	2	$k'(15k+(3k-3)\log p+4)+6mk'+3mk+3k$	$(6k^2+mk+2m-k)k'$
Ji *et al.* [28]	$(2,3)$-RSS on $\mathbb{Z}_{2^k}$	$(3,3)$-ASS on $\mathbb{Z}_m$	1	1	$12m$	$6((m+1)\lambda+2m+k)$
Ours (Protocol 4)	$(2,3)$-RSS on $\mathbb{Z}_N$	$(2,3)$-RSS on $\mathbb{Z}_m$	1	2	$4mk+3k'$	$10mk$

*'SSS' means Shamir's secret sharing, 'ASS' means Additive secret sharing, and 'RSS' means Replicated secret sharing.
n is the number of parties, p is the smallest prime number greater than k, and λ is security parameter for FSS.

PDTE requires 25 online rounds. Table 5 shows a more detailed comparison of the required round complexity.

Our feature selection protocol requires only 1 online round, the comparison protocol requires 2 online rounds, and the path evaluation protocol requires 2 online rounds. These make the total required rounds to be equal to 5. Since all offline communication can be done at once, the total number of offline rounds is equal to 2.

Compared to the state-of-the-art round-efficient scheme [28], our scheme achieves an asymptotic improvement in terms of online communication complexity. Our protocol does not have to fully rely on functional secret sharing schemes as in Ji *et al.* [28]. Thus we achieved less communication complexity and less storage cost than [28].

Use Cases. We highlight the use cases where our protocol fits well. We succeeded in reducing the round complexity at the expense of the larger communication complexity compared to linear-round protocols such as Ma *et al.* [36]. Our protocol is particularly advantageous when used in a network with a significant delay, such as real-world WAN. For BREAST dataset, we have a running time of 0.86 seconds in the WAN setting with 160 ms of network latency while Ma *et al.* [36] requires approximately 3 seconds in the WAN setting with a smaller network delay, 80 ms. See Sect. 4.2.

Our proposed protocol is designed to work well in the offline/online paradigm. Compared to prior work [28], the online phase of our scheme allows us to avoid some random generation operations such as FSS key generation. This reduces online computational complexity and allows larger decision-making models, such as DIABETES-18, in constant round communication, as shown in Table 7.

Experiment with Real-World Datasets. We ran experiments on four classical real-world datasets; Wine, Linnerud, Breast cancer, and Digits dataset from the UCI machine learning repository [22]. For each dataset, we obtained a decision tree as a trained model in the clear and adjusted the decision tree to be a perfect binary tree by adding a certain amount of dummy nodes. Then, we executed our PDTE protocol for each model and measured the required execution time. As shown in Table 7, our evaluation experiments demonstrate that our secure protocol finishes the evaluation within a reasonably short time. Thus, our PDTE protocol is not only based on a solid theoretical foundation, but it is also very efficient and practical.

Table 3: Notation Table

Secret Sharing	
$[\![x]\!]^N = ([\![x]\!]^N_1, [\![x]\!]^N_2, [\![x]\!]^N_3)$	2-out-of-3 replicated secret sharing (RSS) over $\mathbb{Z}_N$
$[\![x]\!]^N_i = (x_i, x_{i+1})$	Server i's share of value $x \in \mathbb{Z}_N$
$[\![\boldsymbol{v}]\!]^N_i = (\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(i+1)})$	Server i's share of vector $\boldsymbol{v} \in \mathbb{Z}^\ell_N$ (cf. $\boldsymbol{v}_i$ is the i-th entry of vector $\boldsymbol{v}$)
$[\![x]\!]^m$	2-out-of-3 RSS over $\mathbb{Z}_m$
$[\![x]\!]^B$	2-out-of-3 RSS over $\mathbb{Z}_2$
$\langle x \rangle$	2-out-of-2 additive secret sharing over $\mathbb{Z}_N$
$\langle x \rangle^B$	2-out-of-2 additive secret sharing over $\mathbb{Z}_2$
$\langle \pi \rangle$	2-out-of-3 RSS of permutation $\pi \in \mathcal{S}_n$
Decision Tree	
$\mathbb{Z}_N$	The domain of each attribute and the class labels
k	The bitlength of N, *i.e.,* $k = \lceil \log N \rceil$
m	The number of attributes
k'	The bitlength of m, *i.e.,* $k' = \lceil \log m \rceil$
h	The depth of decision tree
$(\mathsf{idx}_j, \tau_j, \mathsf{cond}_j)$	The label of decision node $j \in T_{h-1}$
val_ℓ	The label of end node $\ell \in \mathbb{Z}_{2^h}$

2 PRELIMINARIES

In this section, we describe the notation and the model we consider. Frequently referred notation is listed in Table 3.

2.1 Notation

Let N be a positive number of power 2. We write $\mathbb{Z}_N := \mathbb{Z}/N\mathbb{Z}$. We use bold symbols (*e.g.,* $\boldsymbol{v}$, $\boldsymbol{y}$) to represent vectors, and let $\boldsymbol{v}_i$ denote the i-th entry of $\boldsymbol{v}$. Let $\oplus$ be an operation of XOR of bits. When it is used with vectors, it denotes an element-wise XOR of vectors.

Let $\mathcal{S}_n$ denote the symmetric group on a set of size n. We denote by $\pi_2 \cdot \pi_1 \in \mathcal{S}_n$ the composite of π_1 and π_2. That is, $(\pi_2 \cdot \pi_1)(i) := \pi_2(\pi_1(i))$. For a vector $\boldsymbol{v}$ of length n, we write the permutation application action as $(\boldsymbol{v} \cdot \pi)_i := \boldsymbol{v}_{\pi(i)}$. In particular, when considering circular shift permutations, we write the result as $\mathsf{shift}(\boldsymbol{v}, r)$ if $(\mathsf{shift}(\boldsymbol{v}, r))_i = \boldsymbol{v}_{i-r \bmod n}$.

For $n \in \mathbb{Z}_N$, we let $n|_d$ denote the d-th digit of n. Here d starts from the least significant bit as 0. For example, $6|_0 = 0$, $6|_1 = 1$ and $6|_2 = 1$.

Let T_h denote the set of nodes in the complete binary tree of height h. Concretely, we write $T_h := \{(d, j) \mid 0 \le d \le h, 0 \le j \le 2^d - 1\}$. Here, (d, j) represents the j-th node (from left) of depth d. Accordingly, the parent node of (d, j) is $(d-1, \lfloor \frac{j}{2} \rfloor)$. We define the function $\mathsf{anc} : \{0, \ldots, h\} \times \mathbb{Z}_{2^h} \to T_h$ as $\mathsf{anc}(d, j) = (d, \lfloor \frac{j}{2^{h-d}} \rfloor)$, which means that node $\mathsf{anc}(d, j)$ is the ancestor node at depth d of the j-th leaf. Let $C_h := \mathbb{Z}_2^{T_h}$. Then C_h can be seen as the set of

assignments of a boolean label to each node in the complete binary tree with height h. For $\mathfrak{c} \in C_h$, we denote by $\mathfrak{c}_{d,j}$ the label for node (d, j).

Servers are represented by S_1, S_2, and S_3. Note that operations occurring at indices in S indicate modular arithmetic, *i.e.,* S_{i-1} turns to be S_3 for $i = 1$, and S_{i+1} turns to be S_1 for $i = 3$.

2.2 Replicated Secret Sharing

Throughout this paper, we use three types of secret sharing schemes.

We mainly use a 2-out-of-3 replicated secret sharing scheme [2, 19] and we just mention it as 'replicated secret sharing' or 'RSS'. The RSS scheme is specified by the following functionalities:

$[\![\cdot]\!]$-Sharing Given an input $x \in \mathbb{Z}_N$, this operation picks randomness $x_1, x_2, x_3 \in \mathbb{Z}_N$ such that $x = x_1 + x_2 + x_3$. Then, it sets $[\![x]\!]^N = ([\![x]\!]^N_1, [\![x]\!]^N_2, [\![x]\!]^N_3)$ as a share of x, where $[\![x]\!]^N_i = (x_i, x_{i+1})$ is for a server S_i, $i \in \{1, 2, 3\}$. Note that $x_{i+1} = x_1$ when $i = 3$.

We denote by $[\![\boldsymbol{x}]\!]^N = ([\![\boldsymbol{x}]\!]^N_1, [\![\boldsymbol{x}]\!]^N_2, [\![\boldsymbol{x}]\!]^N_3)$ the share of a vector $\boldsymbol{x}$, where $[\![\boldsymbol{x}]\!]^N_i = (\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(i+1)})$ for $i \in \{1, 2, 3\}$. Note that we use superscript notation $\boldsymbol{x}^{(i)}$ to distinguish them from vector elements; $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1})$.

Reconstruct Servers can communicate with each other to reconstruct the secret value x from a share of x. A naive reconstruction algorithm in the semi-honest setting is that each S_i sends x_i to S_{i+1} simultaneously and obtains the sum of the received value and the values in $[\![x]\!]^N_i$. We write Reconst for this algorithm.

We would like to note that we define a boolean share $[\![x]\!]^B$ in a similar manner by using randomness x_1, x_2, x_3 such that $x = x_1 \oplus x_2 \oplus x_3$ [2]. When considering two different domains, $\mathbb{Z}_N$ and $\mathbb{Z}_m$, of arithmetic shares, we will explicitly write $[\![\cdot]\!]^N$ and $[\![\cdot]\!]^m$ for shares over each domain. Also note that for any vector $\boldsymbol{v}$, we simply write $[\![\boldsymbol{v}]\!]$ for a vector consisting of shares of all the entries in $\boldsymbol{v}$.

$\langle\cdot\rangle$-Sharing As an adjunct to the RSS scheme, we also use the 2-out-of-2 additive secret sharing scheme. We will write $\langle x \rangle$ for an additive share (between S_i and S_j) of $x \in \mathbb{Z}_N$, which consists of two random values $(x_i, x_j) \in \mathbb{Z}^2_N$ satisfying $x = x_i + x_j$. Here, S_i and S_j hold $\langle x \rangle_i = x_i$ and $\langle x \rangle_j = x_j$, respectively.

When considering sharing a permutation, we use the RSS scheme for permutations. For $\pi \in \mathcal{S}_n$, we will write $\langle \pi \rangle$ for a RSS share of π. Here, S_i's share is defined as $\langle \pi \rangle_i = (\pi_i, \pi_{i+1})$ for random permutations π_1, π_2 and $\pi_3 \in \mathcal{S}_n$ satisfying $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$, *i.e.,* $\pi(i) = \pi_1(\pi_2(\pi_3(i)))$ for all $i \in \{0, \ldots, n-1\}$.

In these secret sharing schemes, we assume the following operations. Note that this paper counts the number of communication rounds assuming a duplex network, so it does not necessarily mean a "round trip". If two-way communication is possible simultaneously, we count it as 1 round. If one is performed depending on the other, we count it as two rounds, even if the directions differ.

Local operations Given $[\![x]\!]^N$, $[\![y]\!]^N$, and a public value $\alpha \in \mathbb{Z}_N$, the servers can compute shares of secure addition $[\![x + y]\!]^N$, multiplication with a public value $[\![\alpha \cdot x]\!]^N$, and an addition with a public value $[\![x + \alpha]\!]^N$ locally without any communication among the servers [2].

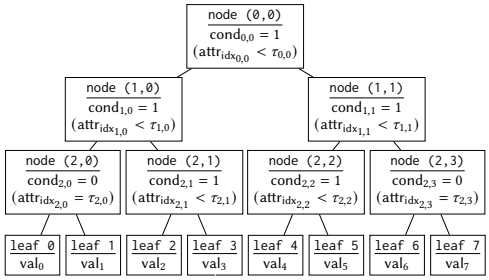


Figure 2: Example of a Decision Tree with Height 3.

Secure Equality Check and Comparison Given $\langle x \rangle$, $\langle y \rangle$, the servers can compute $\langle x \stackrel{?}{=} y \rangle^B$ and $\langle x \stackrel{?}{<} y \rangle^B$ by using precomputed correlated randomness. We write Equal and LessThan for these functionalities. Each requires one communication round [9].

Share Conversion Given $[\![x]\!]^N$, any two of the three servers can obtain $\langle x \rangle$ by local computation without communication [19]. We write ShareConv for the conversion. We assume that the output of ShareConv is uniformly random when viewed from the third server. Conversely, the servers can also convert $\langle x \rangle$ given by any two servers to $[\![x]\!]^N$, with one communication round. We will write $\mathsf{ShareConv}^{-1}$ for this inversion. Detailed definitions of our protocols for these operations can be found in Appendix.

Oblivious Shuffle Given a share $[\![\boldsymbol{x}]\!]^N$ of an n-degree vector and a share $\langle \pi \rangle$ of $\pi \in \mathcal{S}_n$, the servers can compute a share $[\![\boldsymbol{w}]\!]$ (or a revealed vector $\boldsymbol{w}$), where $\boldsymbol{w}$ is the vector obtained by applying π to $\boldsymbol{v}$. We write Shuffle (resp. ShuffleReveal) for the oblivious shuffling operation. Both of these operations require two communication rounds [17].

Random Share Generation When given a parameter N, the servers can generate a share $[\![r]\!]$, where $r \in \mathbb{Z}_N$ is picked uniformly at random. We will denote by $\mathsf{RndGen}(N)$ the functionality for generating random shares.

For the $(2, 3)$-RSS, there are two RndGen protocols depending on the security requirements proposed in [2]. In the information-theoretic security setting, RndGen is achieved with one-round communication if each party S_i picks a random value and sends it to S_{i+1}. In the computational security setting, RndGen can be achieved without communication, assuming that S_i and S_{i+1} hold a common secret key about which S_{i+2} knows nothing.

2.3 Security

In this paper, we consider security for three-party computation (3PC) protocols in the presence of a static semi-honest corruption of at most one server with no collusion. Semi-honest security is sufficient when the computing parties (servers) somewhat trust each other or when all secure operations are embedded in each server and are not tampered. We focus on semi-honest security and leave the development of a malicious secure PDTE protocol for future work.

For a three-party protocol Π among servers S_1, S_2, S_3, the variable $\mathsf{view}^{\Pi}_i(x_1, x_2, x_3)$ denotes the view of the server S_i during a real execution of a protocol Π on input (x_1, x_2, x_3). Here, $\mathsf{view}^{\Pi}_i(x_1, x_2, x_3)$

consists of S_i's input x_i, S_i's internal random coins, and all the messages received by S_i. The variable $\text{out}^{\Pi}(x_1, x_2, x_3)$ denotes the output of three servers from an execution of Π on input (x_1, x_2, x_3).

Definition 1 (Correctness)**.** Let $\mathcal{F} : D_1 \times D_2 \times D_3 \to R_1 \times R_2 \times R_3$ be a function with three inputs and three outputs for some sets $D_1, D_2, D_3, R_1, R_2, R_3$. Let Π be a three-party protocol that computes a functionality $\mathcal{F}$ by the servers S_1, S_2, S_3.

We say that a protocol Π is correct if for all $x_1 \in D_1, x_2 \in D_2, x_3 \in D_3$, the distribution of $\text{out}^{\Pi}(x_1, x_2, x_3)$ is identical to the distribution of $\mathcal{F}(x_1, x_2, x_3)$.

Definition 2 (Semi-honest Security)**.** We say that a protocol Π perfectly realizes a functionality $\mathcal{F}$ in the presence of a static semi-honest corruption if there exists a polynomial time simulator Sim such that for any $i \in \{1, 2, 3\}$ and any $\vec{x} = (x_1, x_2, x_3) \in D_1 \times D_2 \times D_3$, where $|x_1| = |x_2| = |x_3|$:

$$\{(\text{Sim}(x_i, f_i(\vec{x})), f(\vec{x}))\} \equiv \left\{(\text{view}_i^{\Pi}(\vec{x}), \text{out}^{\Pi}(\vec{x}))\right\}.$$

Our proposed protocols are proven to be secure by using the hybrid model, where servers execute a protocol with messages and have access to a trusted party that computes a sub-functionality for them.

2.4 Decision Tree

There are two aspects in a decision tree classification; one is a training phase that creates a classification model and another is an evaluation (inference) phase that predicts the class of input data by using the model. Throughout this paper, we focus on the evaluation phase and we assume that we already have a model in shared form.

To facilitate comprehension, we now introduce the model in the plain setting and not in the secret shared setting. Let $\{\text{attr}_i\}_{i=0}^{m-1}$ be an attribute vector that represents an input to be evaluated, where m is the size of the attribute vector that is determined by a trained model. We assume that the trained model is a complete binary tree with height h, size of an attribute vector m, an index vector $\left\{\text{idx}_j\right\}_{j \in T_{h-1}}$, a threshold vector $\left\{\tau_j\right\}_{j \in T_{h-1}}$, a condition vector $\left\{\text{cond}_j\right\}_{j \in T_{h-1}}$, and a leaf value vector $\{\text{val}_\ell\}_{\ell=0}^{2^h-1}$. A decision tree model $\mathcal{T}$ is defined as follows:

$$\mathcal{T} = \{h, m, \left\{\text{idx}_j\right\}_{j \in T_{h-1}}, \left\{\tau_j\right\}_{j \in T_{h-1}}, \left\{\text{cond}_j\right\}_{j \in T_{h-1}}, \{\text{val}_\ell\}_{\ell=0}^{2^h-1}\}.$$

By abuse of notation, given an attribute vector $\boldsymbol{A}$, $\mathcal{T}(\boldsymbol{A})$ represents the result of the decision tree on $\boldsymbol{A}$.

Fig 2 shows an example of a decision tree with height $h = 3$ and $m = 4$, in which each inner node (d, j) contains an index $\text{idx}_{d,j}$, a threshold $\tau_{d,j}$ and a conditional value $\text{cond}_{d,j}$. Given an attribute vector $\{\text{attr}_0, \text{attr}_1, \text{attr}_2, \text{attr}_3\}$ as input, the evaluation algorithm starts at node $(0, 0)$.

Given the conditional value $\text{cond}_{0,0} = 1$ at node $(0, 0)$, it selects a 'less-than' as a comparison operation and computes a proposition $(\text{attr}_{0,0} < \tau_{0,0})$. If the conditional value is 0 as in node $(2, 0)$, $\text{cond}_{2,0} = 0$, the algorithm selects an 'equality check' as a comparison operation and computes a proposition $(\text{attr}_{2,0} = \tau_{2,0})$.

If a comparison operation outputs 0 (resp. a comparison operation outputs 1), then the algorithm indicates us to move onto the left child node $(1, 0)$ (resp. right child node $(1, 1)$). The above steps are iterated until the algorithm reaches a leaf node. Once it reaches the leaf node, it outputs the corresponding leaf value as a decision (classification result).

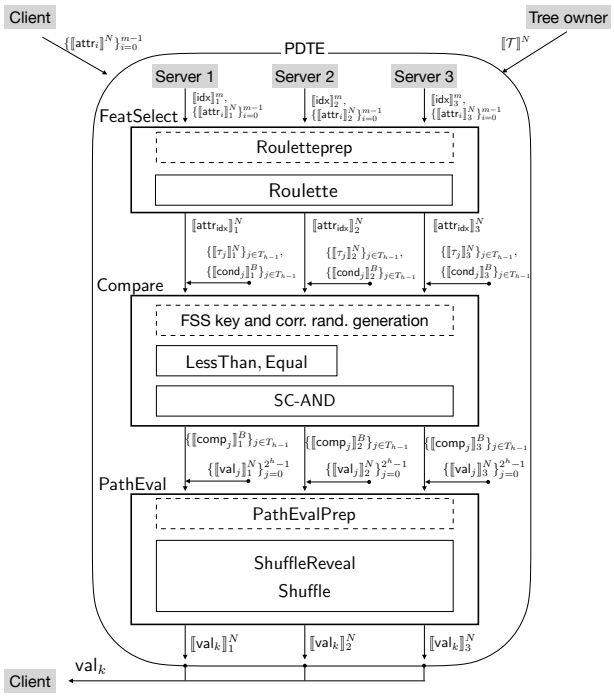


Figure 3: Overview of our PDTE protocol: Protocols in the dotted box can be executed in the offline phase. The LessThan **and** Equal **protocols are executed by any two servers, while other sub-protocols are executed by the three servers.**

3 CONSTRUCTION

We construct a three-party scheme for private decision tree evaluation (PDTE). Our main focus is the outsourced setting: three outsourced computing servers, a service provider (or tree owner) with a trained decision tree model, and clients with an attribute vector as input. The service provider generates shares of the decision tree model and distributes them to three computing servers. Similarly, each client does the same with its own input attribute vector. Since the computing servers only receive values in a shared form, none of the servers can obtain any information about the input data other than the data size. The computing servers cooperatively execute a PDTE protocol and return the shares of a decision result to the client. Finally, the client can reconstruct the evaluation result from the received shares.

Formally, we consider the ideal functionality of PDTE shown in Functionality 1. All inputs and outputs of the scheme are given as shares in the 2-out-of-3 replicated secret sharing scheme. Note that the scheme only allows at most one corruption, *i.e.,* the honest majority setting. PDTE takes as input a shared attribute vector, and a shared decision tree model consisting of an index vector, a threshold value vector and a conditional value vector with size $2^h - 1$ which corresponds to the inner nodes, and a shared leaf value vector with size 2^h. PDTE outputs shares of the result of the

(plaintext) decision tree evaluation procedure on the input private values.

FUNCTIONALITY 1 ($\mathcal{F}_{\mathsf{pdte}}$ – PDTE)**.**

Upon receiving a share of decision tree $[\![\mathcal{T}]\!]^N$ and an attribute vector $([\![\mathsf{attr}_0]\!]^N, [\![\mathsf{attr}_1]\!]^N, \ldots, [\![\mathsf{attr}_{m-1}]\!]^N)$, $\mathcal{F}_{\mathsf{pdte}}$ reconstructs $\mathcal{T}$ and $\boldsymbol{A} = (\mathsf{attr}_0, \mathsf{attr}_1, \ldots, \mathsf{attr}_{m-1})$, computes $\mathsf{val} \leftarrow \mathcal{T}(\boldsymbol{A})$, generates shares $[\![\mathsf{val}]\!]^N$, and sends $[\![\mathsf{val}]\!]^N_i$ to S_i.

Fig. 3 shows the overview of our protocol. Our protocol design follows the framework proposed in [31]. In this framework, the PDTE procedure is divided into three phases; *feature selection* phase, *comparison* phase, and *path evaluation* phase. The feature selection phase securely chooses a designated attribute from the attribute vector for each inner node ($2^h - 1$ nodes in total). In the comparison phase, the resulting attributes are securely compared against thresholds for all the inner nodes at the same time. The path evaluation phase securely computes the certain leaf value as the final result using all the comparison results. We will give a detailed definition of the ideal functionalities of the three phases in Functionality 2-4.

3.1 Feature Selection Phase

In this section, we introduce our feature selection protocol FeatSelect (Protocol 4). Functionality 2 shows the ideal functionality of the feature selection phase. The functionality takes a share of an index and a shared attribute vector as input, and outputs a share of an attribute that corresponds to the index.

FUNCTIONALITY 2 ($\mathcal{F}_{\mathsf{featselect}}$ – Feature Selection)**.**

Upon receiving a share of an index $[\![\mathsf{idx}]\!]^m$ and an attribute vector $([\![\mathsf{attr}_0]\!]^N, [\![\mathsf{attr}_1]\!]^N, \ldots, [\![\mathsf{attr}_{m-1}]\!]^N)$, $\mathcal{F}_{\mathsf{featselect}}$ reconstructs idx and $\boldsymbol{A} = (\mathsf{attr}_0, \mathsf{attr}_1, \ldots, \mathsf{attr}_{m-1})$, selects $\mathsf{attr}_{\mathsf{idx}}$, generates shares $[\![\mathsf{attr}_{\mathsf{idx}}]\!]^N$, and sends $[\![\mathsf{attr}_{\mathsf{idx}}]\!]^N_i$ to S_i.

To achieve round-optimal feature selection, we first propose a three-party variant of the two-party oblivious selection protocol in [5]. This protocol requires preprocessing to generate correlated randomness, which optimizes the round complexity of the online phase. We construct a protocol Roulette (Protocol 3) and its offline protocols PlainRoulette, Rouletteprep (Protocols 1 and 2).

3.1.1 Preprocessing for Roulette. We describe a preprocessing protocol for Roulette, which we call Rouletteprep. It aims to generate correlated randomness that enables online-round-optimal execution of Roulette. More specifically, the protocol functionality takes no input, and outputs a shared vector $[\![\boldsymbol{\alpha}]\!]^N$, a shared shifting value $[\![n]\!]^m$, and random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ only given to S_1 and S_2, respectively. Here $[\![\boldsymbol{\alpha}]\!]^N$ holds a correlation such as $\boldsymbol{\alpha} = -\mathsf{shift}(\boldsymbol{a}, n_3) - \mathsf{shift}(\boldsymbol{b}, n_1)$ for $[\![n]\!]^m_i = (n_i, n_{i+1})$. We write $\mathcal{F}_{\mathsf{rouletteprep}}$ for the functionality.

PlainRoulette The main challenge of Rouletteprep is to securely compute the two shifted values, $[\![\mathsf{shift}(\boldsymbol{a}, n_3)]\!]^N$ and $[\![\mathsf{shift}(\boldsymbol{b}, n_1)]\!]^N$. We propose PlainRoulette protocol for oblivious circular shifting in Protocol 1. The input of PlainRoulette protocol is a vector $\boldsymbol{x}$ in the plain form (not the shared form) from the server S_i and a shifting value s from the server S_j and S_k that is also the plain form, where $i, j, k \in \{1, 2, 3\}$ are all different. The protocol returns the circular-shifted value in the shared form; $[\![\mathsf{shift}(\boldsymbol{x}, s)]\!]^N$.

PlainRoulette requires 2 communication rounds. The first round (Steps 1, 2 and 3) is a variant of the 3-party oblivious transfer protocol proposed in [37]. As in the OT protocol, S_i and S_j first sample a random masking vector $\boldsymbol{w}$ using RndGen. The difference from the OT protocol is that S_j sends the entire circular-shifted vector of $\boldsymbol{w}$ to S_k, rather than a single element in $\boldsymbol{w}$. After that, the servers execute a share conversion in the second round to obtain a replicated share of $\mathsf{shift}(\boldsymbol{x}, s)$.

Protocol 1 Plain Circular Shift (PlainRoulette)

Functionality: $[\![\boldsymbol{y}]\!]^N \leftarrow \mathsf{PlainRoulette}((\boldsymbol{x}, \mathsf{S}_i), (s, \mathsf{S}_j), (s, \mathsf{S}_k))$
Input: A vector $\boldsymbol{x}$ from S_i and a shifting value s from S_j and S_k for mutually different $i, j, k \in \{1, 2, 3\}$
Output: Arithmetic-shared circular-shifted vector $[\![\boldsymbol{y}]\!]^N$, where $\boldsymbol{y} = \mathsf{shift}(\boldsymbol{x}, s)$
1: S_i and S_j generate a random vector $\boldsymbol{w}$ using RndGen
2: S_i computes $\boldsymbol{m} = \boldsymbol{x} - \boldsymbol{w}$ and sends it to S_k
3: S_j samples $\boldsymbol{v}$, locally computes $\boldsymbol{m}' = \mathsf{shift}(\boldsymbol{w}, s) - \boldsymbol{v}$ and sends it to S_k
4: S_k computes $\boldsymbol{v}' = \mathsf{shift}(\boldsymbol{m}, s) + \boldsymbol{m}'$
5: Servers invoke $[\![\boldsymbol{y}]\!]^N \leftarrow \mathsf{ShareConv}^{-1}(\boldsymbol{v}, \boldsymbol{v}')$

Protocol 2 Preprocess of Roulette (Rouletteprep)

Functionality: $((\boldsymbol{a}, [\![\boldsymbol{\alpha}]\!]^N_1, [\![n]\!]^m_1), (\boldsymbol{b}, [\![\boldsymbol{\alpha}]\!]^N_2, [\![n]\!]^m_2), ([\![\boldsymbol{\alpha}]\!]^N_3, [\![n]\!]^m_3)) \leftarrow \mathsf{Rouletteprep}()$
Input: $\perp$
Output: A random vector $\boldsymbol{a}$ only for S_1, a random vector $\boldsymbol{b}$ only for S_2, an arithmetic shared random value $[\![n]\!]^m$, and an arithmetic shared vector $[\![\boldsymbol{\alpha}]\!]^N$ for $\boldsymbol{\alpha} = -\mathsf{shift}(\boldsymbol{a}, n_3) - \mathsf{shift}(\boldsymbol{b}, n_1)$ where $[\![n]\!]^m_i = (n_i, n_{i+1})$
1: Servers collaboratively pick a random value $[\![n]\!]^m \leftarrow \mathsf{RndGen}(m)$
2: S_1 picks a random vector $\boldsymbol{a}$ and S_2 picks a random vector $\boldsymbol{b}$
3: Servers collaboratively execute $[\![\boldsymbol{v}]\!]^N \leftarrow \mathsf{PlainRoulette}((\boldsymbol{a}, \mathsf{S}_1), (n_3, \mathsf{S}_2), (n_3, \mathsf{S}_3))$
4: Servers collaboratively execute $[\![\boldsymbol{w}]\!]^N \leftarrow \mathsf{PlainRoulette}((\boldsymbol{b}, \mathsf{S}_2), (n_1, \mathsf{S}_3), (n_1, \mathsf{S}_1))$
5: $[\![\boldsymbol{\alpha}]\!]^N = -[\![\boldsymbol{v}]\!]^N - [\![\boldsymbol{w}]\!]^N$
6: S_1 returns $(\boldsymbol{a}, [\![\boldsymbol{\alpha}]\!]^N_1, [\![n]\!]^m_1)$, S_2 returns $(\boldsymbol{b}, [\![\boldsymbol{\alpha}]\!]^N_2, [\![n]\!]^m_2)$, and S_3 returns $([\![\boldsymbol{\alpha}]\!]^N_3, [\![n]\!]^m_3)$

Rouletteprep Protocol 2 shows our Rouletteprep protocol. The protocol consists of two parallel calls to PlainRoulette, so it takes 2 communication rounds in total. In the protocol, the shared random value $[\![n]\!]^m$ is obtained by using the random generation algorithm RndGen shown in Sect. 2.2. The random vector $\boldsymbol{a}$ is picked by S_1 locally and the random vector $\boldsymbol{b}$ is picked by S_2 locally as well. $[\![\alpha]\!]^N$ is computed by secure negation done locally and by PlainRoulette. Therefore, as long as PlainRoulette works properly, the value α satisfies the required relation. Such a well-prepared randomness $[\![\alpha]\!]^N$ is later used in Roulette together with another masking value and random vectors.

Theorem 1. The protocol Rouletteprep for $\mathcal{F}_{\mathsf{rouletteprep}}$ is perfectly semi-honest secure in the $\mathcal{F}_{\mathsf{rndgen}}$-hybrid model.

Proof Sketch (Protocol 1 and 2). We first show the correctness and semi-honest security of PlainRoulette. By definition, $[\![\boldsymbol{y}]\!]^N$ is a share

of $v+v'$, which coincides with $\mathsf{shift}(x-w,s)+\mathsf{shift}(w,s) = \mathsf{shift}(x,s)$. The security of the protocol is satisfied because the communication is done with masking by a random vector v and w chosen by the sender servers.

By the definition of Rouletteprep, it holds that $v = \mathsf{shift}(a, n_3)$ and $w = \mathsf{shift}(b, n_1)$, so α is the desired output and correctness is satisfied. Since a, b, n_1 and n_3 are independent randomness known only to the input servers of each PlainRoulette, the parallel two calls to PlainRoulette leak no information.

3.1.2 Roulette Protocol. We next present an online protocol for secure roulette (Protocol 3), which works with the RSS setting. Intuitively, the protocol randomly circular-shifts a given shared vector like a roulette machine, while keeping the given vector and shifting values secret from the servers. More precisely, Roulette takes as input a shared vector $[\![x]\!]^N$, uses a shared shifting value $[\![n]\!]^m$ from the output of Rouletteprep and outputs a share of circular-shifted value $[\![y]\!]^N$ that satisfies $y = \mathsf{shift}(x, n)$.

Here is an overview of the protocol: S_1 (resp. S_2) first locally computes v_1 (resp. v_2) by circular shifting by a random amount and masking with a random vector, then sends it to the other servers. The randomness consumed in computing v_1 and v_2 is provided as the output of Rouletteprep. In the next step, the servers locally circularly shift the received messages so that every message is shifted by n. The final step uses the randomness $[\![\alpha]\!]^N$ to cancel mask vectors a and b, resulting in a share $[\![y]\!]^N$ of the circular-shifted value.

Our Roulette protocol only requires one communication round. Note that Rouletteprep can be done in an offline phase since correlated randomness is independent of input values. We write it as an auxiliary input. In the online protocol, steps 1 and 2 can be done at the same time, and the other steps require no communication.

Protocol 3 Oblivious Circular Shift (Roulette)

Functionality: $([\![y]\!]^N, [\![n]\!]^m) \leftarrow \mathsf{Roulette}([\![x]\!]^N; \mathsf{aux})$
Correlated Randomness: aux which are computed by Rouletteprep
Input: Arithmetic shared vector $[\![x]\!]^N$, where $[\![x]\!]^N_i = (x^{(i)}, x^{(i+1)})$
Output: Arithmetic-shared circular-shifted value $[\![y]\!]^N$ and an arithmetic-shared value $[\![n]\!]^m$, where $y = \mathsf{shift}(x, n)$ and n is from aux

1: S_1 locally computes $v^{(1)} = \mathsf{shift}(x^{(1)} + x^{(2)}, n_1 + n_2) + a$ and sends it to S_2 and S_3, where n_1, n_2, and a are from aux
2: S_2 locally computes $v^{(2)} = \mathsf{shift}(x^{(3)}, n_2 + n_3) + b$ and sends it to S_3 and S_1, where n_2, n_3, and b are from aux
3: S_2 and S_3 locally compute $w^{(3)} = \mathsf{shift}(v^{(1)}, n_3)$
4: S_3 and S_1 locally compute $w^{(1)} = \mathsf{shift}(v^{(2)}, n_1)$
5: Servers set $[\![w]\!]^N_1 = (w^{(1)}, 0)$, $[\![w]\!]^N_2 = (0, w^{(3)})$, and $[\![w]\!]^N_3 = (w^{(3)}, w^{(1)})$
6: Servers locally compute $[\![y]\!]^N = [\![w]\!]^N + [\![\alpha]\!]^N$ and return $([\![y]\!]^N, [\![n]\!]^m)$

Theorem 2. The protocol Roulette for $\mathcal{F}_{\mathsf{roulette}}$ is perfectly semi-honest secure in $\mathcal{F}_{\mathsf{rouletteprep}}$-hybrid model.

Proof Sketch. We first show the correctness of the protocol. By construction,

$$\begin{aligned} w^{(1)} + w^{(3)} &= \mathsf{shift}(v^{(1)}, n_3) + \mathsf{shift}(v^{(2)}, n_1) \\ &= \mathsf{shift}(\mathsf{shift}(x^{(1)} + x^{(2)}, n_1 + n_2) + a, n_3) \\ &\qquad + \mathsf{shift}(\mathsf{shift}(x^{(3)}, n_2 + n_3) + b, n_1) \\ &= \mathsf{shift}(x^{(1)} + x^{(2)}, n) + \mathsf{shift}(a, n_3) \\ &\qquad + \mathsf{shift}(x^{(3)}, n) + \mathsf{shift}(b, n_1) \end{aligned}$$

Thus, we obtain that

$$y = w^{(1)} + w^{(3)} + \alpha = \mathsf{shift}(x^{(1)} + x^{(2)}, n) + \mathsf{shift}(x^{(3)}, n) = \mathsf{shift}(x, n),$$

which shows the correctness of the Roulette.

We next show the semi-honest security. Consider the case S_3 is corrupted. The other cases are similar. Let $(\alpha^{(3)}, \alpha^{(1)}) = [\![\alpha]\!]^N_3$ and $(y^{(3)}, y^{(1)}) = [\![y]\!]^N_3$. Observe that the above correctness proof ensures that the distribution of $[\![y]\!]^N$ is independent of a and b. Since $v^{(1)}$ and $v^{(2)}$ are respectively masked by a and b, the simulator should first pick $v^{(1)}$ and $v^{(2)}$ randomly. Subsequently, the simulator computes from S_3's input and output as $\alpha^{(1)} := y^{(1)} - \mathsf{shift}(v^{(2)}, n_1)$ and $\alpha^{(3)} := y^{(3)} - \mathsf{shift}(v^{(1)}, n_3)$, which gives the same distribution as S_3's view $\mathsf{view}^{\Pi}_3([\![x]\!]^N)$.

3.1.3 Feature Selection Protocol. We now construct our efficient feature selection protocol FeatSelect (Protocol 4). The protocol aims to select a designated attribute at each node using the given index, as defined in Functionality 1.

This protocol is based on a variant of the shuffle-and-reveal technique for circular shifting. First, the servers pick a share of the random value by using RndGen, which is done offline since it is independent of the protocol's input. Second, using such a share of randomness and given a shared attribute vector, the servers cooperatively execute Roulette to output a share of circular-shifted vector $[\![y]\!]^N$. Third, the servers locally compute an addition and reveal the value s. Lastly, the servers output a share by using the revealed value m and securely obtain the circular-shifted vector $[\![y]\!]^N$.

The protocol takes 1 online communication round. This is because Steps 1 and 3 can be done in parallel. The offline computation is only required for Step 1, and needs 2 communication rounds.

Protocol 4 Feature Selection (FeatSelect)

Functionality:

$$[\![\mathsf{attr}_{\mathsf{idx}}]\!]^N \leftarrow \mathsf{FeatSelect}([\![\mathsf{idx}]\!]^m, \{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1}; \mathsf{aux})$$

Correlated Randomness: aux which are computed by Rouletteprep
Input: Arithmetic shared index $[\![\mathsf{idx}]\!]^m$ and arithmetic shared attributes vector $\{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1}$
Output: Arithmetic shared idx-th attribute $[\![\mathsf{attr}_{\mathsf{idx}}]\!]^N$

1: Servers cooperatively compute $([\![y]\!]^N, [\![n]\!]^m) \leftarrow \mathsf{Roulette}(\{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1}; \mathsf{aux})$
2: Servers compute $[\![s]\!]^m = [\![\mathsf{idx}]\!]^m + [\![n]\!]^m$
3: Servers reveal s
4: Each server S_i sets $[\![\mathsf{out}]\!]^N_i = [\![y_s]\!]^N_i$

Theorem 3. The protocol FeatSelect for $\mathcal{F}_{\mathsf{featselect}}$ is perfectly semi-honest secure in the $(\mathcal{F}_{\mathsf{rouletteprep}}, \mathcal{F}_{\mathsf{roulette}})$-hybrid model.

Proof Sketch. By construction, $\boldsymbol{y} = \mathsf{shift}(\{\mathsf{attr}_i\}_{i=0}^{m-1}, n)$, *i.e.,* $\boldsymbol{y}_j = \mathsf{attr}_{(j-n \bmod m)}$ for all $j \in \{0, \ldots, m-1\}$. On the other hand, we have $s = (\mathsf{idx} + n \bmod m)$. Thus,

$$\boldsymbol{y}_s = \boldsymbol{y}_{(\mathsf{idx}+n \bmod m)} = \mathsf{attr}_{(\mathsf{idx}+n-n \bmod m)} = \mathsf{attr}_{\mathsf{idx}},$$

which shows the correctness of FeatSelect.

The semi-honest security of FeatSelect is derived from the shuffle-and-reveal technique. We omit the details since it is similar to the proof for the two-party oblivious selection protocol [5].

3.2 Comparison Phase

In this section, we introduce our protocol Compare (Protocol 6) for the comparison phase. The comparison phase aims to identify if the designated attribute at each node satisfies a certain relation with a threshold, where either less-than or equality is computed. Assigned condition values at each node decide which operation should be used. The comparison is performed for each of the $2^h - 1$ inner nodes. The ideal functionality of the comparison phase is shown in Functionality 3. The functionality takes as input shares of a decision tree model and an input attribute vector and outputs the comparison results in the shared form at all inner nodes. We focus on the *less than* comparison and the *equality check* as comparison operations in this paper and we leave other functionalities for future work.

Functionality 3 ($\mathcal{F}_{\mathsf{compare}}$ – Comparison).
Upon receiving a share of attribute vector $\{[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N\}_{j\in T_{h-1}}$, a threshold vector $\{[\![\tau_j]\!]^N\}_{j\in T_{h-1}}$, a condition vector $\{[\![\mathsf{cond}_j]\!]^N\}_{j\in T_{h-1}}$), $\mathcal{F}_{\mathsf{compare}}$ reconstructs $\left\{\mathsf{attr}_{\mathsf{idx}_j}\right\}_{j\in T_{h-1}}$, $\{\tau_j\}_{j\in T_{h-1}}$ and $\{\mathsf{cond}_j\}_{j\in T_{h-1}}$, computes $C_j \leftarrow (\mathsf{attr}_{\mathsf{idx}_j} < \tau_j)$ if $\mathsf{cond}_j = 1$, computes $C_j \leftarrow (\mathsf{attr}_{\mathsf{idx}_j} = \tau_j)$ if $\mathsf{cond}_j = 0$, generates shares $[\![C_j]\!]^B$, and sends $[\![C_j]\!]^B_i$ to S_i for $j \in T_{h-1}$.

3.2.1 One Round Protocols for LessThan *and* Equal. We use the LessThan and Equal protocols introduced in [9]. These protocols execute 2 + 1 party computations constructed in the scheme 'MPC with preprocessing via function secret sharing (FSS)' [10]. In this scheme, only two servers join in the online computation. In contrast, the third server needs to pre-generate and distribute FSS keys in the offline phase to be consumed in the online computation. An advantage of these protocols is the optimal online communication complexity. In addition, recent studies have achieved an $O(\kappa\ell)$-bit FSS key assuming PSG. Here, κ is a security parameter. For more details, see [9].

3.2.2 Comparison Protocol. Our Compare protocol is described in Protocol 6. We provide an overview of the protocol here. In the first step, the servers perform share conversion and two out of the three servers obtain the corresponding 2-out-of-2 additive shares. Then the protocol computes both LessThan and Equal in parallel. Since these conditional values $\mathsf{cond}_{d,j}$ are given in the shared form, the protocol should be data-oblivious with respect to the shared condition values $\mathsf{cond}_{d,j}$, and must compute both LessThan and Equal. The final step obliviously selects one of the results according to the shared condition value $\mathsf{cond}_{d,j}$. Note that the oblivious selection can be achieved with a secure AND gate.

We also propose a round-efficient protocol SC-AND for the final step, as shown in Protocol 5. The purpose of the SC-AND protocol is to evaluate an AND gate and share conversion at the same time. 1 round as in Protocol 5. The protocol is based on the (3-party) beaver triple technique. Due to the parallelism, the protocol only requires 1 online communication round.

Protocol 5 AND with Share Conversion (SC-AND)

Functionality: $[\![z]\!]^B \leftarrow \mathsf{SC\text{-}AND}(\langle\!\langle x\rangle\!\rangle^B, \langle\!\langle y\rangle\!\rangle^B)$
Input: $(2,2)$-SS boolean value $\langle\!\langle x\rangle\!\rangle^B$ and $\langle\!\langle y\rangle\!\rangle^B$
Correlated Randomness: $(2,3)$-RSS Beaver triple $([\![\alpha]\!]^B, [\![\beta]\!]^B, [\![\gamma]\!]^B)$ s.t. $\alpha\beta = \gamma$
Output: $(2,3)$-RSS boolean value $[\![z]\!]^B$, where $z = xy$
1: Two servers locally obtain the following:
$\langle\!\langle\alpha\rangle\!\rangle^B \leftarrow \mathsf{ShareConv}([\![\alpha]\!]^B)$
$\langle\!\langle\beta\rangle\!\rangle^B \leftarrow \mathsf{ShareConv}([\![\beta]\!]^B)$
2: Two servers reveal the following p and q (also to the third server):
$p \leftarrow \mathsf{Reconst}(\langle\!\langle x\rangle\!\rangle^B \oplus \langle\!\langle\alpha\rangle\!\rangle^B)$
$q \leftarrow \mathsf{Reconst}(\langle\!\langle y\rangle\!\rangle^B \oplus \langle\!\langle\beta\rangle\!\rangle^B)$
3: Three servers locally compute $[\![z]\!]^B \leftarrow pq \oplus p[\![\beta]\!]^B \oplus q[\![\alpha]\!]^B \oplus [\![\gamma]\!]^B$

The Compare protocol requires two online communication rounds. The algorithm involves iterations for $j \in T_{h-1}$, but each iteration step is independent and can run in parallel. For each node, the protocol takes one round for parallel invocations of LessThan and Equal, and another round for SC-AND.

Protocol 6 Comparison (Compare)

Functionality: $\{[\![\mathsf{comp}_j]\!]^B\}_{j\in T_{h-1}} \leftarrow \mathsf{Compare}(\{[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N, [\![\tau_j]\!]^N, [\![\mathsf{cond}_j]\!]^B\}_{j\in T_{h-1}})$
Input: Arithmetic shared attribute vector $\{[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N\}_{j\in T_{h-1}}$, arithmetic shared threshold value $\{[\![\tau_j]\!]^N\}_{j\in T_{h-1}}$, and arithmetic shared conditional flag $\{[\![\mathsf{cond}]\!]^B\}_{j\in T_{h-1}}$
Output: Arithmetic shared comparison result $\{[\![\mathsf{comp}_j]\!]^N\}_{j\in T_{h-1}}$
1: **for** $j \in T_{h-1}$ **do**
2: $\langle\!\langle\mathsf{attr}_{\mathsf{idx}_j}\rangle\!\rangle \leftarrow \mathsf{ShareConv}([\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N)$
3: $\langle\!\langle\tau_j\rangle\!\rangle \leftarrow \mathsf{ShareConv}([\![\tau_j]\!]^N)$
Servers cooperatively compute the following:
4: $\langle\!\langle a\rangle\!\rangle^B \leftarrow \mathsf{LessThan}(\langle\!\langle\mathsf{attr}_{\mathsf{idx}_j}\rangle\!\rangle, \langle\!\langle\tau_j\rangle\!\rangle)$
5: $\langle\!\langle b\rangle\!\rangle^B \leftarrow \mathsf{Equal}(\langle\!\langle\mathsf{attr}_{\mathsf{idx}_j}\rangle\!\rangle, \langle\!\langle\tau_j\rangle\!\rangle)$
6: $[\![\mathsf{comp}_j]\!]^B \leftarrow \mathsf{ShareConv}^{-1}(\langle\!\langle b\rangle\!\rangle^B) \oplus \mathsf{SC\text{-}AND}(\langle\!\langle\mathsf{cond}_j\rangle\!\rangle^B, \langle\!\langle a\rangle\!\rangle^B \oplus \langle\!\langle b\rangle\!\rangle^B)$
7: **end for**

Theorem 4. The protocol Compare for $\mathcal{F}_{\mathsf{compare}}$ is perfectly semi-honest secure in $(\mathcal{F}_{\mathtt{lessthan}}, \mathcal{F}_{\mathtt{equal}})$-hybrid model.

Proof Sketch. By the composability of secure protocols, it is sufficient to show that the SC-AND protocol is secure against semi-honest adversaries. The security proof of the SC-AND protocol is the same as for the standard Beaver-triple-based secure multiplication protocol except for the case where the third server is corrupted. When S_3 is the corrupted third server, the view of the third server consists of the messages from S_1 and S_2 at Step 2. Since ShareConv guarantees that the outputs a and b are uniformly random, these are uniformly random from S_3.

3.3 Path Evaluation Phase

The final phase of PDTE is the path evaluation phase using the comparison results at every inner node as input. We provide a path evaluation protocol PathEval (Protocol 8) with just 2 rounds.

Let PathEval be the path evaluation functionality defined in Functionality 4. PathEval takes, as input, shares of leaf values and shares of comparison results at every inner node, and outputs the corresponding leaf node that is indicated by the comparison results. Here the tracing step chooses one of the leaf values, whose position $path(\{\mathsf{comp}_j\}_{j\in T_{h-1}})$ is given as the unique integer $j \in \mathbb{Z}_{2^h}$ satisfying $j|_{h-d-1} = \mathsf{comp}_{\mathsf{anc}(d,j)}$.

FUNCTIONALITY 4 ($\mathcal{F}_{\mathsf{patheval}}$ – Path Evaluation).

Upon receiving a share of a leaf value vector $\{[\![\mathsf{val}_\ell]\!]^N\}_{\ell=0}^{2^h-1}$ and a comparison result vector $\{[\![\mathsf{comp}_j]\!]^B\}_{j\in T_{h-1}}$, $\mathcal{F}_{\mathsf{patheval}}$ reconstructs $\{\mathsf{val}_\ell\}_{\ell=0}^{2^h-1}$ and $\left\{\mathsf{comp}_j\right\}_{j\in T_{h-1}}$, traces the comparison results from the node $(0,0)$ to a leaf node to obtain the reached leaf value $v := \mathsf{val}_{path(\{\mathsf{comp}_j\}_j)}$, generates shares $[\![v]\!]^N$, and sends $[\![v]\!]_i^N$ to S_i.

To grasp the intuition of the tracing step in PathEval, let us describe how (the plain-text version of) the algorithm will work in the clear. Consider the decision tree in Fig. 2. Suppose that the comparison result at node $(0,0)$ is 1, that is, $\mathsf{comp}_{0,0} = 1$. Then, you move to node $(1,1)$ (a child node on the right) and see $\mathsf{comp}_{1,1}$. When $\mathsf{comp}_{1,1} = 0$, you move to node $(2,2)$ (a child node on the left) and see $\mathsf{comp}_{2,2}$. Let us suppose $\mathsf{comp}_{2,2} = 1$. Then, you reach the leaf node 5, and finally, you take val_5 as output.

3.3.1 Overview of our PathEval Protocol. To compute the tracing step in a few communication rounds, we construct an *oblivious tree shuffle* technique in this section. The purpose of this technique is to shuffle a shared tree obliviously while sustaining the tree structure. The technique lets us detect the right trace path in constant communication rounds by revealing the shuffled node values while leaking no information on the input tree.

The high-level algorithm of our PathEval protocol can be viewed as a tree-structured variant of the shuffle-and-reveal technique [25]. Our proposed tree shuffle aims to directly randomize node values while preserving the tree structure, which can improve communication costs. To clarify this advantage, we highlight the difference from the state-of-the-art protocol [46], which uses the original shuffle-and-reveal technique as follows: The servers first convert the binary tree values to a vector, reorder the obtained vector and the leaf label vector using the same random permutation, and reveal the node label vector to choose the desired leaf value. Note that the vector obtained after the first conversion must require $2^h h$ bits. Our technique avoids such conversion and reduces the data size to $2^h - 1$ bits.

We note that the tree permutation we will define is inspired by the technique proposed by Ma *et al.* [36], but our definition is different from theirs as explained in Sect. 1.3.3. Our tree permutation takes a shared tree as input and outputs a shared shuffled tree with node-wise random flip. In our protocol, to make the round-complexity constant, every random flip is executed independently of the others, rather than using h random bits as in [36].

3.3.2 Tree Permutation. We first define permutations based on node swapping on a complete binary tree, which we call *tree permutation*. Tree permutation is defined for the permutation group derived from C_{h-1}. Each node value specifies whether or not the two sub-trees under the node are swapped. Tree permutation consists of two operations; a permutation of leaf values and a transformation of inner node values.

Definition. Let $\mathfrak{c} \in C_{h-1}$ and $\boldsymbol{v}$ be a vector of length 2^h. The tree permutation of $(\mathfrak{c}, \boldsymbol{v})$ defined by $\mathfrak{r} \in C_{h-1}$ is given as follows:

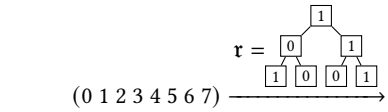
$$(\mathfrak{c}, \boldsymbol{v}) \cdot \mathfrak{r} := (\mathfrak{c} \circ \mathfrak{r}, \boldsymbol{v} \cdot \pi_{\mathfrak{r}}),$$

where $\pi_{\mathfrak{r}} \in S_{2^h}$ and $(\circ) : C_{h-1} \times C_{h-1} \to C_{h-1}$ are defined below.

First, we define the permutation derived from a tree label. For $\mathfrak{r} \in C_{h-1}$, we define a function $\pi_{\mathfrak{r}} : \mathbb{Z}_{2^h} \to \mathbb{Z}_{2^h}$ such that

$$\pi_{\mathfrak{r}}(j)|_{h-d-1} := j|_{h-d-1} \oplus \mathfrak{r}_{\mathsf{anc}(d,j)}$$

for $0 \le d < h$. By definition, $\pi_{\mathfrak{r}}$ is bijective, so it is a permutation, *i.e.*, $\pi_{\mathfrak{r}} \in \mathcal{S}_{2^h}$. For instance, assume that $h = 3$ and $\mathfrak{r}$ is as follows, then values from $\mathbb{Z}_{2^3}$ would be operated by $\pi_{\mathfrak{r}}$ as follows:



Next, we define a binary operation $(\circ)$ on C_{h-1}. For $\mathfrak{r} \in C_{h-1}$, we define the tree extension of $\pi_{\mathfrak{r}}$ as $\pi_{\mathfrak{r}} : T_{h-1} \to T_{h-1}$ such that

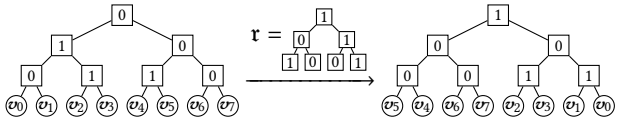
$$\pi_{\mathfrak{r}}(d, j) = (d, \pi_{\mathfrak{r}\restriction_{d-1}}(j)),$$

where $\mathfrak{r} \restriction_{d-1} \in C_{d-1}$ is restricted to be taken from a complete binary tree with depth $d-1$. Note that the extended $\pi_{\mathfrak{r}}$ is also a permutation on T_{h-1}, which consists of permutations on vectors at each depth, and preserves the relations between a parent node and its child nodes. The binary operation $(\circ) : C_{h-1} \times C_{h-1} \to C_{h-1}$ is defined as follows:

$$\mathfrak{c} \circ \mathfrak{r} = (\mathfrak{c} \cdot \pi_{\mathfrak{r}}) \oplus \mathfrak{r},$$

where $(\mathfrak{c} \cdot \pi_{\mathfrak{r}})$ is given as $(\mathfrak{c} \cdot \pi_{\mathfrak{r}})_{d,j} := \mathfrak{c}_{\pi_{\mathfrak{r}}(d,j)}$ and $\oplus$ denotes an element-wise XOR of C_{h-1}.

The following is an example of a tree permutation. When $\mathfrak{c} \in C_2$ and a vector $\boldsymbol{v}$ of length 8 are given as on the left-hand side, $(\mathfrak{c}, \boldsymbol{v}) \cdot \pi_{\mathfrak{r}}$ corresponds to the label on the right tree.



Another Formulation. To assist in finding the intuition of tree permutations, we also provide an operational definition of tree permutations. Given $(\mathfrak{c}, \boldsymbol{v})$ and $\mathfrak{r} \in C_{h-1}$ as input, the calculation of $(\mathfrak{c} \circ \mathfrak{r}, \boldsymbol{v} \cdot \pi_{\mathfrak{r}})$ can proceed as follows:

1: Set $(\mathfrak{d}, \boldsymbol{z}) := (\mathfrak{c}, \boldsymbol{v})$
2: **for** $d = 0, \ldots, h-1$ **do**
3: **for** $j = 0, \ldots, 2^d - 1$ **do in parallel**
4: If $\mathfrak{r}_{d,j} = 1$, update $(\mathfrak{d}, \boldsymbol{z})$ by swapping the two sub-trees under the node (d, j)
5: **end for**
6: **end for**
7: Output $(\mathfrak{d} \oplus \mathfrak{r}, \boldsymbol{z})$

Under the operational definition, the above example of tree permutation is calculated as in Fig. 4. This formulation shows that a tree permutation can be viewed as a composite of node swapping at every inner node.

Properties. We remark on important properties of tree permutations:

- For $\mathfrak{c}, \mathfrak{r} \in C_{h-1}$, it holds that $\pi_{\mathfrak{c}\circ\mathfrak{r}} = \pi_{\mathfrak{c}} \cdot \pi_{\mathfrak{r}}$ for the permutation composition operation '$\cdot$', *i.e.,* $\pi_{\mathfrak{c}\circ\mathfrak{r}}(d, j) = \pi_{\mathfrak{c}}(\pi_{\mathfrak{r}}(d, j))$ for all $(d, j) \in T_h$.
- $(C_{h-1}, \circ)$ forms a finite group. The identity is the all-zero label $\mathbf{0}$, and the inverse of $\mathfrak{c}$ is given as $\mathfrak{c} \cdot \pi_{\mathfrak{c}}^{-1}$.
- A tree permutation preserves the output of the tracing step of PathEval. That is, for any $\mathfrak{r} \in C_{h-1}$, $\boldsymbol{v}_{path(\mathfrak{c})}$ and $(\boldsymbol{v} \cdot \pi_{\mathfrak{r}})_{path(\mathfrak{c}\circ\mathfrak{r})}$ are identical. This is shown as follows:

$$\begin{aligned}\pi_{\mathfrak{r}}(j)|_{h-d-1} &= j|_{h-d-1} \oplus \mathfrak{r}_{\mathsf{anc}(d,j)} \\ &= (\mathfrak{c} \circ \mathfrak{r})_{\mathsf{anc}(d,j)} \oplus \mathfrak{r}_{\mathsf{anc}(d,j)} \\ &= (\mathfrak{c} \cdot \pi_{\mathfrak{r}})_{\mathsf{anc}(d,j)} \\ &= \mathfrak{c}_{\mathsf{anc}(d,\pi_{\mathfrak{r}}(j))}\end{aligned}$$

This means that $t = \pi_{\mathfrak{r}}(j)$, thus $\boldsymbol{v}_t = \boldsymbol{v}_{\pi_{\mathfrak{r}}(j)} = (\boldsymbol{v} \cdot \pi_{\mathfrak{r}})_j$.

Oblivious Tree Shuffle. Now we explain our *oblivious tree shuffle* algorithm. It is defined as a secure computation of tree permutation using random inner node values: Let $\mathfrak{c} \in C_{h-1}$ and $\boldsymbol{v} \in \mathbb{Z}_N^{2^h}$, let $\{[\![\mathfrak{c}]\!]^B, [\![\boldsymbol{v}]\!]^N\}$ be input tree values, and let $\mathfrak{r} \in C_{h-1}$ be a random inner node value unknown to any party. Then, the procedure randomizes $\{[\![\mathfrak{c}]\!]^B, [\![\boldsymbol{v}]\!]^N\}$ with $\mathfrak{r}$, by computing $\{[\![\mathfrak{c} \circ \mathfrak{r}]\!]^B, [\![\boldsymbol{v} \cdot \pi_{\mathfrak{r}}]\!]^N\}$.

Note that this randomization technique satisfies two important properties as follows: (1) If $\mathfrak{r} \in C_{h-1}$ is chosen uniformly at random, then $\mathfrak{c} \circ \mathfrak{r}$ is also uniformly distributed, regardless of what $\mathfrak{c}$ is. This follows from the fact that $(C_{h-1}, \circ)$ is a finite group. (2) The path evaluation algorithm returns the same result for $(\mathfrak{c}, \boldsymbol{v})$ and $(\mathfrak{c} \circ \mathfrak{r}, \boldsymbol{v} \cdot \pi_{\mathfrak{r}})$, as shown above. These properties ensure that oblivious tree shuffling can be used with the shuffle-and-reveal technique.

3.3.3 Path Evaluation Protocol. We now describe a protocol for the path evaluation phase. The protocol is intended to perform an oblivious tree shuffle, using the (ordinary) oblivious shuffle protocol. Note that $\mathfrak{c} \circ \mathfrak{r} = (\mathfrak{c} \oplus \mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1}) \cdot \pi_{\mathfrak{r}}$. Here, the application of $\pi_{\mathfrak{r}}$ consists of parallel vector permutations at each depth, so we can securely compute it using an oblivious shuffle at each depth. Accordingly, the task of the offline phase is to generate $\langle \pi_{\mathfrak{r}} \rangle$ for some random $\mathfrak{r} \in C_{h-1}$ as an auxiliary input of Shuffle and ShuffleReveal, and the input-independent shared-value $[\![\mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1}]\!]^B$.

Offline Phase. We construct an offline phase protocol PathEvalPrep to generate correlated randomness as shown in Protocol 7. The protocol takes 2 rounds. First, $\langle \pi_{\mathfrak{r}} \rangle$ is generated locally by using the random generation algorithm RndGen. This step is well-defined because $(\mathfrak{r} \mapsto \pi_{\mathfrak{r}})$ is injective. Then, S_1 and S_2 pick a random $\mathfrak{s}$, and S_2 uses it for masking the value of $\mathfrak{m}$. After all, $\mathfrak{s}$ is canceled when considering $\mathfrak{u}_1 + \mathfrak{u}_3$. $(\mathfrak{u}_1, \mathfrak{u}_3)$ is an additive sharing of $\mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1}$, since $\mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1} = \mathfrak{r}_1 \cdot \pi_{\mathfrak{r}_1}^{-1} \oplus \mathfrak{r}_2 \cdot (\pi_{\mathfrak{r}_1} \cdot \pi_{\mathfrak{r}_2})^{-1} \oplus \mathfrak{r}_3 \cdot (\pi_{\mathfrak{r}_1} \cdot \pi_{\mathfrak{r}_2} \cdot \pi_{\mathfrak{r}_3})^{-1}$ for $\mathfrak{r} = \mathfrak{r}_1 \circ \mathfrak{r}_2 \circ \mathfrak{r}_3$. Thus, $\mathsf{ShareConv}^{-1}$ gives a desired output $[\![\mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1}]\!]^B$.

Online Phase. We define a protocol for the online phase as shown in Protocol 8. The protocol takes $\{[\![\mathfrak{c}]\!]^B, [\![\boldsymbol{v}]\!]^N\}$ as input, and outputs $[\![\boldsymbol{v}_t]\!]^N$ where $t = path(\mathfrak{c})$. Steps 1-4 perform an oblivious tree shuffle to reveal inner node labels. As a result, each S_i holds a new decision tree $(\mathfrak{e}, [\![\boldsymbol{w}]\!]_i^N)$. In step 5, S_i locally computes the (plain-text) trace step for input $(\mathfrak{e}, [\![\boldsymbol{w}]\!]_i^N)$ to get the resulting position $t' = path(\mathfrak{e})$, and obtains the desired output $[\![\boldsymbol{w}_{t'}]\!]^N$. Note that in ShuffleReveal, shuffling $[\![\mathfrak{d}]\!]^B$ with $\langle \pi \rangle$ requires a permutation of nodes at each depth. Since Shuffle and ShuffleReveal can run in parallel, the protocol only requires two online communication rounds when we use a known 2-round private shuffling protocols (OptShuffle and ShuffleReveal in [17]) for each depth in parallel.

Protocol 7 Preprocess of Path Evaluation (PathEvalPrep)

Functionality: $(\langle \pi \rangle, [\![\mathfrak{b}]\!]^B) \leftarrow \mathsf{PathEvalPrep}(\perp)$
Input: $\perp$.
Output: $(\langle \pi \rangle_i, [\![\mathfrak{b}]\!]_i^B)$ to S_i, where $\pi = \pi_{\mathfrak{r}_1} \cdot \pi_{\mathfrak{r}_2} \cdot \pi_{\mathfrak{r}_3}$ for some $\mathfrak{r}_i \in C_{h-1}$, and $\mathfrak{b} = \mathfrak{r} \cdot \pi_{\mathfrak{r}}^{-1}$ for $\mathfrak{r} = \mathfrak{r}_1 \circ \mathfrak{r}_2 \circ \mathfrak{r}_3$

1: S_{i-1} and S_i sample $\mathfrak{r}_i \xleftarrow{\$} C_{h-1}$ using RndGen, and locally compute $\pi_{\mathfrak{r}_i}$ from $\mathfrak{r}_i$
2: S_i sets $\langle \pi \rangle_i = (\pi_{\mathfrak{r}_i}, \pi_{\mathfrak{r}_{i+1}})$
3: S_1 and S_2 sample $\mathfrak{s} \xleftarrow{\$} C_{h-1}$ using RndGen
4: S_1 locally computes $\mathfrak{u}_1 = \mathfrak{s} \cdot \pi_{\mathfrak{r}_1}^{-1} \oplus \mathfrak{r}_1 \cdot \pi_{\mathfrak{r}_1}^{-1} \oplus \mathfrak{r}_2 \cdot (\pi_{\mathfrak{r}_1} \cdot \pi_{\mathfrak{r}_2})^{-1}$
5: S_2 locally computes $\mathfrak{m} = \mathfrak{r}_3 \cdot (\pi_{\mathfrak{r}_2} \cdot \pi_{\mathfrak{r}_3})^{-1} \oplus \mathfrak{s}$ and sends it to S_3
6: S_3 locally computes $\mathfrak{u}_3 = \mathfrak{m} \cdot \pi_{\mathfrak{r}_1}^{-1}$
7: Servers invoke $[\![\mathfrak{b}]\!]^B \leftarrow \mathsf{ShareConv}^{-1}(\mathfrak{u}_1, \mathfrak{u}_3)$

Protocol 8 Path Evaluation (PathEval)

Functionality: $[\![\boldsymbol{v}_j]\!]^N \leftarrow \mathsf{PathEval}([\![\mathfrak{c}]\!]^B, [\![\boldsymbol{v}]\!]^N)$
Input: $[\![\mathfrak{c}]\!]^B, [\![\boldsymbol{v}]\!]^N$, where $\mathfrak{c} \in C_{h-1}$, $\boldsymbol{v} \in \mathbb{Z}_N^{2^h}$
Output: $[\![\boldsymbol{v}_j]\!]^N$

1: $(\langle \pi \rangle, [\![\mathfrak{b}]\!]^B) \leftarrow \mathsf{PathEvalPrep}(\perp)$
2: $[\![\mathfrak{d}]\!]^B = [\![\mathfrak{c}]\!]^B \oplus [\![\mathfrak{b}]\!]^B$
3: $\mathfrak{e} \leftarrow \mathsf{ShuffleReveal}([\![\mathfrak{d}]\!]^B; \langle \pi \rangle)$
4: $[\![\boldsymbol{w}]\!]^N \leftarrow \mathsf{Shuffle}([\![\boldsymbol{v}]\!]^N; \langle \pi \rangle)$
5: Each server locally computes $t' := path(\mathfrak{e})$, and outputs $[\![\boldsymbol{w}_{t'}]\!]^N$.

Theorem 5. The protocol PathEval for $\mathcal{F}_{\mathsf{patheval}}$ is perfectly semi-honest secure in the $(\mathcal{F}_{\mathsf{Shuffle}}, \mathcal{F}_{\mathsf{ShuffleReveal}}, \mathcal{F}_{\mathsf{PathEvalPrep}})$-hybrid model.

Proof Sketch. The correctness of the protocol directly comes from property (2) of oblivious tree shuffle. The security proof is similar to the proof for protocols using the shuffle-and-reveal technique (for example, [25]). Remark that property (1) of oblivious tree shuffle guarantees that the revealed values $\mathfrak{e}$ do not leak any information about the input values.

3.4 Private Decision Tree Evaluation

We now propose our PDTE protocol in Protocol 9. Remark that the ideal functionality PDTE is given in Functionality 1. The protocol performs FeatSelect, Compare, and PathEval sequentially, so the total number of online communication rounds is 5.

Theorem 6. The protocol PDTE for $\mathcal{F}_{\mathsf{pdte}}$ is perfectly semi-honest secure in the $(\mathcal{F}_{\mathsf{featselect}}, \mathcal{F}_{\mathsf{compare}}, \mathcal{F}_{\mathsf{patheval}})$-hybrid model.

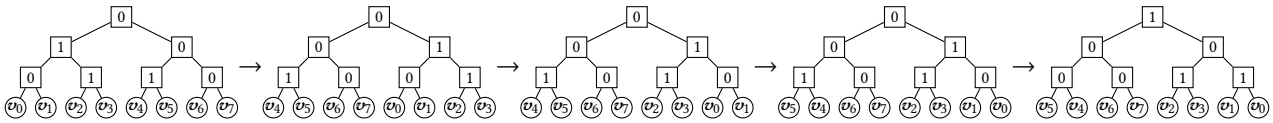


Figure 4: A Calculation Example of Tree Permutation. Each step indicates swapping by $\mathfrak{r}_{0,0} = 1$, swapping by $\mathfrak{r}_{1,1} = 1$, swapping by $\mathfrak{r}_{2,0} = 1$ and $\mathfrak{r}_{2,3} = 1$, and taking the element-wise XOR with $\mathfrak{r}$, respectively.

Protocol 9 Private Decision Tree Evaluation (PDTE)

Functionality: $[\![\mathsf{val}]\!]^N \leftarrow \mathsf{PDTE}(\{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1}, \{[\![\mathsf{idx}_j]\!]^m\}_{j\in T_{h-1}}, \{[\![\tau_j]\!]^N\}_{j\in T_{h-1}}, \{[\![\mathsf{cond}_j]\!]^B\}_{j\in T_{h-1}}, \{[\![\mathsf{val}_\ell]\!]^N\}_{\ell=0}^{2^h-1})$

Input: Arithmetic shared attribute vector $\{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1}$, index vector $\{[\![\mathsf{idx}_j]\!]^m\}_{j\in T_{h-1}}$, threshold value vector $\{[\![\tau_j]\!]^N\}_{j\in T_{h-1}}$, condition vector $\{[\![\mathsf{cond}_j]\!]^B\}_{j\in T_{h-1}}$, leaf value vector $\{[\![\mathsf{val}_\ell]\!]^N\}_{\ell=0}^{2^h-1}$)

Output: Arithmetic shared leaf value $[\![\mathsf{val}]\!]^N$

1: **for** $j \in T_{h-1}$ **do**
2: Servers cooperatively compute and store the share of attribute $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N \leftarrow \mathsf{FeatSelect}([\![\mathsf{idx}_j]\!]^m, \{[\![\mathsf{attr}_i]\!]^N\}_{i=0}^{m-1})$
3: **end for**
4: $\{[\![\mathsf{comp}_j]\!]^B\}_{j\in T_{h-1}} \leftarrow \mathsf{Compare}(\{[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]^N\}_{j\in T_{h-1}}, \{[\![\tau_j]\!]^N\}_{j\in T_{h-1}}, \{[\![\mathsf{cond}_j]\!]^B\}_{j\in T_{h-1}})$
5: $[\![\mathsf{val}_k]\!]^N \leftarrow \mathsf{PathEval}(\{[\![\mathsf{comp}_j]\!]^B\}_{j\in T_{h-1}}, \{[\![\mathsf{val}_j]\!]^N\}_{j=0}^{2^h-1})$
6: **return** $[\![\mathsf{val}_k]\!]^N$

4 EXPERIMENT

In this section, we provide detailed evaluation results of our PDTE protocol.

4.1 Setting

As mentioned in Sect. 2.4, we assume that the servers obtain a shared decision tree as a (shared) trained model. First, we show our target dataset and then we show how to prepare such trained models from the available dataset.

4.1.1 Dataset and model. We select datasets from the UCI machine learning repository [22], which are Wine, Linnerud, Breast cancer, Digits and Diabetes. All of these datasets are also used in [36] to evaluate the performance of their protocols.

The decision tree for each dataset is trained using the scikit-learn [42] toolkit, which enables to set up of the maximum possible height of the desired final evaluation tree. Dummy nodes are then added to this decision tree as in [31] and used for MPC evaluation. Note that comparison with an experiment with sparse trees in [36] and our experiment with a complete tree by adding dummy nodes imply that computation and storage cost in a complete tree is not a big deal, while the bandwidth of the network will affect the total execution time.

From the dataset above, we trained seven different decision tree models ranging from a depth of 5 to 18. See Table 4 and the Appendix C for more details.

4.1.2 Implementation Setup. We implemented the online phase of FeatSelect, Compare, and PathEval, in which three parties execute the protocols with respective secret sharings as input and interact with each other. For the offline phase, we evaluate the protocol construction by estimating the actual network communication cost theoretically.

Table 4: Tree Parameters in Our Experiments

Decision Tree	#Attributes	Tree Depth	#Nodes	Padding%
WINE	6	5	11	64.52%
LINNERUD	3	6	19	69.84%
BREAST	13	7	21	83.46%
DIGITS-10	48	10	138	86.51%
DIGITS-12	48	12	159	96.12%
DIGITS-15	48	15	167	99.49%
DIABETES-18	10	18	369	99.86%

For a prototype implementation [1], we implemented our PDTE protocol using Python 3.10.4. We use the *sycret* Python wrapper [45] to implement the $\mathcal{F}_{\mathsf{Equal}}$ function. This wrapper efficiently handles both the *Distributed Point Function* (DPF) and *Distributed Comparison Function* (DCF) using Rust. Based on DPF, we can realize $\mathcal{F}_{\mathsf{Equal}}$ within the pre-processing model. However, the *sycret* wrapper does not offer an implementation for $\mathcal{F}_{\mathsf{LessThan}}$. To address this, we follow the approach for creating an interval containment gate, as described in Fig.3 of [9], to implement $\mathcal{F}_{\mathsf{LessThan}}$.

To test the performance of our protocols in a genuine heterogeneous network setting, we used a test bed composed of three droplets provided by DigitalOcean, referred to as S_1, S_2, and S_3. Each of these droplets shares identical configurations, possessing a relatively low specification of 8GB memory and 4-core CPUs, and operating on Ubuntu 22.04 LTS. The geographical locations of the droplets vary, with S_1 situated in San Francisco, S_2 in Singapore, and S_3 in Frankfurt. The RTT latency and bandwidth measured between these servers were, on average, 180ms with 62Mbps between S_1 and S_2, 160ms with 135Mbps for S_1 and S_3, and 170ms with 115Mbps for S_2 and S_3.

To prepare the RSS shares of the trained decision tree models, we first execute the training via non-MPC machine learning algorithms. Then we expand the decision tree to a full binary tree by padding multiple dummy nodes and obtain RSS shares of the tree.

To test the real performance of our protocol, we run our implementations on the above WAN test bed. Here, we execute our protocol with the RSS shares of the test vectors from the UCI dataset and the extended complete binary decision tree. This process allows us to learn the real-world performance of our protocol such as actual computation time and the amount of communication volume.

Both run-time cost and communication volume are measured in the online phase, while we measured only communication volume in the offline phase. More detailed settings are described in Appendix C.

Table 5: Round and Communication Cost of PDTE Protocols

		FeatSelect		Compare		PathEval	
		Online	Offline	Online	Offline	Online	Offline
Tsuchida *et al.* [46]	Round	11	2	10	1	4	0
	Comm.	$(k'(15k+(3k-3)\log p+4)+6mk'+3mk+3k)\cdot(2^h-1)$ ▹50.1 MB	$(6k^2+mk+2m-k)k'\cdot(2^h-1)$ ▹190.3 MB	$(45k+(9k-9)\log p+22)\cdot(2^h-1)$ ▹11.9 MB	$18k(k-1)\cdot(2^h-1)$ ▹73.1 MB	$(6k+6h+3)\cdot 2^h$ ▹1.1 MB	0
Ji *et al.* [28]	Round	1	1	1	1	2	0
	Comm.	$12m\cdot(2^h-1)$ ▹2.3 MB	$6((m+1)\lambda+2m+k)\cdot(2^h-1)$ ▹157.2 MB	$6k\cdot(2^h-1)$ ▹6.2 MB	$2(k(2\lambda+3)+2\lambda)\cdot(2^h-1)$ ▹69.9 MB	$(2\lambda+2k)\cdot 2^h+2\cdot((h+1)\lambda+2h+k)$ ▹1.3 MB	0
This work	Round	1	2	2	1	2	2
	Comm.	$(4mk+3k')\cdot(2^h-1)$ ▹25.2 MB	$10mk\cdot(2^h-1)$ ▹62.9 MB	$(4k+13)\cdot(2^h-1)$ ▹0.5 MB	$(2(\lambda+7)(k+1)-5)\cdot(2^h-1)$ ▹36.4 MB	$5k\cdot 2^h+4\cdot(2^h-1)$ ▹0.6 MB	$4\cdot(2^h-1)$ ▹16.3 KB

The estimated communication cost for DIGITS-15 is written after the "▹"symbol by calculating with $k = 32, p = 37, m = 48, h = 15, \lambda = 128$. Here, m is the size of feature vector, $k = \lceil \log N \rceil$, p is the smallest prime number greater than k, $k' = \lceil \log m \rceil$, and λ is security parameter for FSS. Note that this estimation is naively calculated using the equations of approximated communication cost in this table so that the numbers only give a rough estimation.

Table 6: Total Bytes Sent during Our PDTE Protocol (KB/all parties)

	FeatSelect		Compare		PathEval		Total	
	Online	Offline	Online	Offline	Online	Offline	Online	Offline
WINE	4.96	7.44	3.32	34.50	1.44	0.015	9.71	41.9
LINNERUD	6.33	7.56	6.53	70.12	2.59	0.031	15.46	77.71
BREAST	37.51	66.04	12.97	141.36	4.90	0.063	55.39	207.47
DIGITS-10	1009.07	1964.16	103.10	1138.72	37.28	0.51	1149.45	3103.39
DIGITS-12	4039.07	7862.40	412.09	4558.24	148.28	2.04	4599.45	12422.69
DIGITS-15	32319.07	62912.64	3296.09	36473.76	1184.27	16.38	36799.44	99402.79
DIABETES-18	61951.83	104857.20	26368.09	291797.92	9472.24	131.07	97792.16	396786.19

Table 7: Running Time of Our PDTE Protocol [s]

	FeatSelect	Compare	PathEval	Total
WINE	0.13	0.27	0.44	0.83
LINNERUD	0.12	0.27	0.45	0.85
BREAST	0.13	0.27	0.46	0.86
DIGITS-10	0.75	0.48	0.41	1.64
DIGITS-12	1.12	0.75	0.77	2.64
DIGITS-15	3.22	3.71	2.79	9.73
DIABETES-18	10.94	31.96	38.04	80.94

4.2 Evaluation

We first estimate the bits sent among the servers and the communication rounds required for our PDTE protocol by definition. We counted them in an online/offline paradigm to focus on the actual execution time after obtaining the PDTE input, and we distinguished offline pre-computations from the online execution. We compare our protocol with the state-of-the-art PDTE protocol [46] theoretically. Since [46] does not distinguish between online and offline rounds, we re-counted rounds to separate them into online/offline in Table 5 and have a fair comparison.

The evaluation results in Table 5 show that our protocol requires almost the same offline rounds as [46] but requires much fewer online rounds. More precisely, our PDTE protocol only requires 5 online-rounds in total while [46] requires 25 online rounds.

Next, we compare our work with the most recent work by Ji *et al.* [28] that claimed it constructed a 4-round PDTE protocol. Although the functionality of PDTE protocol in [28] is simplified as pointed out in Appendix B, their PDTE protocol requires more communication bits than that of our proposed protocol, *i.e.,* [28] requires approximately 235 MB to execute PDTE on the DIGITS-15 decision tree model, while ours requires approximately 135 MB of communication in total. See the estimation of Ji *et al.* and ours in Table 5. Note that, in Table 5, the estimation is naively calculated by using the approximated equations of Tsuchida *et al.* [46] and Ji *et al.* [28], which only give a rough estimation, and by our exact equation, which gives an estimated minimum cost.

In Table 6, we record the actual online communication cost for each of our subprotocols, while we list the estimated offline communication cost. It shows that the communication costs rise exponentially with the depth of the tree. More specifically, across all phases of our protocols, the communication cost demonstrates a linear relationship with 2^h.

In Table 7, we measure the running time of each online subprotocol by calculating the elapsed time from the beginning to the end of the respective subprotocol. We then calculate the average of these measurements across all three droplets. Note that the reported running time includes both the time for sending/receiving message packets on network and the processing of the received messages. To further reduce the execution time, we parallelized data processing during the online FeatSelect and Compare phases. We segmented the collected message from all tree nodes into smaller batches for processing. Each batch, comprising messages from 1400 nodes, was processed in individual threads. Additionally, our design ensures that the sending, receiving, and processing of these messages occur asynchronously. As indicated by Table 7, the online phase executes swiftly for a decision tree of depth less than or equal to 12. However, for trees of depth greater than 15, the vast communication volume and associated computation cost become constraining, even though our protocol maintains a constant round. Factors such as bandwidth constraints, network delays, and processing extensive messages form bottlenecks in execution time. It's worth mentioning that our evaluations ran using Python on modest servers (8GB Memory, 4-core CPUs). Implementing our protocol in a high-performance programming language and deploying it on superior servers will surely expedite the evaluation of deeper decision trees significantly.

The outsourced PDTE protocol by Ma *et al.* [36] reported in Fig. 15 of Ji *et al.* [28] has a running time of approximately 3 seconds for BREAST and 10 seconds for DIGITS-15 in the WAN setting with 80 ms of network latency, while our construction takes 0.86 seconds and 9.73 seconds, respectively, in the WAN with two times larger network latency (160-180 ms). This fact convinces us to claim that our protocol works well even in networks with larger delays because of its small constant round complexity.

ACKNOWLEDGMENTS

This work was supported by the Digital Research Centre Denmark (DIREC) under the Privacy and Machine Learning project, DIGIT Aarhus University Centre for Digitalisation, Big Data and Data Analytics, JSPS KAKENHI Grant Number JP21H05052, JST CREST Grant Number JPMJCR22M1, and the GFF project "Decent-IoT: Decentralised Private and Secure Internet of Things". We thank the anonymous reviewers of PoPETs 2024 for their valuable comments.

REFERENCES

[1] Anonymous. 2023. PDTE Implementation of this work. https://anonymous.4open.science/r/PDTE-8B2A

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, Vienna, Austria, 805–817. https://doi.org/10.1145/2976749.2978331

[3] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 610–629. https://doi.org/10.1145/3460120.3484560

[4] Nuttapong Attrapadung, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Takahiro Matsuda, Ibuki Mishina, Hiraku Morita, and Jacob C. N. Schuldt. 2022. Adam in Private: Secure and Fast Training of Deep Neural Networks with Adaptive Moment Estimation. *PoPETs* 2022, 4 (Oct. 2022), 746–767. https://doi.org/10.56553/popets-2022-0131

[5] Nuttapong Attrapadung, Goichiro Hanaoka, Takahiro Matsuda, Hiraku Morita, Kazuma Ohara, Jacob C. N. Schuldt, Tadanori Teruya, and Kazunari Tozawa. 2021. Oblivious Linear Group Actions and Applications. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 630–650. https://doi.org/10.1145/3460120.3484584

[6] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2009. Secure Evaluation of Private Linear Branching Programs with Medical Applications. In *ESORICS 2009 (LNCS, Vol. 5789)*, Michael Backes and Peng Ning (Eds.). Springer, Heidelberg, Germany, Saint-Malo, France, 424–439. https://doi.org/10.1007/978-3-642-04444-1_26

[7] Marina Blanton, Ah Reum Kang, and Chen Yuan. 2020. Improved Building Blocks for Secure Multi-party Computation Based on Secret Sharing with Honest Majority. In *ACNS 20, Part I (LNCS, Vol. 12146)*, Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi (Eds.). Springer, Heidelberg, Germany, Rome, Italy, 377–397. https://doi.org/10.1007/978-3-030-57808-4_19

[8] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. 2012. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.* 11, 6 (2012), 403–418. https://doi.org/10.1007/s10207-012-0177-2

[9] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In *EUROCRYPT 2021, Part II (LNCS, Vol. 12697)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Heidelberg, Germany, Zagreb, Croatia, 871–900. https://doi.org/10.1007/978-3-030-77886-6_30

[10] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure Computation with Preprocessing via Function Secret Sharing. In *TCC 2019, Part I (LNCS, Vol. 11891)*, Dennis Hofheinz and Alon Rosen (Eds.). Springer, Heidelberg, Germany, Nuremberg, Germany, 341–371. https://doi.org/10.1007/978-3-030-36030-6_14

[11] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. 2007. Privacy-preserving remote diagnostics. In *ACM CCS 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM Press, Alexandria, Virginia, USA, 498–507. https://doi.org/10.1145/1315245.1315307

[12] Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. 2020. Efficient 3-Party Distributed ORAM. In *SCN 20 (LNCS, Vol. 12238)*, Clemente Galdi and Vladimir Kolesnikov (Eds.). Springer, Heidelberg, Germany, Amalfi, Italy, 215–232. https://doi.org/10.1007/978-3-030-57990-6_11

[13] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PoPETs* 2020, 2 (April 2020), 459–480. https://doi.org/10.2478/popets-2020-0036

[14] Octavian Catrina and Sebastiaan de Hoogh. 2010. Secure Multiparty Linear Programming Using Fixed-Point Arithmetic. In *ESORICS 2010 (LNCS, Vol. 6345)*, Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou (Eds.). Springer, Heidelberg, Germany, Athens, Greece, 134–150. https://doi.org/10.1007/978-3-642-15497-3_9

[15] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. 2020. Secret-Shared Shuffle. In *ASIACRYPT 2020, Part III (LNCS, Vol. 12493)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, Heidelberg, Germany, Daejeon, South Korea, 342–372. https://doi.org/10.1007/978-3-030-64840-4_12

[16] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS 2020*. The Internet Society, San Diego, CA, USA.

[17] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Naoto Kiribuchi, and Benny Pinkas. 2019. An Efficient Secure Three-Party Sorting Protocol with an Honest Majority. Cryptology ePrint Archive, Report 2019/695. https://eprint.iacr.org/2019/695.

[18] Geoffroy Couteau. 2018. New Protocols for Secure Equality Test and Comparison. In *ACNS 18 (LNCS, Vol. 10892)*, Bart Preneel and Frederik Vercauteren (Eds.). Springer, Heidelberg, Germany, Leuven, Belgium, 303–320. https://doi.org/10.1007/978-3-319-93387-0_16

[19] Ronald Cramer, Ivan Damgård, and Yuval Ishai. 2005. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC 2005 (LNCS, Vol. 3378)*, Joe Kilian (Ed.). Springer, Heidelberg, Germany, Cambridge, MA, USA, 342–362. https://doi.org/10.1007/978-3-540-30576-7_19

[20] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. 2006. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In *TCC 2006 (LNCS, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, Heidelberg, Germany, New York, NY, USA, 285–304. https://doi.org/10.1007/11681878_15

[21] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey Truex. 2019. Efficient and Private Scoring of Decision Trees, Support Vector Machines and Logistic Regression Models Based on Pre-Computation. *IEEE Transactions on Dependable and Secure Computing* 16, 2 (2019), 217–230. https://doi.org/10.1109/TDSC.2017.2679189

[22] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[23] Sky Faber, Stanislaw Jarecki, Sotirios Kentros, and Boyang Wei. 2015. Three-Party ORAM for Secure Computation. In *ASIACRYPT 2015, Part I (LNCS, Vol. 9452)*, Tetsu Iwata and Jung Hee Cheon (Eds.). Springer, Heidelberg, Germany, Auckland, New Zealand, 360–385. https://doi.org/10.1007/978-3-662-48797-6_16

[24] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, Bethesda, MD, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[25] Koki Hamada, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. 2014. Oblivious Radix Sort: An Efficient Sorting Algorithm for Practical Secure Multi-party Computation. Cryptology ePrint Archive, Report 2014/121. https://eprint.iacr.org/2014/121.

[26] Keitaro Hiwatashi, Satsuya Ohata, and Koji Nuida. 2020. An Efficient Secure Division Protocol Using Approximate Multi-bit Product and New Constant-Round Building Blocks. In *ACNS 20, Part I (LNCS, Vol. 12146)*, Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi (Eds.). Springer, Heidelberg, Germany, Rome, Italy, 357–376. https://doi.org/10.1007/978-3-030-57808-4_18

[27] Atsunori Ichikawa, Wakaha Ogata, Koki Hamada, and Ryo Kikuchi. 2019. Efficient Secure Multi-Party Protocols for Decision Tree Classification. In *ACISP 19 (LNCS, Vol. 11547)*, Julian Jang-Jaccard and Fuchun Guo (Eds.). Springer, Heidelberg, Germany, Christchurch, New Zealand, 362–380. https://doi.org/10.1007/978-3-030-21548-4_20

[28] K. Ji, B. Zhang, T. Lu, L. Li, and K. Ren. 2022. UC Secure Private Branching Program and Decision Tree Evaluation. https://doi.org/10.1109/TDSC.2022.3202916

[29] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, Baltimore, MD, USA, 1651–1669.

[30] Marcel Keller and Peter Scholl. 2014. Efficient, Oblivious Data Structures for MPC. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C., 506–525. https://doi.org/10.1007/978-3-662-45608-8_27

[31] Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. 2019. SoK: Modular and Efficient Private Decision Tree Evaluation. *PoPETs* 2019, 2 (April 2019), 187–208. https://doi.org/10.2478/popets-2019-0026

[32] Peeter Laud. 2015. Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees. *PoPETs* 2015, 2 (April 2015), 188–205. https://doi.org/10.1515/popets-2015-0011

[33] Peeter Laud. 2015. A Private Lookup Protocol with Low Online Complexity for Secure Multiparty Computation. In *ICICS 14 (LNCS, Vol. 8958)*, Lucas Chi Kwong Hui, S. H. Qing, Elaine Shi, and S. M. Yiu (Eds.). Springer, Heidelberg, Germany, Hong Kong, 143–157. https://doi.org/10.1007/978-3-319-21966-0_11

[34] Sven Laur, Riivo Talviste, and Jan Willemson. 2013. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *ACNS 13 (LNCS, Vol. 7954)*, Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini (Eds.). Springer, Heidelberg, Germany, Banff, AB, Canada, 84–101. https://doi.org/10.1007/978-3-642-38980-1_6

[35] Lin Liu, Jinshu Su, Rongmao Chen, Jinrong Chen, Guangliang Sun, and Jie Li. 2019. Secure and Fast Decision Tree Evaluation on Outsourced Cloud Data. , 361–377 pages. https://doi.org/10.1007/978-3-030-30619-9_26

[36] Jack P. K. Ma, Raymond K. H. Tai, Yongjun Zhao, and Sherman S. M. Chow. 2021. Let's Stride Blindfolded in a Forest: Sublinear Multi-Client Decision Trees Evaluation. In *NDSS 2021*. The Internet Society, Virtual.

[37] Payman Mohassel and Peter Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 35–52. https://doi.org/10.1145/3243734.3243760

[38] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 19–38. https://doi.org/10.1109/SP.2017.12

[39] Hiraku Morita, Nuttapong Attrapadung, Satsuya Ohata, Koji Nuida, Shota Yamada, Kana Shimizu, Goichiro Hanaoka, and Kiyoshi Asai. 2018. Secure Division Protocol and Applications to Privacy-preserving Chi-squared Tests. , 530–534 pages. https://doi.org/10.23919/ISITA.2018.8664337

[40] Hiraku Morita, Nuttapong Attrapadung, Tadanori Teruya, Satsuya Ohata, Koji Nuida, and Goichiro Hanaoka. 2018. Constant-Round Client-Aided Secure Comparison Protocol. In *ESORICS 2018, Part II (LNCS, Vol. 11099)*, Javier López, Jianying Zhou, and Miguel Soriano (Eds.). Springer, Heidelberg, Germany, Barcelona, Spain, 395–415. https://doi.org/10.1007/978-3-319-98989-1_20

[41] Takashi Nishide and Kazuo Ohta. 2007. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In *PKC 2007 (LNCS, Vol. 4450)*, Tatsuaki Okamoto and Xiaoyun Wang (Eds.). Springer, Heidelberg, Germany, Beijing, China, 343–360. https://doi.org/10.1007/978-3-540-71677-8_23

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[43] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *ASIACCS 18*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM Press, Incheon, Republic of Korea, 707–721.

[44] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. 2017. Privacy-Preserving Decision Trees Evaluation via Linear Functions. In *ESORICS 2017, Part II (LNCS, Vol. 10493)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer, Heidelberg, Germany, Oslo, Norway, 494–512. https://doi.org/10.1007/978-3-319-66399-9_27

[45] Pierre Tholoniat. 2021. Sycret. https://pypi.org/project/sycret/

[46] Hikaru Tsuchida, Takashi Nishide, and Yusaku Maeda. 2020. Private Decision Tree Evaluation with Constant Rounds via (Only) SS-3PC over Ring. In *ProvSec 2020 (LNCS, Vol. 12505)*, Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang (Eds.). Springer, Heidelberg, Germany, Singapore, 298–317. https://doi.org/10.1007/978-3-030-62576-4_15

[47] Anselme Tueno, Yordan Boev, and Florian Kerschbaum. 2020. Non-interactive Private Decision Tree Evaluation. , 174–194 pages. https://doi.org/10.1007/978-3-030-49669-2_10

[48] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *PoPETs* 2019, 3 (July 2019), 26–49. https://doi.org/10.2478/popets-2019-0035

[49] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PoPETs* 2021, 1 (Jan. 2021), 188–208. https://doi.org/10.2478/popets-2021-0011

[50] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. 2016. Privately Evaluating Decision Trees and Random Forests. *PoPETs* 2016, 4 (Oct. 2016), 335–355. https://doi.org/10.1515/popets-2016-0043

[51] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, Toronto, Ontario, Canada, 162–167. https://doi.org/10.1109/SFCS.1986.25

[52] Yifeng Zheng, Huayi Duan, and Cong Wang. 2019. Towards Secure and Efficient Outsourcing of Machine Learning Classification. In *ESORICS 2019, Part I (LNCS, Vol. 11735)*, Kazue Sako, Steve Schneider, and Peter Y. A. Ryan (Eds.). Springer, Heidelberg, Germany, Luxembourg, 22–40. https://doi.org/10.1007/978-3-030-29959-0_2

A SHARE CONVERSION PROTOCOLS

We describe our protocols for share conversions here. Our ShareConv protocol requires a single invocation of RndGen to make the output values uniformly random when viewed from S_3. The ShareConv^{-1} protocol also requires RndGen as a sub-functionality to generate a 3-out-of-3 zero-sharing. The main idea for generating zero-sharings follows the technique proposed in [2].

Protocol 10 Share Conversion (ShareConv)

Functionality: $\langle\!\langle x \rangle\!\rangle \leftarrow \mathsf{ShareConv}(\llbracket x \rrbracket^N)$
Input: $(2,3)$-RSS arithmetic value $\llbracket x \rrbracket^N$
Output: $(2,2)$-SS arithmetic value $\langle\!\langle x \rangle\!\rangle$

1: S_1 and S_2 sample $r \xleftarrow{\$} \mathbb{Z}_N$ using RndGen
2: S_1 computes $y_1 = x_1 + x_2 + r$ where $\llbracket x \rrbracket_1^N = (x_1, x_2)$
3: S_2 computes $y_2 = x_3 - r$ where $\llbracket x \rrbracket_2^N = (x_2, x_3)$
4: S_1 returns $\langle\!\langle x \rangle\!\rangle_1 = y_1$ and S_2 returns $\langle\!\langle x \rangle\!\rangle_2 = y_2$

Protocol 11 Share Conversion (ShareConv^{-1})

Functionality: $\llbracket x \rrbracket \leftarrow \mathsf{ShareConv}^{-1}(\langle\!\langle x \rangle\!\rangle)$
Input: $\langle\!\langle x \rangle\!\rangle = (x_i, x_j)$, where x_i from S_i, x_j from S_j
Output: $(2,3)$-RSS arithmetic value $\llbracket x \rrbracket^N$

1: Servers generate a random share $\llbracket r \rrbracket = (r_1, r_2, r_3)$ using RndGen
2: S_i computes $m_i = x_i + r_{i+1} - r_i$ and sends it to S_{i+1}
3: S_j computes $m_j = x_j + r_{j+1} - r_j$ and sends it to S_{j+1}
4: S_k computes $m_k = r_{k+1} - r_k$ and sends it to S_{k+1}
5: Servers set $\llbracket x \rrbracket = ((m_1, m_2), (m_2, m_3), (m_3, m_1))$

B JI *ET AL.*'S PROTOCOL

We highlight the difference between Ji *et al.* [28]'s PDTE protocol and our proposed protocol. As briefly mentioned in Sect. 1.2, the PDTE protocol in [28] is different in three aspects.

First of all, the functionality they achieved was slightly different from Tsuchida *et al.* [46] and ours. For example, the comparison protocol does not consider switching operations such as less-than comparison and equality check. More precisely, [28] only considers a single operation, *i.e.,* the interval check.

Another disadvantage is that their path evaluation protocol does not let servers hold proper secret shares at the final step. Therefore, their PDTE protocol's output cannot be an input of other functionalities. To fix this problem, we believe that Ji *et al.*'s protocol needs an extra step to convert the final output into proper shares.

Finally, the PDTE protocol in [28] deploys the FSS for all three algorithms, the feature selection, comparison, and path evaluation, while our protocol only relies on the FSS when constructing the comparison protocol. Using FSS increases total communication complexity and storage cost because of generations of FSS keys. Table 5 shows the communication complexities.

C IMPLEMENTATION DETAILS

Table 4 shows the parameters of decision trees of each dataset using scikit-learn, which are the number of attributes, the depth of trees, and the number of nodes before padding. Note that we derived three different decision trees from the same dataset DIGITS, which are DIGITS-10, DIGITS-12, and DIGITS-15, respectively of depth 10, 12, and 15. It is realized by setting the maximum possible decision tree height when using scikit-learn.

Since our protocol only considers integers, we convert the decimals in the dataset (feature attributes) as well as the feature attributes in the trained decision tree model to 32-bit integers. This conversion is done by multiplying each attribute of each instance

with a suitable power of 10 to ensure that all of them are in integer form and the threshold values are also in integer form. Also, we generate secret sharings within the ring $\mathbb{Z}_2^{32}$, this includes the secret sharing of feature attribute values, threshold values, and the final classification values.

The run-time was counted using the Python Timer Function. The communication bytes were calculated by actually measuring the transmitted messages between servers. The times reported are averaged over 10 trials. Decimal numbers are truncated at the second decimal place for simplicity.