Zero Knowledge Proofs for Decision Tree Predictions and Accuracy

Jiaheng Zhang
UC Berkeley
jiaheng_zhang@berkeley.edu

Zhiyong Fang
Texas A&M University
zhiyong.fang.1997@tamu.edu

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

Dawn Song
UC Berkeley
dawnsong@berkeley.edu

ABSTRACT

Machine learning has become increasingly prominent and is widely used in various applications in practice. Despite its great success, the integrity of machine learning predictions and accuracy is a rising concern. The reproducibility of machine learning models that are claimed to achieve high accuracy remains challenging, and the correctness and consistency of machine learning predictions in real products lack any security guarantees.

In this paper, we initiate the study of zero knowledge machine learning and propose protocols for zero knowledge decision tree predictions and accuracy tests. The protocols allow the owner of a decision tree model to convince others that the model computes a prediction on a data sample, or achieves a certain accuracy on a public dataset, without leaking any information about the model itself. We develop approaches to efficiently turn decision tree predictions and accuracy into statements of zero knowledge proofs. We implement our protocols and demonstrate their efficiency in practice. For a decision tree model with 23 levels and 1,029 nodes, it only takes 250 seconds to generate a zero knowledge proof proving that the model achieves high accuracy on a dataset of 5,000 samples and 54 attributes, and the proof size is around 287 kilobytes.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Zero knowledge proofs; Machine learning; Decision tree

ACM Reference Format:
Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero Knowledge Proofs for Decision Tree Predictions and Accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3372297.3417278

1 INTRODUCTION

Machine learning has seen a great development over the past years, leading to important progress in various research areas such as computer vision, data mining, and natural language processing. Despite the great success, there are many security concerns of machine learning techniques, one of which is the *integrity* of machine learning models and their predictions. Newly developed machine learning models are claimed to achieve high accuracy, yet it is challenging to reproduce the results and validate these claims in many cases. In addition, even if a high quality model exists, it may not be used consistently in real-world products. For example, an online service for image classification claiming to use a particular model may simply return a random answer, and there is no guarantee on the integrity of the result for the clients. The authors in [4] report an extreme case where a company claims to use machine learning techniques to build robots delivering food automatically, yet the robots are actually operated by remote workers.

These scenarios urge the need for a solution to ensure that the owner indeed has a valid machine learning model, and it is used to compute the predictions correctly or it achieves high accuracy on public datasets. A naïve approach is to release the machine learning models publicly. However, it completely sacrifices the privacy of the machine learning models. Machine learning models are becoming important intellectual properties of the companies and cannot be shared publicly for validation. Releasing the machine learning models as black-box testing software does not address the issues as well. For example, in the machine learning service scenarios above, the customers can just take the black-box software away without paying. Even worse, recent research shows that one can infer sensitive information or reconstruct the machine learning models with only black-box accesses [25].

In this paper, we propose to address the problem of machine learning integrity using the cryptographic primitive of *zero knowledge proof* (ZKP). A zero knowledge proof allows a *prover* to produce a short proof π that can convince any *verifier* that the result of a public function f on the public input x and secret input w of the prover is $y = f(x, w)$. w is usually referred as the witness or auxiliary input. Zero knowledge proofs guarantee that the verifier rejects with overwhelming probability if the prover cheats on computing the result, while the proof reveals no extra information about the secret w beyond the result.

During the last decade, there has been great progress on generic ZKP schemes that are nearly practical. They allow proving arbitrary computations modeled as arithmetic circuits. In principle, we can apply these general-purpose ZKP schemes to address the problem of machine learning integrity. The prover proves that she knows a secret machine learning model that computes the prediction of an input or achieves the claimed accuracy on a public dataset, without leaking any additional information about the machine learning model. The proof is *succinct*, meaning that it is much smaller than the machine learning model and the prediction function. However, it is particularly challenging to construct efficient ZKP for machine learning predictions and accuracy tests because of the high overhead on the proof generation time. Because of these challenges, ZKP schemes for machine learning computations have not been widely studied in the literature.

Our contributions. In this paper, we initiate the study of zero knowledge machine learning predictions and accuracy, and propose several efficient schemes for zero knowledge decision trees. We also extend our techniques with minimal changes to support variants of decision trees, including regression, multivariate decision trees and random forests. Decision trees and random forests play important roles in various applications in practice, because of their good explainability and interpretability: the predictions can be explained by meaningful rules and conditions. They are widely used for product recommendations, fraud detection and automated trading in financial applications. Our concrete contributions are:

- **Zero knowledge decision tree predictions.** First, we propose an efficient protocol for zero knowledge decision tree predictions. After a setup phase to commit to a decision tree in linear time to the size of the decision tree, the prover time is only proportional to the length of the prediction path h, and the number of attributes d of the data. We apply several critical techniques in the literature of ZKP for computations in the random access memory (RAM) model in non-black-box ways, and translate the decision tree prediction to a small circuit of size $O(d + h)$.
- **Zero knowledge decision tree accuracy.** Second, we generalize our protocol to prove the accuracy of a decision tree in zero knowledge. We develop two important optimizations to bound the number of hashes in our ZKP backend to be exactly the number of nodes N in the decision tree. It is independent of the number of data samples to test, and is much less than 2^h if the decision tree is unbalanced.
- **Implementation and evaluations.** Finally, we fully implement our protocols and evaluate their performance on several real-world datasets. Notably, for a large decision tree with 23 levels and 1,029 nodes, it only takes 250s to generate a proof for its accuracy on a testing dataset with 5,000 samples and 54 attributes. The proof size is 287KB and the verification time is 15.6s.

1.1 Related Work

Zero knowledge proofs were introduced by Goldwasser et al. in [27] and generic constructions based on probabilistically checkable proofs were proposed in the seminal work of Kilian [30] and Micali [33]. In recent years there has been significant progress in efficient ZKP protocols and systems. Categorized by their underlying techniques, there are succinct non-interactive argument of knowledge (SNARK) schemes [9, 12, 17, 21, 24, 29, 34, 39], discrete-log-based schemes [7, 13, 18, 28], hash-based schemes [14], interactive oracle proofs (IOP) [6, 8, 10, 42] and interactive-proof-based schemes [40, 41, 43, 44]. Their security relies on different assumptions and settings, and they provide trade-offs between prover time and proof size. In our construction, we use the ZKP scheme proposed in [10], named Aurora, as our backend because of its fast prover time and good scalability. The proof size is relatively large compared to other schemes. Please refer to [40–42] for more details on the performance and comparisons of different ZKP schemes.

Most ZKP schemes model the computations as arithmetic circuits, while decision tree predictions are naturally in the RAM model with comparisons and conditional branching. Several papers [9, 11, 15, 39, 45] proposed ZKP schemes for RAM programs. We use some of their techniques in our constructions, without going through the heavy machinery of RAM-to-circuit reductions.

Zero knowledge proofs for machine learning applications have not been studied extensively before. In [26], Ghodsi et al. proposed a system named SafetyNet to delegate neural network predictions to a cloud server. It assumes that the verifier has the neural network and guarantees the soundness of the predictions. The scheme does not support witness from the prover, and there is no notion of zero knowledge. In [46], Zhao et al. proposed to use SNARK to validate neural network predictions. The prover commits to the values of all intermediate layers, and the verifier validates one layer randomly with a SNARK proof. The scheme does not provide negligible soundness. It also justifies the challenge of the overhead on the prover time as we mentioned in the introduction, as it is too expensive to apply SNARK to the whole machine learning model.

Finally, there is a rich literature on privacy-preserving decision tree predictions and training using secure multiparty computations (MPC), oblivious RAM (ORAM) and fully homomorphic encryptions (FHE) [16, 37, 38]. We note here that both the focus and the techniques of these schemes are quite different from zero knowledge proofs. These schemes primarily guarantee the privacy of the data during training and predictions. They do not provide integrity of the results and the succinctness of the communication. The settings and applications are also different from our zero knowledge decision trees.

2 PRELIMINARIES

We use $\mathsf{negl}(\cdot) : \mathbb{N} \to \mathbb{N}$ to denote the negligible function, where for each positive polynomial $f(\cdot)$, $\mathsf{negl}(k) < \frac{1}{f(k)}$ for sufficiently large integer k. Let λ denote the security parameter. Let $[m]$ denote the set of $\{1, 2, \cdots, m\}$. "PPT" standards for probabilistic polynomial time. We use bold letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$ to represent vectors, and $\mathbf{x}[i]$ denote the i-th element in vector $\mathbf{x}$.

2.1 Zero-knowledge Arguments

An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in \mathcal{R}$ for some input x. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w. We use $\mathcal{G}$ to represent the generation phase of the public parameters pp. Formally, consider the definition below, where we assume $\mathcal{R}$ is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 2.1. Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero-knowledge argument of knowledge for $\mathcal{R}$ if the following holds.

- **Completeness**. For every pp output by $\mathcal{G}(1^\lambda)$, $(x; w) \in \mathcal{R}$ and $\pi \leftarrow \mathcal{P}(x, w, \mathsf{pp})$,
$$\Pr[\mathcal{V}(x, \pi, \mathsf{pp}) = 1] = 1$$
- **Knowledge Soundness**. For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that given the access to the entire executing process and the randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $\pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp})$ and $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, x, \pi^*)$, the following probability is $\mathsf{negl}(\lambda)$:
$$\Pr[(x; w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \mathsf{pp}) = 1]$$
- **Zero knowledge**. There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, $(x; w) \in \mathcal{R}$, pp output by $\mathcal{G}(1^\lambda)$, it holds that
$$\mathsf{View}(\mathcal{V}^*(\mathsf{pp}, x)) \approx \mathcal{S}^{\mathcal{V}^*}(x)$$

We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a **succinct** argument system if the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) is $\mathsf{poly}(\lambda, |x|, \log |w|)$.

In the definition of zero knowledge, $\mathsf{View}(\mathcal{V}^*(\mathsf{pp}, x))$ denotes the veiw the verifier sees during the execution of the interactive process with $\mathcal{P}$ while $\mathcal{S}^{\mathcal{V}^*}(x)$ denotes the view generated by $\mathcal{S}$ given input x and transcript of $\mathcal{V}^*$, and $\approx$ denotes two distributions perfect indistinguishable. This definition is commonly used in existing transparent zero knowledge proof schemes [6, 10, 18, 40].

2.2 Our Zero-knowledge Argument Backend

With the recent progress on efficient zero knowledge proofs(ZKP) protocols, there are several general purpose ZKP systems with different trade-offs on the prover time, the proof size and the verification time. In our construction and implementation, we aim to optimize for fast prover time and to scale to large decision trees. Therefore, after careful comparisons among all existing ZKP systems, we choose the scheme named Aurora proposed in [10] as the ZKP backend in our zero knowledge decision tree construction. We state its properties in the following theorem. Note that our construction is also compatible with other ZKP systems.

Theorem 2.2. *[10]. Let λ be the security parameter, for a finite field $\mathbb{F}$ and a family of layered arithmetic circuit $C_{\mathbb{F}}$ over $\mathbb{F}$, there exists a zero knowledge argument of knowledge for the relation*
$$\mathcal{R} = \{(C, x; w) : C \in C_{\mathbb{F}} \wedge C(x; w) = 1\},$$
as defined in Definition 2.1, where x is the public input and w is the auxiliary input(private to the prover) to the circuit C.

Moreover, for every $(C, x; w) \in \mathcal{R}$, the running time of $\mathcal{P}$ is $O(|C| \log |C|)$ field operations. The running time of $\mathcal{V}$ is $O(|C|)$ and the proof size is $O(\log^2 |C|)$, where $|C|$ is the number of arithmetic gates in the circuit C.

Aurora has no trusted setup. Its security is proven in the random oracle model, and is plausibly post-quantum secure. It can be made non-interactive using Fiat-Shamir [23] in the random oracle model.

In addition, in order to build our zero knowledge decision tree scheme, we require an additional algorithm of the general purpose ZKP protocol to commit the witness. This is formalized as "Commit-and-Prove" in [19], and is naturally supported by Aurora and most of ZKP systems. We denote the algorithm as $\mathsf{com}_w \leftarrow \mathsf{Commit}(w, \mathsf{pp})$. It is executed after $\mathcal{G}$ and before $\mathcal{P}$, and $\mathcal{V}$ additionally takes com_w as an input. It satisfies the extractability of commitment. Similar to the extractability in Definition 2.1, there exists a PPT extractor $\mathcal{E}$, given any tuple $(\mathsf{pp}, x, \mathsf{com}^*_w)$ and the executing process of $\mathcal{P}^*$, it could always extract a witness w^* such that $\mathsf{com}^*_w \leftarrow \mathsf{Commit}(w^*, \mathsf{pp})$ except for the negligible probability in λ. Formally speaking, $\mathsf{com}^*_w = \mathsf{Commit}(\mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, x, \mathsf{com}^*_w), \mathsf{pp})$.

3 ZERO KNOWLEDGE DECISION TREE

In this section, we present our main construction of zero knowledge decision tree predictions. We first introduce the background on decision trees. Then we formally define the notion of zero knowledge decision tree predictions, and present our protocol with the security analysis.

3.1 Decision Tree

Decision tree is one of the most commonly used machine learning algorithms. It performs particularly well for classification problems, and has good explainability and interpretability. Therefore, it is widely deployed in applications such as product recommendations, fraud detection and automated trading.

For simplicity, we focus on binary decision trees for classification problems in our presentation, but our techniques can be generalized naturally to decision trees with higher degrees, decision trees for regression problems, multivariate decision trees and random forests with small changes. We present these variants in Appendix A. In a decision tree $\mathcal{T}$, each intermediate node contains an attribute, each branch represents a decision and each leaf node denotes an outcome (categorical or continues value). More formally, each internal node v has an associated attribute index $v.\mathsf{att}$ from the set $[d]$ of d attributes, a threshold $v.\mathsf{thr}$ and two children $v.\mathsf{left}$ and $v.\mathsf{right}$. Each leaf node u stores the classification result $u.\mathsf{class}$. Each data sample is represented as a size-d vector $\mathbf{a}$ of values corresponding to each attribute. The algorithm of decision tree prediction is shown in Algorithm 1. It starts from the root of $\mathcal{T}$. For each node of v in $\mathcal{T}$, it compares $\mathbf{a}[v.\mathsf{att}]$ with $v.\mathsf{thr}$, and moves to $v.\mathsf{left}$ if $\mathbf{a}[v.\mathsf{att}] < \mathsf{v.thr}$, and $v.\mathsf{right}$ otherwise. Eventually, the algorithm reaches a leaf node u and the result of the prediction is $u.\mathsf{class}$.

To train a decision tree, given a training dataset, the decision tree is obtained by splitting the set into subsets from the root to the children. The splitting is based on some splitting rules by maximizing the certain objective function such as the information gain

Algorithm 1 Decision Tree Prediction

Input: Decision tree $\mathcal{T}$, data sample $\mathbf{a}$
Output: classification $y_{\mathbf{a}}$

```
1: $v := \mathcal{T}.\mathsf{root}$
2: while $v$ is not a leaf node do
3:     if $\mathbf{a}[v.\mathsf{att}] < \mathsf{v.thr}$ then
4:         $v := v.\mathsf{left}$
5:     else
6:         $v := v.\mathsf{right}$
7: return $v.\mathsf{class}$
```

and the splitting process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion will halt when the subset of a node has the same classification, or when splitting can not increase the value of the objective function. In our scheme, we only consider proving predictions given the pretrained decision tree, and the training process is out of the scope.

3.2 Zero Knowledge Decision Tree Prediction

Motivated by the applications mentioned in the introduction, in our model, the prover owns a pre-trained decision tree. The prover commits to the decision tree first, and then later the verifier queries for the prediction of a data sample. The prover generates a proof together with the result to convince the verifier its validity.

Formally speaking, let $\mathbb{F}$ be a finite field, $\mathcal{T}$ be a binary decision tree of height h and N nodes ($N \leq 2^h - 1$). Suppose the test dataset is $\mathcal{D} \subseteq \mathbb{F}^d$, in which each data point has d features. So for each data $\mathbf{a}$ in $\mathcal{D}$, $\mathbf{a} \in \mathbb{F}^d$. Let $[M]$ be the set of all target classifications. We treat the decision tree algorithm as a mapping $\mathcal{T} : \mathbb{F}^d \rightarrow [M]$. For a data point $\mathbf{a} \in \mathcal{D}$, $\mathcal{T}(\mathbf{a}) \in [M]$ is the prediction for the classification of $\mathbf{a}$ using the decision tree algorithm on $\mathcal{T}$. A zero-knowledge decision tree scheme (zkDT) consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkDT}.\mathcal{G}(1^\lambda)$: given the security parameter, generate the public parameter pp.
- $\mathsf{com}_\mathcal{T} \leftarrow \mathsf{zkDT.Commit}(\mathcal{T}, \mathsf{pp}, r)$: commit the decision tree $\mathcal{T}$ with a random point r generated by the prover.
- $(y_\mathbf{a}, \pi) \leftarrow \mathsf{zkDT}.\mathcal{P}(\mathcal{T}, \mathbf{a}, \mathsf{pp})$: given a data $\mathbf{a}$, run the decision tree algorithm to get $y_\mathbf{a} = \mathcal{T}(\mathbf{a})$ and the corresponding proof π.
- $\{0, 1\} \leftarrow \mathsf{zkDT}.\mathcal{V}(\mathsf{com}_\mathcal{T}, h, \mathbf{a}, y_\mathbf{a}, \pi, \mathsf{pp})$: validate the prediction of $\mathbf{a}$ given $y_\mathbf{a}$ and π obtained from the prover.

In our scheme, we assume the height of the decision tree (or an upper bound) is known to both parties. $\mathsf{com}_\mathcal{T}$ is the commitment of the decision tree. $y_\mathbf{a}$ denotes the class of $\mathbf{a}$ returned by the decision tree. And π denotes the proof generated by the prover. $\{0, 1\}$ represents reject or accept output by the verifier after seeing the classification and the proof.

Definition 3.1. We say that a scheme is a zero knowledge decision tree if the following holds:

- **Completeness.** For any decision tree $\mathcal{T}$ and a data point $\mathbf{a} \in \mathbb{F}^d$, $\mathsf{pp} \leftarrow \mathsf{zkDT}.\mathcal{G}(1^\lambda)$, $\mathsf{com}_\mathcal{T} \leftarrow \mathsf{zkDT.Commit}(\mathcal{T}, \mathsf{pp}, r)$, $(y_\mathbf{a}, \pi) \leftarrow \mathsf{zkDT}.\mathcal{P}(\mathcal{T}, \mathbf{a}, \mathsf{pp})$, it holds that

$$\Pr\left[\mathsf{zkDT}.\mathcal{V}(\mathsf{com}_\mathcal{T}, h, \mathbf{a}, y_\mathbf{a}, \pi, \mathsf{pp}) = 1\right] = 1$$

- **Soundness.** For any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr\begin{bmatrix} \mathsf{pp} \leftarrow \mathsf{zkDT}.\mathcal{G}(1^\lambda) \\ (\mathcal{T}^*, \mathsf{com}_{\mathcal{T}^*}, \mathbf{a}, y^*_\mathbf{a}, \pi^*) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, r) \\ \mathsf{com}_{\mathcal{T}^*} = \mathsf{zkDT.Commit}(\mathcal{T}^*, \mathsf{pp}, r) \\ \mathsf{zkDT}.\mathcal{V}(\mathsf{com}_{\mathcal{T}^*}, h, \mathbf{a}, y^*_\mathbf{a}, \pi^*, \mathsf{pp}) = 1 \\ \mathcal{T}(\mathbf{a}) \neq y^*_\mathbf{a} \end{bmatrix}$$

- **Zero Knowledge.** For security parameter λ, $\mathsf{pp} \leftarrow \mathsf{zkDT}.\mathcal{G}(1^\lambda)$, for a decision tree $\mathcal{T}$ with h levels, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:

```
Real_{A,T}(pp):
  (1) $\mathsf{com}_\mathcal{T} \leftarrow \mathsf{zkDT.Commit}(\mathcal{T}, \mathsf{pp}, r)$
  (2) $\mathbf{a} \leftarrow \mathcal{A}(h, \mathsf{com}_\mathcal{T}, \mathsf{pp})$
  (3) $(y_\mathbf{a}, \pi) \leftarrow \mathsf{zkDT}.\mathcal{P}(\mathcal{T}, \mathbf{a}, \mathsf{pp})$
  (4) $b \leftarrow \mathcal{A}(\mathsf{com}_\mathcal{T}, h, \mathbf{a}, y_\mathbf{a}, \pi, \mathsf{pp})$
  (5) Output b
```

```
Ideal_{A,S^A}(pp, h):
  (1) $\mathsf{com} \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{pp}, h)$
  (2) $\mathbf{a} \leftarrow \mathcal{A}(h, \mathsf{com}, \mathsf{pp})$
  (3) $(y_\mathbf{a}, \pi) \leftarrow \mathcal{S}_2^{\mathcal{A}}(\mathsf{com}, h, \mathbf{a}, \mathsf{pp})$, given oracle access
      to $y_\mathbf{a} = \mathcal{T}(\mathbf{a})$.
  (4) $b \leftarrow \mathcal{A}(\mathsf{com}, h, \mathbf{a}, y_\mathbf{a}, \pi, \mathsf{pp})$
  (5) Output b
```

For any PPT algorithm $\mathcal{A}$ and all decision tree $\mathcal{T}$ with the height of h, there exists simulator $\mathcal{S}$ such that

$$|\Pr[\mathsf{Real}_{\mathcal{A},\mathcal{T}}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^\mathcal{A}}(\mathsf{pp}, h) = 1]| \leq \mathsf{negl}(\lambda).$$

Intuition of the specific construction of zkDT: given the algorithm of decision tree and the definition of the zero knowledge decision tree protocol(zkDT), we will focus on the specific construction of our zkDT scheme in the subsequent subsections. The general idea of the construction is as follows. In the beginning, the prover sends the committment of a decision tree $\mathcal{T}$, $\mathsf{com}_\mathcal{T}$, to the verifier. After receiving $\mathbf{a}$ from the verifier, the prover computes $y_\mathbf{a}$ and the corresponding witness w for proving $y_\mathbf{a} = \mathcal{T}(\mathbf{a})$, then sends $y_\mathbf{a}$ to the verifier. We treat it as some relationship $\mathcal{R} = ((y_\mathbf{a}, \mathbf{a}, \mathsf{com}_\mathcal{T}); w)$ in Definition 2.1. Then the verifier and the prover invoke the backend zero-knowledge proofs protocol in subsection 2.2 to verify the relationship $\mathcal{R}$ without leaking any information of $\mathcal{T}$ except for $y_\mathbf{a}$.

3.3 Authenticated Decision Tree

We start with the scheme to commit to a decision tree. A naive approach to do so is to simply compute the cryptographic hash of the whole decision tree. However, later when the prover wants to prove the prediction of a data sample using the committed decision tree, the whole decision tree has to be included as the witness of the zero knowledge proof, while only the path from the root to the leaf node of the output is relevant for the prediction. This would introduce a high overhead to recompute the hash of the whole tree.

One could also build a Merkle hash tree [32] on top of all the nodes in the decision tree (with a predefined order). Then later in $\mathsf{zkDT}.\mathcal{P}$, the prover associates each node in the prediction path with a valid Merkle tree proof. The overhead of this approach will be $O(h \log N)$ hashes in the zero knowledge proof backend to validate the all nodes in the prediction path. Instead, we propose to leverage the rich literature of authenticated data structures [36]. We propose to build authenticated decision trees (ADT) directly on the data structure of decision trees so that proving a prediction path only relies on the information stored in the nodes.

Construction of ADT: Figure 1 illustrates our construction of ADT. The construction is very similar to Merkle hash tree, with the difference that the data stored in an intermediate node of the decision tree is also included in the computation of its hash, together with the hashes of the two children. In particular, each node v contains the attribute $v.\mathsf{att}$, the threshold $v.\mathsf{thr}$, the pointers to the

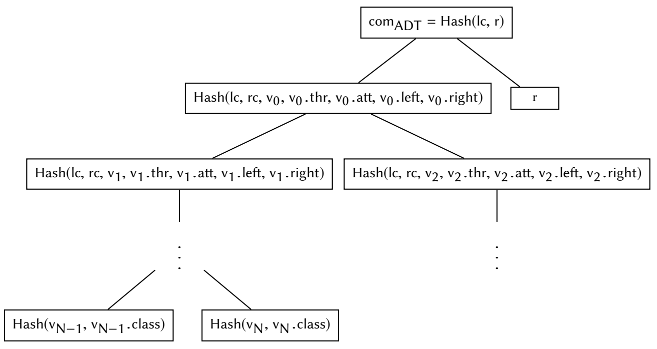

com_ADT = Hash(lc, r)
Hash(lc, rc, v_0, v_0.thr, v_0.att, v_0.left, v_0.right)
r
Hash(lc, rc, v_1, v_1.thr, v_1.att, v_1.left, v_1.right)
Hash(lc, rc, v_2, v_2.thr, v_2.att, v_2.left, v_2.right)
Hash(v_{N−1}, v_{N−1}.class)
Hash(v_N, v_N.class)

Figure 1: Committing algorithm of ADT scheme, lc **and** rc **represent the left child value and the right child value respectively.**

children v.left and v.right, and the hashes of its children. In the real implementation, we use different identities in $[N]$ to represent the nodes in $\mathcal{T}$. Therefore, v.left and v.right are identities of the left child and right child of v respectively.

The verification algorithm is also similar to that of the Merkle hash tree. To validate the prediction for a data sample, the proof includes the prediction path from the root to the leaf node that outputs the prediction result. In addition, the proof also includes the hashes of the siblings of the nodes along the prediction path. With the proof, the verifier can recompute the root hash and compare it with the commitment. In this way, to prove the validity of a prediction, the verification only computes $O(h)$ hashes. Note that in our construction of zkDT, the verification of ADT is never executed by the verifier directly. As we will show in the next section, the prover further proves that there exists such a valid prediction path through a general purpose zero knowledge proof. Because of this design, the verification of ADT does not have to be zero knowledge.

The algorithms of our ADT are in the following. Note that in order to prove the zero knowledge property of the scheme later, the commitment has to be randomized and we add a random point r to the root of the decision tree and use the hash of the root concatenated with r as the final commitment, as shown in Figure 1. Moreover, for the purpose of our application, the ADT does not have to support dynamic insertions and deletions, which simplifies the construction significantly.

- $\mathsf{pp} \leftarrow \mathsf{ADT}.\mathcal{G}(1^\lambda)$: Sample a collision resistant hash function from the family of hash functions.
- $\mathsf{com}_{\mathsf{ADT}} \leftarrow \mathsf{ADT.Commit}(\mathcal{T}, \mathsf{pp}, r)$: compute hashes from leaf nodes to the root of $\mathcal{T}$ with the random point r as shown in Figure 1.
- $\pi_{\mathsf{ADT}} \leftarrow \mathsf{ADT}.\mathcal{P}(\mathcal{T}, \mathsf{Path}, \mathsf{pp})$: given a path in $\mathcal{T}$, π_{ADT} contains all siblings of the nodes along the path Path and the randomness r in Figure 1.
- $\{0,1\} \leftarrow \mathsf{ADT}.\mathcal{V}(\mathsf{com}_{\mathsf{ADT}}, \mathsf{Path}, \pi_{\mathsf{ADT}}, \mathsf{pp})$: given Path and π_{ADT}, recompute hashes along Path with π_{ADT} as the same progress in Figure 1 and compare the root hash with $\mathsf{com}_{\mathsf{ADT}}$. Output 1 if they are the same, otherwise output 0.

Given these algorithms and the construction, we have the following theorem:

Theorem 3.2. *Let $\mathcal{T}$ be a decision tree with h levels and N nodes, our* ADT *scheme satisfies the following properties.*

- ***Completeness:*** *if* $\mathsf{pp} \leftarrow \mathsf{ADT}.\mathcal{G}(1^\lambda)$, $\mathsf{Path} \in \mathcal{T}$, $\mathsf{com}_{\mathsf{ADT}} \leftarrow \mathsf{ADT.Commit}(\mathcal{T}, \mathsf{pp}, r)$ *and* $\pi_{\mathsf{ADT}} \leftarrow \mathsf{ADT}.\mathcal{P}(\mathcal{T}, \mathsf{Path}, \mathsf{pp})$*, then*
$$\Pr[\mathsf{ADT}.\mathcal{V}(\mathsf{com}_{\mathsf{ADT}}, \mathsf{Path}, \pi_{\mathsf{ADT}}, \mathsf{pp}) = 1] = 1$$
- ***Soundness:*** *for any PPT adversary $\mathcal{A}$, if* $\mathsf{pp} \leftarrow \mathsf{ADT}.\mathcal{G}(1^\lambda)$, $\mathsf{com}_{\mathsf{ADT}} \leftarrow \mathsf{ADT.Commit}(\mathcal{T}, \mathsf{pp}, r)$, $\pi^*_{\mathsf{ADT}} \leftarrow \mathcal{A}(\mathcal{T}, \mathsf{Path}, \mathsf{pp})$ *but* $\mathsf{Path} \notin \mathcal{T}$*, then*
$$\Pr[\mathsf{ADT}.\mathcal{V}(\mathsf{com}_{\mathsf{ADT}}, \mathsf{Path}, \pi^*_{\mathsf{ADT}}, \mathsf{pp}) = 1] \le \mathsf{negl}(\lambda)$$
- ***Hiding:*** $\mathsf{pp} \leftarrow \mathsf{ADT}.\mathcal{G}(1^\lambda)$*, for any decision tree $\mathcal{T}$ with h levels, any PPT algorithm $\mathcal{A}$, there exists a simulator $\mathcal{S}_{\mathsf{ADT}}$: let* $\mathsf{com}_{\mathsf{ADT}} = \mathsf{ADT.Commit}(\mathcal{T}, \mathsf{pp}, r)$ *and* $\mathsf{com}'_{\mathsf{ADT}} = \mathcal{S}_{\mathsf{ADT}}(\mathsf{pp}, h, r)$*,*
$$|\Pr[\mathcal{A}(\mathsf{com}_{\mathsf{ADT}}, \mathsf{pp}) = 1] - \Pr[\mathcal{A}(\mathsf{com}'_{\mathsf{ADT}}, \mathsf{pp}) = 1]| \le \mathsf{negl}(\lambda)$$
In addition, the time of ADT.Commit *is $O(N)$, and the prover time, verification time and the proof size are all $O(h)$.*

Proof Sketch: The completeness of our ADT scheme is straight forward. The soundness holds because of the collision-resistance of the hash function. To prove the hiding property, we can construct a simulator $\mathcal{S}_{\mathsf{ADT}}(\mathsf{pp}, h, r) = \mathsf{ADT.Commit}(\vec{0}_h, \mathsf{pp}, r)$, where $\vec{0}_h$ represents a decision tree with h levels and all nodes containing only 0 strings. It is indistinguishable from the real algorithm because the verifier does not know r, which is uniformly random. We omit the formal proofs here.

3.4 Proving the validity of the prediction

Following the algorithm to commit to a decision tree, we further present our protocol to prove the correctness of the prediction. A natural idea is to invoke $\mathsf{ADT}.\mathcal{P}$ and $\mathsf{ADT}.\mathcal{V}$ directly to obtain the valid prediction path, and check the prediction in Algorithm 1. However, this is not zero knowledge as the verifier would learn a path in the decision tree. We propose to apply an additional zero knowledge proof protocol on top of this validation. As mentioned in the previous section, the prover instead proves that there exists a valid prediction path such that $\mathsf{ADT}.\mathcal{V}$ would accept, and the prediction algorithm is correctly executed. By applying generic zero knowledge proofs on this relationship, the prediction path and the hashes of the siblings remain confidential as the witness, and the output is merely 1 or 0, i.e. all the checks are satisfied or not. In this way, the protocol is both sound and zero knowledge.

However, efficiently designing zero knowledge proof protocols for such an relationship turns out to be non-trivial. This is because every node v on the prediction path for a data sample $\mathbf{a}$ will access one attribute from $\mathbf{a}$ indexed by v.att for comparison, which is a classical random access operation. Most generic zero knowledge proof protocols represent computations as arithmetic circuits. Implementing the decision tree prediction algorithm as an arithmetic circuit introduces a high overhead on the prover time. In particular, each comparison in an internal node leads to an overhead of $O(d)$, and the overall size of the circuit is $O(dh)$. There are a few RAM-based generic zero knowledge proof protocols [9, 11, 15, 45] in the literature that represent computations as RAM programs. However, though asymptotically the prover time only depends on the running time of the RAM program, the concrete overhead is very high

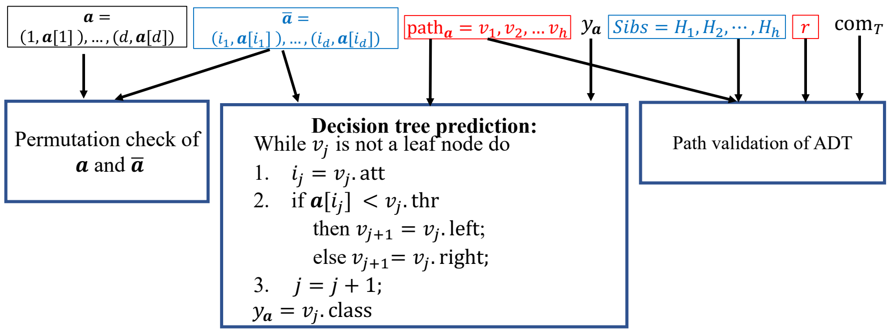

$\boldsymbol{a} = (1, \boldsymbol{a}[1]), \ldots, (d, \boldsymbol{a}[d])$
$\bar{\boldsymbol{a}} = (i_1, \boldsymbol{a}[i_1]), \ldots, (i_d, \boldsymbol{a}[i_d])$
$\text{path}_{\boldsymbol{a}} = v_1, v_2, \ldots v_h$
$y_{\boldsymbol{a}}$
$Sibs = H_1, H_2, \cdots, H_h$
r
com_T
Permutation check of $\boldsymbol{a}$ and $\bar{\boldsymbol{a}}$
Decision tree prediction:
While v_j is not a leaf node do
1. $i_j = v_j.\text{att}$
2. if $\boldsymbol{a}[i_j] < v_j.\text{thr}$
then $v_{j+1} = v_j.\text{left}$;
else $v_{j+1} = v_j.\text{right}$;
3. $j = j + 1$;
$y_{\boldsymbol{a}} = v_j.\text{class}$
Path validation of ADT

Figure 2: Zero knowledge decision tree prediction. Public inputs are in black, secret witness is in red, and extended witness for efficiency is in blue.

(thousands of arithmetic gates per step of the program). Instead, in our construction, we apply some of the ideas in these work in a non-black-box way to design a specific and efficient zero knowledge proof protocol for the validation of decision tree predictions.

Reducing decision tree prediction circuit efficiently. Figure 2 illustrates our design to efficiently reduce the validity of the prediction using a committed decision tree to an arithmetic circuit. As shown in the figure, the public input (in black) of the circuit consists of the data sample $\mathbf{a}$, the commitment of the decision tree $\text{com}_{\mathcal{T}}$ and the prediction result $y_{\mathbf{a}}$. The secret witness (in red) from the prover includes the prediction path $\text{path}_{\mathbf{a}}$, and the randomness r used in the commitment of ADT (for technical reasons to prove zero knowledge). In order to improve the efficiency, the prover further inputs the siblings of nodes on the prediction path, and the permutation $\bar{\mathbf{a}}$ of the data sample $\mathbf{a}$ ordered by v.att of the nodes on the prediction path as part of the witness (in blue). The purpose of these "extended" witness will be explained below. The whole circuit consists of three parts: (1) validating the prediction algorithm of the decision tree, (2) checking the permutation between $\mathbf{a}$ and $\bar{\mathbf{a}}$, and (3) checking the validity of the prediction path in the committed decision tree. Finally, the output of the circuit is either 1 or 0, denoting either all the conditions are satisfied or some check fails.

Decision tree prediction. The first component of the circuit is to validate the prediction algorithm. With the help of $\bar{\mathbf{a}}$, this can be efficiently implemented using an arithmetic circuit. In particular, we slightly modify the representation of $\mathbf{a}$ and $\bar{\mathbf{a}}$ to be index-value pairs, i.e., $\mathbf{a} = (1, \mathbf{a}[1]), \ldots, (d, \mathbf{a}[d])$ and $\bar{\mathbf{a}} = (i_1, \mathbf{a}[i_1]), \ldots, (i_d, \mathbf{a}[i_d])$. Under this representation, the circuit simply checks that for every internal node v_j on the prediction path ($j = 1, \ldots, h-1$), (1) $v_j.\text{att} = i_j$, and (2) if $\mathbf{a}[i_j] < v_j.\text{thr}$, $v_{j+1} = v_j.\text{left}$, otherwise $v_{j+1} = v_j.\text{right}$. As we explained in the previous subsection, $v, v.\text{left}, v.\text{right} \in [N]$. The equality tests and comparisons are computed using standard techniques in the literature of circuit-based zero knowledge proof with the help of auxiliary input [34]. Finally, the circuit checks if $y_{\mathbf{a}} = v_h.\text{class}$. The circuit outputs 1 if all the checks pass, and outputs 0 otherwise. The total number of gates in this part is $O(d + h)$, which is asymptotically the same as the plain decision tree prediction in Algorithm 1.

Note that if $h < d$, which is usually true in practice, the circuit only checks the indices of the first $h - 1$ pairs in $\bar{\mathbf{a}}$. The rest of the indices are arbitrary, as long as $\bar{\mathbf{a}}$ is a permutation of $\mathbf{a}$. It does not affect the correctness or the soundness of the scheme, as those attributes are not used for prediction anyway. In addition, concrete decision trees are usually not balanced. The prover and the verifier can either agree on the length of the prediction path and construct a separate circuit for every data sample, or use the height of the tree as an upper-bound to construct the same circuit for all data samples. The former is more efficient, but leaks the length of the prediction paths. Both options are supported by our scheme and the asymptotic complexity are the same. For simplicity, we abuse the notation and use h both for the height of the tree and for (an upper bound of) the length of the prediction path.

Permutation test. The second component is to check that $\bar{\mathbf{a}}$ is indeed a permutation of $\mathbf{a}$. Together with the first component, it ensures that $y_{\mathbf{a}}$ is the correct prediction result of $\mathbf{a}$ using the prediction path $\text{path}_{\mathbf{a}}$. The construction is inspired by the RAM-based zero knowledge proof systems and we also apply the techniques of characteristic polynomials proposed in [15, 43, 45] to check permutations. In particular, the characteristic polynomial of a vector $\mathbf{c} = (\mathbf{c}[1], \cdots, \mathbf{c}[d]) \in \mathbb{F}^d$ is $\chi_{\mathbf{c}}(x) = \Pi_{i=1}^{d}(x - \mathbf{c}[i])$, the polynomial with roots $\mathbf{c}[i]$. To prove permutations between $\mathbf{c}$ and $\bar{\mathbf{c}}$, it suffices to show that their characteristic polynomial evaluates to the same value at a random point $r \in \mathbb{F}$ chosen by the verifier:

$$\Pi_{i=1}^{d}(r - \mathbf{c}[i]) = \Pi_{i=1}^{d}(r - \bar{\mathbf{c}}[i])$$

The soundness error is $\frac{d}{|\mathbb{F}|}$ by Schwartz-Zippel Lemma [35, 47].

In our construction, however, we need to prove that two vectors of pairs $\mathbf{a}$ and $\bar{\mathbf{a}}$ are permutations of each other. To this end, we use the approach proposed in [15] to pack each pair to a single value by a random linear combination. The verifier chooses a random point $z \in \mathbb{F}$. For each pair $(j, \mathbf{a}[j])$ in $\mathbf{a}$ and $(i_j, \bar{\mathbf{a}}[j])$ in $\bar{\mathbf{a}}$, the circuit computes $\mathbf{c}[j] = \mathbf{a}[j] + z \times j$ and $\bar{\mathbf{c}}[j] = \bar{\mathbf{a}}[j] + z \times i_j$. We then invoke the characteristic polynomial evaluation directly on $\mathbf{c}$ and $\bar{\mathbf{c}}$.

The completeness is straight forward. To prove the soundness, suppose $\bar{\mathbf{a}}$ is not a permutation of $\mathbf{a}$, then there must exist a pair $(i, \mathbf{a}[i])$ not appearing in $\bar{\mathbf{a}}$. After the packing process, for each j, $\Pr[\mathbf{a}[i] + z \times i = \bar{\mathbf{a}}[j] + z \times i_j | (i, \mathbf{a}[i]) \neq (i_j, \bar{\mathbf{a}}[j])] \leq \frac{1}{|\mathbb{F}|}$ by Schwartz-Zippel Lemma. Therefore, although $\mathbf{a}[i] + z \times i$ is one root of the characteristic polynomial of $\mathbf{c}$, the probability of that it is also one root of the characteristic polynomial for $\bar{\mathbf{c}}$ is at most $\frac{d}{|\mathbb{F}|}$ by the union bound. Combining with the soundness of the characteristic polynomial checking, we obtain that our method has the soundness error at most $\frac{2d}{|\mathbb{F}|}$, which is also negligible of λ. The technique can be extended to verify the permutation of vectors of more than two elements by packing them with a polynomial of z.

The circuit outputs 1 if and only if the above check passes. The number of gates is $O(d)$. Here we assume that every attribute is only used at most once in any prediction path. When an attribute can be used multiple times, which is also very common in practice, the circuit instead checks that $\bar{\mathbf{a}}$ is a *multiset* of $\mathbf{a}$, i.e., every pair in $\mathbf{a}$ appears in $\bar{\mathbf{a}}$ with cardinality greater than or equal to 0. We will present a technique to check the multiset relationship in Section 4. The sub-circuits for decision tree prediction and path validation remain unchanged, and the total number of gates in the circuit is the same asymptotically, as the size of $\bar{\mathbf{a}}$ in bounded by h in this case.

Path validation. Finally, the only missing component is to ensure that the prediction path is indeed valid as committed before. The third sub-circuit implements the ADT.$\mathcal{V}$ algorithm with input $\mathsf{path}_{\mathbf{a}}$, all sibling hash values on $\mathsf{path}_{\mathbf{a}}$, random point r and $\mathsf{com}_{\mathcal{T}}$. The circuit recomputes the hashes from the leaf to the root, and compares the root hash with $\mathsf{com}_{\mathcal{T}}$. In total, the circuit computes $O(h)$ hashes, which justifies the design of our ADT scheme.

Finally, the circuit aggregates all three checks and output 1 if and only if all checks pass. The size of the whole circuit is $O(d+h)$, which is also asymptotically optimal. The prover and the verifier then execute a generic circuit-based zero knowledge proof protocol on the circuit in Figure 2.

3.5 Putting Everything Together

In this section, we combine everything together and formally present our zero knowledge decision tree prediction scheme in Protocol 1. Our scheme has a transparent setup phase where zkDT.$\mathcal{G}$ does not have a trapdoor. It merely samples a collision-resistant hash function for ADT, and executes the algorithm $\mathcal{G}$ in the generic zero knowledge proof. Using Aurora as our backend, it also samples a hash function modeled as a random oracle.

We have the following theorem:

Theorem 3.3. *Protocol 1 is a zero knowledge decision tree scheme by Definition 3.1.*

Proof. **Completeness.** As explained in Section 3.3 and 3.4, the circuit in zkDT.$\mathcal{P}$ outputs 1 if $y_{\mathbf{a}}$ is the correct prediction of $\mathbf{a}$ output by Algorithm 1 on $\mathcal{T}$. Therefore, the correctness of Protocol 1 follows the correctness of the ADT and the zero knowledge proof protocol by Theorem 3.2 and 2.2.

Soundness. By the extractability of commitment of in Theorem 2.2, with overwhelming probability, there exists a PPT extractor $\mathcal{E}$ such that given com_w, it extracts a witness w^* such that com_w = ZKP.Commit(w^*, pp_2). By the soundness of zkDT in Definition 3.1, if $\mathsf{com}_{\mathcal{T}}$ = zkDT.Commit($\mathcal{T}$, pp, r) and zkDT.$\mathcal{V}$($\mathsf{com}_{\mathcal{T}}$, h, $\mathbf{a}$, $y_{\mathbf{a}}$, π, pp) = 1 but $y_{\mathbf{a}} \neq \mathcal{T}(\mathbf{a})$, let com_w = ZKP.Commit(w^*, pp_2) during the interactive process in Protocol 1, then there are two cases.

- Case 1: $w^* = (\bar{\mathbf{a}}^*, path^*_{\mathbf{a}}, aux^*, r)$ satisfying to $C((\mathsf{com}_{\mathcal{T}}, \mathbf{a}, y_{\mathbf{a}}, \mathbf{r}'); w^*) = 1$. Then we could know either $\mathsf{path}^*_{\mathbf{a}}$ is not a path in $\mathcal{T}$ but passing the verification for $\mathsf{com}_{\mathcal{T}}$, or $\bar{\mathbf{a}}^*$ is not a permutation of $\mathbf{a}$ but passing the permutation test. The probability of both events are $\mathsf{negl}(\lambda)$ as claimed by the soundness the ADT scheme and the soundness of the characteristic polynomial check respectively. Hence, the probability that $\mathcal{P}$ could generate such w^* is also $\mathsf{negl}(\lambda)$ by the union bound.
- Case 2: $w^* = (\bar{\mathbf{a}}^*, path^*_{\mathbf{a}}, aux^*, r)$ but $C((\mathsf{com}_{\mathcal{T}}, \mathbf{a}, y_{\mathbf{a}}, \mathbf{r}'); w^*) = 0$. Then according to the soundness of Aurora, given the commitment com^*_w, the adversary could generate a proof π_w making $\mathcal{V}$ accept the incorrect witness and output 1 with probability $\mathsf{negl}(\lambda)$.

Combining these two cases, the soundness of the zkDT scheme is also $\mathsf{negl}(\lambda)$.

Zero-knowledge. In order to prove the zero-knowledge property, we construct a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ in Figure 3. Let $\mathcal{S}_{\mathsf{ADT}}$ and $\mathcal{S}_{\mathsf{ZKP}}$ represent the simulator for ADT protocol and the backend ZKP protocol respectively. The proof follows by a standard hybrid argument.

Hybrid H_0: H_0 behaves in exactly the same way as the honest prover in Protocol 1.

Hybrid H_1: H_1 uses the real zkDT.Commit($\mathcal{T}$, pp, r) in Protocol 1

Protocol 1 (Zero Knowledge Decision Tree(zkDT)). *Let λ be the security parameter, $\mathbb{F}$ be a prime field, $\mathcal{T}$ be a decision with h levels, C be the arithmetic circuit in Figure 2. Let $\mathcal{P}$ and $\mathcal{V}$ be the prover and the verifier respectively. We use* ZKP.$\mathcal{G}$, ZKP.Commit, ZKP.$\mathcal{P}$, ZKP.$\mathcal{V}$ *to represent the algorithms of the backend ZKP protocol.*

- pp ← zkDT.$\mathcal{G}(1^\lambda)$*: let* pp_1 ← ADT.$\mathcal{G}(1^\lambda)$, pp_2 ← ZKP.$\mathcal{G}(1^\lambda)$ *and* pp = (pp_1, pp_2).
- $\mathsf{com}_{\mathcal{T}}$←zkDT.Commit($\mathcal{T}$, pp, r)*:* $\mathsf{com}_{\mathcal{T}}$←ADT.Commit($\mathcal{T}$, pp_1, r), *where r is the randomness generated by $\mathcal{P}$.*
- ($y_{\boldsymbol{a}}$, π) ← zkDT.$\mathcal{P}$($\mathcal{T}$, $\boldsymbol{a}$, pp)*:*
  (1) *$\mathcal{P}$ runs the algorithm 1 with input $\mathcal{T}$ and $\boldsymbol{a}$ to get $y_{\boldsymbol{a}} = \mathcal{T}(\boldsymbol{a})$. Then generates the witness $w = (\bar{\boldsymbol{a}}, path_{\boldsymbol{a}}$,* aux, r) *for the circuit C in accordance with the procedure of the decision tree algorithm.* aux *represents the extended witness in Figure 2. Let* com_w ← ZKP.Commit(w, pp_2). *$\mathcal{P}$ sends* com_w *and $y_{\boldsymbol{a}}$ to $\mathcal{V}$.*
  (2) *After receiving the randomness $\mathbf{r}'$ for checking the permutation of $\boldsymbol{a}$ and $\bar{\boldsymbol{a}}$ from $\mathcal{V}$, $\mathcal{P}$ invokes* ZKP.$\mathcal{P}$(C, (com_T, $\boldsymbol{a}$, $y_{\boldsymbol{a}}$, $\mathbf{r}'$), w, pp_2) *to get π. Sends π to $\mathcal{V}$.*
- {0, 1} ← zkDT.$\mathcal{V}$($\mathsf{com}_{\mathcal{T}}$, h, $\boldsymbol{a}$, $y_{\boldsymbol{a}}$, π, pp)*: $\mathcal{V}$ outputs 1 if* ZKP.$\mathcal{V}$(C, (com_T, $\boldsymbol{a}$, $y_{\boldsymbol{a}}$, $\mathbf{r}'$), π, com_w, pp_2) = 1*, otherwise it outputs 0.*

[Simulator for Protocol 1] Let λ be the security parameter, $\mathbb{F}$ be a prime field, $\mathcal{T}$ be a decision with h levels, C be the arithmetic circuit in Figure 2. (pp_1, pp_2) ← zkDT.$\mathcal{G}(1^\lambda)$.

- com ← $\mathcal{S}_1(1^\lambda$, pp, h): $\mathcal{S}_1$ invokes $\mathcal{S}_{\mathsf{ADT}}$ to generate com = $\mathcal{S}_{\mathsf{ADT}}$($\mathsf{pp}_1$, h, r), where r is the randomness generated by $\mathcal{S}_{\mathsf{ADT}}$.
- ($y_{\mathbf{a}}$, π) ← $\mathcal{S}_2^{\mathcal{A}}$($h$, $\mathbf{a}$, pp):
  (1) $\mathcal{S}_2$ asks the oracle of $\mathcal{T}$ to get $y_{\mathbf{a}} = \mathcal{T}(\mathbf{a})$. Then $\mathcal{S}_2$ shares all public input of C to $\mathcal{S}_{\mathsf{ZKP}}$ and invokes $\mathcal{S}_{\mathsf{ZKP}}$.Commit($\mathsf{pp}_2$) to get com_w.
  (2) After receiving the randomness $\mathbf{r}'$ for the permutation check from $\mathcal{A}$, it invokes $\mathcal{S}_{\mathsf{ZKP}}$.$\mathcal{P}$($C$, (com, $\mathbf{a}$, $y_{\mathbf{a}}$, $\mathbf{r}'$), pp_2) to get π. Then S_2 sends π to $\mathcal{A}$.
- {0, 1} ← $\mathcal{A}$(com, h, $\mathbf{a}$, $y_{\mathbf{a}}$, π, pp): wait $\mathcal{A}$ for validation.

for the commitment phase, it invokes $\mathcal{S}_{\text{ZKP}}$ to simulate the interactive proof phase.

Hybrid H_2: H_2 behaves in exactly the same way as the simulator of Protocol 1.

Given the same commitment $\text{com}_{\mathcal{T}}$, the verifier cannot distinguish H_0 and H_1, because of the zero knowledge property of the backend ZKP protocol given the same circuit C and the same public input. If the verifier could distinguish H_1 from H_2, then we can find a PPT adversary to distinguish whether a commitment is of an empty decision tree with only zero strings or not. This is contradictory to the hiding property of our ADT scheme. Therefore, the verifier cannot distinguish H_0 from H_2 by the hybrid, which completes the proof of zero knowledge. □

Efficiency. The prover's computation consists of two parts: committing to the decision tree $\mathcal{T}$ and generating the proof for circuit C. For the committing phase, the provers needs to do $O(N)$ hashes. For the proof generation phase, the total computation is $O(|C|\log|C|) = O((d+h)\log(d+h)) = O(d\log d)$ in accordance with Theorem 2.2 and $d > h$. The verifier's computation only contains the verification for the circuit C with size $O(d+h)$, so it is $O(|C|) = O(d)$ due to Theorem 2.2. The proof size is one digest plus the Aurora proof for the circuit C, which is only $O(\log^2|C|) = O(\log^2 d)$. Finally, we can apply the Fiat-Shamir heuristic [23] to remove the interactions in our zkDT protocol in the random oracle model.

4 ZERO KNOWLEDGE DECISION TREE ACCURACY

In this section, we present our scheme for zero knowledge decision tree accuracy. As motivated in the introduction, in this scenario, the prover owns and commits to a decision tree, receives a testing dataset from the verifier, and then proves the accurary of the committed decision tree model on this dataset. The verifier learns nothing about the model except its accuracy.

Formally speaking, similar to Section 3.2, suppose the decision tree model is $\mathcal{T}$ with h levels and N nodes, where h and N are known to both parties, the testing dataset is $\mathcal{D} = \{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n\}$ with n data samples and their matching labels are $\mathcal{L} = \{\ell_1, \ell_2, \cdots, \ell_n\}$. A zero knowledge decision tree accuracy (zkDTA) scheme consists of the following algorithms:

- $\text{pp} \leftarrow \text{zkDTA}.\mathcal{G}(1^\lambda)$: given the security parameter, generate the public parameter pp.
- $\text{com}_{\mathcal{T}} \leftarrow \text{zkDTA.Commit}(\mathcal{T}, \text{pp}, r)$: commit the decision tree $\mathcal{T}$ with a random point r generated by the prover.
- $(\text{accu}, \pi) \leftarrow \text{zkDTA}.\mathcal{P}(\mathcal{T}, \mathcal{D}, \mathcal{L}, \text{pp})$: given a dataset $\mathcal{D}$ and their labels $\mathcal{L}$, run the decision tree algorithm for each data sample in $\mathcal{D}$, compare the predictions with $\mathcal{L}$, and then output accu denoting the accuracy of the decision tree, i.e., the total number of correct predictions. The algorithm also generates the corresponding proof π.
- $\{0,1\} \leftarrow \text{zkDTA}.\mathcal{V}(\text{com}_{\mathcal{T}}, h, N, \mathcal{D}, \mathcal{L}, \text{accu}, \pi, \text{pp})$: validate the number of correct predictions, accu, given π obtained from the prover.

Definition 4.1. We say that a scheme is a zero knowledge decision tree accuracy if the following holds:

- **Completeness.** For any decision tree $\mathcal{T}$ with h levels and N nodes, a test dataset $\mathcal{D}$ with corresponding labels $\mathcal{L}$, $\text{com}_{\mathcal{T}} \leftarrow \text{zkDTA.Commit}(\mathcal{T}, \text{pp}, r)$, $(\text{accu}, \pi) \leftarrow \text{zkDTA}.\mathcal{P}(\mathcal{T}, \mathcal{D}, \mathcal{L}, \text{pp})$, it holds that
$$\Pr\left[\text{zkDTA}.\mathcal{V}(\text{com}_{\mathcal{T}}, h, N, \mathcal{D}, \mathcal{L}, \text{accu}, \pi, \text{pp}) = 1\right] = 1$$
- **Soundness.** For any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:
$$\Pr\begin{bmatrix} \text{pp} \leftarrow \text{zkDTA}.\mathcal{G}(1^\lambda) \\ (\mathcal{T}^*, \text{com}_{\mathcal{T}^*}, \mathcal{D}, \mathcal{L}, \text{accu}^*, \pi^*) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, \text{r}) \\ \text{com}_{\mathcal{T}^*} = \text{zkDTA.Commit}(\mathcal{T}^*, \text{pp}, r) \\ \text{zkDTA}.\mathcal{V}(\text{com}_{\mathcal{T}^*}, h, N, \mathcal{D}, \mathcal{L}, \text{accu}^*, \pi^*, \text{pp}) = 1 \\ \sum_{i=1}^{n} I(\mathcal{T}^*(\mathbf{a}_i) = \ell_i) \neq \text{accu}^* \end{bmatrix}$$
$I(\mathcal{T}^*(\mathbf{a}_i) = \ell_i) = 1$ if $\mathcal{T}^*(\mathbf{a}_i) = \ell_i$, otherwise $I(\mathcal{T}^*(\mathbf{a}_i) = \ell_i) = 0$.
- **Zero Knowledge.** For security parameter λ, $\text{pp} \leftarrow \text{zkDT}.\mathcal{G}(1^\lambda)$, for any decision tree $\mathcal{T}$ with h levels and N nodes, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:

$\text{Real}_{\mathcal{A},\mathcal{T}}(\text{pp})$:
(1) $\text{com}_{\mathcal{T}} \leftarrow \text{zkDTA.Commit}(\mathcal{T}, \text{pp}, r)$
(2) $\mathcal{D}, \mathcal{L} \leftarrow \mathcal{A}(h, N, \text{com}_{\mathcal{T}}, \text{pp})$
(3) $(\text{accu}, \pi) \leftarrow \text{zkDTA}.\mathcal{P}(\mathcal{T}, \mathcal{D}, \mathcal{L}, \text{pp})$
(4) $b \leftarrow \mathcal{A}(\text{com}_{\mathcal{T}}, h, N, \mathcal{D}, \mathcal{L}, \text{accu}, \pi, \text{pp})$
(5) Output b

$\text{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\text{pp}, h, N)$:
(1) $\text{com} \leftarrow \mathcal{S}_1(1^\lambda, \text{pp}, h, N)$
(2) $\mathcal{D}, \mathcal{L} \leftarrow \mathcal{A}(h, N, \text{com}, \text{pp})$
(3) $(\text{accu}, \pi) \leftarrow \mathcal{S}_2^{\mathcal{A}}(\text{com}, h, N, \mathcal{D}, \mathcal{L}, \text{pp})$, given oracle access to $\text{accu} = \sum_{i=1}^{n} \text{I}(\mathcal{T}(\mathbf{a}_i) = \ell_i)$.
(4) $b \leftarrow \mathcal{A}(\text{com}, h, N, \mathcal{D}, \mathcal{L}, \text{accu}, \pi, \text{pp})$
(5) Output b

For any PPT algorithm $\mathcal{A}$ and all decision tree $\mathcal{T}$ with the height of h and N nodes, there exists simulator $\mathcal{S}$ such that
$$|\Pr[\text{Real}_{\mathcal{A},\mathcal{T}}(\text{pp}) = 1] - \Pr[\text{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\text{pp}, h, N) = 1]| \leq \text{negl}(\lambda).$$

4.1 Checking Multiset

Intuitively, when designing the zkDTA scheme, one can repeat the construction in Section 2 described in Protocol 1 and Figure 2 multiple times for every data sample in the testing dataset, followed by an aggregation circuit testing the accuracy. The prover time in this case grows roughly linearly with the size of the testing dataset. However, the prediction paths share many common nodes and their total size may exceed the number of nodes in the decision tree on a large testing dataset. I.e., $N < nh$.

We introduce an optimization for this case. Instead of validating each prediction path one by one, the idea is to validate all N nodes of the decision tree in one shot. Then the circuit checks that the nodes of each prediction path are drawn from these N nodes of the decision tree and they form a path with the correct parent-children relationships. On top of these checks, the circuit tests the correctness of Algorithm 1 in the same way as the zkDT scheme and computes the accuracy of the model. To test that each node in the prediction paths is included in the original decision tree, it suffices to check that all nodes of the prediction paths form a *multiset* of the set of N nodes of the decision tree. We can again validate such a multiset relationship using the characteristic polynomials.

Multiset check. Suppose $Q = (q_1, q_2, \cdots, q_m)$ is an array of m elements with possible duplicates, and $S = \{s_1, \cdots, s_n\}$ is a set of size n. The prover needs to show that Q is a multiset of S, i.e., $\forall i \in [m], q_i \in S$. We apply the technique proposed in [45]. The characteristic polynomial of a multiset Q is defined as $\Pi_{i=1}^{m}(x - q_i)$ with possible duplicated elements in Q. It can also be computed as $\Pi_{i=1}^{n}(x-s_i)^{f_i}$, where f_i is the multiplicity of each element s_i in array Q. Therefore, to check the multiset relationship, the prover provides the multiplicity f_i, the verifier picks a random number r and the circuit tests $\Pi_{i=1}^{m}(r-q_i) = \Pi_{i=1}^{n}(r-s_i)^{f_i}$. We call polynomial $g(x) = \Pi_{i=1}^{m}(x-q_i) - \Pi_{i=1}^{n}(x-s_i)^{f_i}$ the multiset polynomial. The soundness of the test also follows the Schwartz-Zippel Lemma [35, 47]. If there exists $q_i \notin S$, then the multiset polynomial $g(x)$ is a non-zero polynomial with degree at most m, if we additionally force $\sum_{i=1}^{n} f_i = m$. Then $\Pr[g(r) = 0 | r \xleftarrow{\$} \mathbb{F}] \le \frac{m}{|\mathbb{F}|}$ by the Schwartz-Zippel Lemma. So the soundness error is at most $\frac{m}{|\mathbb{F}|} = \mathsf{negl}(\lambda)$.

To implement this test in an arithmetic circuit, as there is no exponentiation gate, we ask the prover to provide f_i in binary. Then the circuit checks that the inputs are indeed binary, and uses the multiplication tree to compute the exponentiation. As $\sum_{i=1}^{n} f_i = m$ and $f_i \le m$, the prover only provides $\log m$ bits for each f_i.

With the multiset check, in our zkDTA scheme, the prover provides all N nodes in $\mathcal{T}$ and proves that all the nodes in the prediction paths form a multiset of the N nodes. In addition, the circuit reconstructs the ADT using the N nodes of $\mathcal{T}$ and checks that it is consistent with $\mathsf{com}_{\mathcal{T}}$. In this way, the total number of hashes is bounded by N if $N = 2^h$. The number of gates for the multiset check is at most $O(nh + N\log(n))$. In the real implementation, computing hashes is usually the bottleneck of the efficiency. For example, SHA-2 takes around 270,000 multiplication gates and algebraic hash functions[5] take hundreds to thousands of gates to implement. Our optimization reduces the number of hashes from $O(nh)$ to $O(N)$, which greatly improves the performance of the ZKDTA scheme in practice.

Furthermore, the method of the multiset check can also be used in our zkDT protocol in Protocol 1 to support decision trees with repeated attributes on the prediction paths, which is very common in practice. We instead test that $\bar{\mathbf{a}}$ is a multiset rather than a permutation of $\mathbf{a}$ in this case.

4.2 Validating Decision Tree

In the previous optimization, the circuit checks that N nodes provided by the prover form a valid decision tree. Using ADT, we can validate it by reconstructing the decision tree in the circuit 2^h hashes. However, in practice, we notice that most decision trees are not balanced. For example, in our largest decision tree model, there are total 23 levels and 1029 nodes. In this case, $N \ll 2^h$. Therefore, in this section, we present an approach to validate a decision tree with a circuit of size linear to N rather than 2^h.

Algorithm 2 Linear Check for Valid Decision Tree

```
Input: N nodes of $T_1, T_2, \cdots, T_N$.
 1: $T_1$ is the root: $T_1$.id = 1 ∧ $T_1$.depth = 1 ∧ $T_1$.pid = 0.
 2: for $i = 1$ to $N - 1$ do
 3:     $T_{i+1}$.id = $T_i$.id + 1
 4: for $i = 1$ to $N - 1$ do
 5:     $T_{i+1}$.depth = $T_i$.depth ∨ $T_{i+1}$.depth = $T_i$.depth + 1
 6: $T_N$.depth ≤ $h$.
 7: Check all pids except $T_1$ of $\{T_2.\text{pid}, \cdots, T_N.\text{pid}\}$ are a multiset
    of $\{1, 2, 3, \cdots, N\}$ with individual multiplicity at most 2.
 8: Define two vectors of tuples, $S_1$ and $S_2$.
 9: for $i = 1$ to $N$ do
10:     if $T_i$.lid ≠ 0 then
11:         $S_1 = S_1$.append(($T_i$.depth, $T_i$.id, $T_i$.lid))
12:     if $T_i$.rid ≠ 0 then
13:         $S_1 = S_1$.append(($T_i$.depth, $T_i$.id, $T_i$.rid))
14:     if $T_i$.pid ≠ 0 then
15:         $S_2 = S_2$.append(($T_i$.depth − 1, $T_i$.pid, $T_i$.id))
16: Check $S_1$ is a permutation of $S_2$.
17: return 1 if all check pass.
```

We first replace the commitment by a hash of all N nodes concatenated by a random value r, instead of the root of the ADT. In addition, each node contains a unique id(id) in $[N]$, the id of its parent (pid), left child (lid), right child (rid) and its depth (depth) in $[h]$ (the id is 0 means the parent or the child is empty). To verify that all N nodes of $T_1, T_2, \cdots, T_N$ form a binary decision tree, it suffices to check the following conditions:

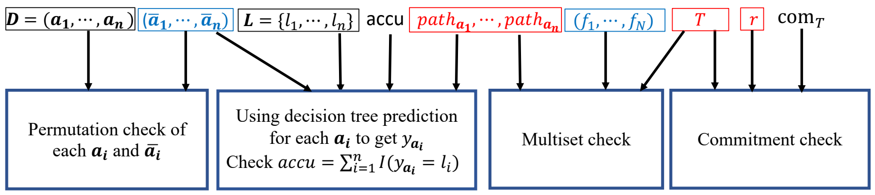
- Only the first node is the root. (i.e, $T_1.\text{pid} = 0$ but $T_i.\text{pid} \neq 0$ when $i \neq 1$.)
- All parent pointers are consistent with the child pointers. (i.e., $T_i.\text{rid} = T_j.\text{id}$ or $T_i.\text{lid} = T_j.\text{id}$ if and only if $T_j.\text{pid} = T_i.\text{id}$.)
- The depth of the parent is smaller than the depth of the child. (i.e., if $T_i.\text{pid} = T_j.\text{id}$ then $T_i.\text{depth} = T_j.\text{depth} + 1$.)
- No repeated child pointers. (i.e., if $T_i.\text{lid} \neq 0$ then $T_i.\text{lid} \neq T_i.\text{rid}$.)

With this idea in the mind, the formal algorithm is presented in Algorithm 2. The main difference is that the checks in Algorithm 2 can be efficiently implemented by arithmetic circuits.

Theorem 4.2. *Algorithm 2 outputs 1 if and only if the input nodes form a valid binary tree except for negligible probability. The total number of arithmetic gates to implement the algorithm is $O(N)$.*

Proof. On the one hand, if the N nodes of $T_1, T_2, \cdots, T_N$ construct a valid binary tree, it is easy to check they will pass all checks and Algorithm 2 always outputs 1.

On the other hand, suppose Algorithm 2 outputs 1. With overwhelming probability, (1) $T_1.\text{depth} = 1$ and $T_1.\text{pid} = 0$; (2) $T_i.\text{id} = i$ for all i; (3) all depths is a multiset of $\{1, 2, \cdots, h\}$; (4) all pids except $T_1.pid$ are in $[N]$; (5) if $T_i.\text{rid} = T_j.\text{id}$ or $T_i.\text{lid} = T_j.\text{id}$ then $T_j.\text{pid} = T_i.\text{id}$ and $T_j.\text{depth} = T_i.\text{depth} + 1$; (6) if $T_i.\text{pid} = T_j.\text{id}$ then $T_j.\text{lid} = T_i.\text{id}$ or $T_j.\text{rid} = T_i.\text{id}$ and $T_j.\text{depth} = T_i.\text{depth} - 1$; (7) if $T_i.lid \neq T_i.rid$ unless $T_i.lid = T_i.rid = 0$. If one of (1), (2), (3), (4) does not hold, it will not pass the checks from line 1 to line 7. If (5) does not hold but S_1 is a permutation of S_2, we know that $(T_i.\text{depth}, T_i.\text{id}, T_i.\text{lid}) = (T_i.\text{depth}, T_i.\text{id}, T_j.\text{id}) \in S_1$ or



$D = (a_1, \cdots, a_n)$
$(\bar{a}_1, \cdots, \bar{a}_n)$
$L = \{l_1, \cdots, l_n\}$
accu
$path_{a_1}, \cdots, path_{a_n}$
$(f_1, \cdots, f_N)$
T
r
com_T
Permutation check of each a_i and $\bar{a}_i$
Using decision tree prediction for each a_i to get y_{a_i}
Check $accu = \sum_{i=1}^{n} I(y_{a_i} = l_i)$
Multiset check
Commitment check

Figure 4: Zero knowledge decision tree accuracy.

Protocol 2 (Zero Knowledge Decision Tree Accuracy). *Let λ be the security parameter, $\mathbb{F}$ be a prime field, $\mathcal{T}$ be a binary decision with h levels and N nodes, C_A be the arithmetic circuit for model accuracy test, $\mathcal{D} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_n\}$ of size n be the test set, $\mathcal{L} = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the corresponding labels of test data. Let $\mathcal{P}_A$ and $\mathcal{V}_A$ be the prover and the verifier respectively.* ZKP.$\mathcal{G}$, ZKP.Commit, ZKP.$\mathcal{P}$, ZKP.$\mathcal{V}$ *represent the algorithms of the backend ZKP protocol.*

- pp $\leftarrow$ zkDTA.$\mathcal{G}(1^\lambda)$: pp $\leftarrow$ ZKP.$\mathcal{G}(1^\lambda)$.
- com$_\mathcal{T}$ $\leftarrow$ zkDTA.Commit($\mathcal{T}$, pp, r): com$_\mathcal{T}$ = Hash($\mathcal{T}$, r)*, it hashes all nodes in $\mathcal{T}$ with r, the randomness generated by $\mathcal{P}_A$.*
- (accu, π) $\leftarrow$ zkDTA.$\mathcal{P}$($\mathcal{T}$, $\mathcal{D}$, $\mathcal{L}$, pp)*: we could use Fiat-Shamir heuristic transformation to make the following process non-interactive.*
  (1) $\mathcal{P}_A$ runs the algorithm 1 with input $\mathcal{T}$ and the data set $\mathcal{D}$ to get $y_{\boldsymbol{a}_i} = \mathcal{T}(\boldsymbol{a}_i)$ for all i. Let accu $= \sum_{i=1}^{n} \mathrm{I}(y_{a_i} = \ell_i)$*. According the procedure of the algorithm, the prover could generate the witness $\overline{\boldsymbol{a}_i}$ and $path_{\boldsymbol{a}_i}$ for each data point $\boldsymbol{a}_i$, together with all nodes in $\mathcal{T}$, their multiplicity in $\{path_{\boldsymbol{a}_i}\}_{i=1}^n$ and the randomness r used in* com$_\mathcal{T}$ *as the witness w_A of the circuit C_A. Let* com$_{w_A}$ $\leftarrow$ ZKP.Commit(w_A, pp$_2$)*. $\mathcal{P}_A$ sends* com$_{w_A}$ *and* accu *to $\mathcal{V}_A$.*
  (2) After receiving the randomness $\boldsymbol{r}'$ for checking characteristic polynomials and the multiset polynomial from $\mathcal{V}_A$. $\mathcal{P}_A$ invokes ZKP.$\mathcal{P}$(C_A, (com$_T$, $\mathcal{D}$, $\mathcal{L}$, accu, $\boldsymbol{r}'$), w$_\mathrm{A}$, pp$_2$) *to get π. Send π to $\mathcal{V}_A$.*
- $\{1, 0\}$ $\leftarrow$ zkDTA.$\mathcal{V}$(com$_\mathcal{T}$, h, N, $\mathcal{D}$, $\mathcal{L}$, accu, π, pp)*: $\mathcal{V}_A$ outputs 1 if* ZKP.$\mathcal{V}$(C_A, (com$_T$, $\{\boldsymbol{a}\}_{i=1}^n$, accu, $\boldsymbol{r}'$), π, com$_{\mathrm{w_A}}$, pp) = 1*, outputs* 0 *otherwise.*

$(T_i.\mathrm{depth}, T_i.\mathrm{id}, T_i.\mathrm{rid}) = (T_i.\mathrm{depth}, T_i.\mathrm{id}, T_j.\mathrm{id}) \in S_1$. There must exist an index $k \in [N]$ such that $(T_k.\mathrm{depth} - 1, T_k.\mathrm{pid}, T_k.\mathrm{id}) = (T_i.\mathrm{depth}, T_i.\mathrm{id}, T_j.\mathrm{id})$. Then we have $k = j$, $T_j.\mathrm{pid} = T_i.\mathrm{id}$ and $T_i.\mathrm{depth} = T_j.\mathrm{depth} - 1$, which is contradictory. Therefore, if (5) does not hold, S_1 is not a permutation of S_2. With very similar argument, if (6) does not hold, S_1 is not a permutation of S_2. If (7) does not hold, S_1 is not a permutation of S_2 as S_1 has duplicate element while S_2 does not. Therefore, if (5) or (6) or (7) does not hold, then it will not pass the check in line 16 with high probability.

When (1)-(7) hold, we can construct a directed graph $G = (T_{[N]}, E)$ as follows. $(T_i, T_j) \in E$ if and only if $T_i.\mathrm{pid} = T_j.\mathrm{id}$. The outdegree of T_1 is 0 while all others are 1. To prove G is a tree, we only need to show T_i connects to T_1 for each i, or equivalently, there is no circle in the graph. That is because $T_i.\mathrm{depth} = T_j.\mathrm{depth} - 1$ if $(T_i, T_j) \in E$. So G does not have a circle and it must be a tree. It is a binary tree as the indegree of each nodes are at most 2.

Complexity. Step 1-6 and Step 8-15 can be computed by a circuit doing a linear scan of all the N nodes. In addition,as the the individual multiplicity is at most 2, both the multiset test in Step 7 and the permutation test in Step 16 are $O(N)$ as we explained in previous sections. Furthermore, we force the N nodes sorted by the id and the depth. It consumes only $O(N)$ to check $T_i.\mathrm{id} = i$ and the depth ranging in $[h]$. So the total number of the gates is $O(N)$. □

An alternative approach. Alternatively, because of our application of decision trees, we can simplify the checks above. Recall that in other parts of the circuit, it is proven that the nodes of the prediction paths are drawn from the set of N nodes, and each prediction path follows the prediction function of a decision tree. Therefore, the graph formed by these nodes already satisfies the second and the third conditions of being a binary decision tree. Because of this, we simplify the test to (1) the N nodes are sorted by the id, $T_i.\mathrm{id} = i$, where T_1 is the root and (2) $T_i.lid \neq T_i.rid$ unless they are empty for all $i \in [N]$. These checks ensure that the subgraph formed by all nodes in the prediction paths is part of a binary decision tree. These nodes are also included in the N nodes because of the multiset check. There is no guarantee on other nodes of the N nodes provided by the prover, but they are not used in the prediction paths anyway, which is good enough for our purposes.

This alternative approach is simpler than the check described in Algorithm 2. It is a trade-off between the efficiency and the security. In practice, as computing hashes is the bottleneck of our system, the difference between these two approaches on the prover time is actually not significant. We use the latter in our implementation.

4.3 Our construction of zkDT accuracy

With the optimizations presented in the previous sections, we show the circuit C_A to validate decision tree accuracy in Figure 4, and present the formal protocol of zkDTA in Protocol 2. Compared with the circuit C in Figure 2 for the zkDT scheme, C_A has extra extended witness with all nodes of $\mathcal{T}$ and an auxiliary list $F = (f_1, f_2, \cdots, f_N)$ representing the multiplicity of N nodes appearing in all prediction paths of the test data. C_A also has one more part for multiset check. Besides, C_A does not need to do path validation for each path, it only recomputes the hash of all nodes in $\mathcal{T}$ with r and compares the final value with the commitment. We call it commitment check. Furthermore, in this new scheme, all $y_{\boldsymbol{a}_i}$ are not public to the verifier. The circuit compares $y_{\boldsymbol{a}_i}$ with ℓ_i and computes the number of correct predictions. Finally, the circuit compares the number of correct predictions to the claimed accuracy. We have the following theorem:

Theorem 4.3. *Protocol 2 is a zero knowledge decision tree accuracy scheme defined by Definition 4.1.*

The completeness, soundness and zero-knowledgeness of Protocol 2 are extensions of these properties in Protocol 1. We omit the proof because of the space limitation.

Efficiency. Consider the circuit C_A, the circuit size is $O(nh + nd + N \log n + N) = O(nd)$ when $N \ll nd$. Therefore, the prover time is $O(nd\log(nd))$ with $O(N)$ hashes in the committing phase, the verification time is $O(nd)$ and proof size is $O(\log^2(nd))$ according to Theorem 2.2.

5 IMPLEMENTATION AND EVALUATIONS

We fully implement our zero knowledge decision tree schemes and we present their performance in this section.

Software. The schemes are implemented in C++. There are around 2000 lines of code for our frontend to compile the decision tree predictions and accuracy to arithmetic circuits[1], as shown in Figure 2 and 4. We use the open-source compiler of libsnark[1] to generate arithmetic circuits in our frontend, and we implement the zero knowledge argument scheme Aurora [10] ourselves as the ZKP backend. We use the extension field of a Mersenne prime $\mathbb{F}_{p^2}$, where $p = 2^{61} - 1$. This is the same as the field used in [42].

We download the datasets from the UCI repository datasets[22] and train the decision tree models using the sklearn package in Python. Then we use these pre-trained decision trees and the testing datasets in our experiments. The attributes are scaled to 32-bit integers in the field for our ZKP backend. As the attributes are only used for comparisons, the scaling does not affect the prediction and the accuracy of the decision trees.

Hardware. We run all of the experiments on Amazon EC2 c5n.2xlarge instances with 64GB of RAM and Intel Xeon platinum 8124m CPU with 3GHz virtual core. Our current implementation is not parallelized and we only use a single CPU core in the experiments. We report the average running time of 10 executions.

Hash function. As we will show in the experiments, computing hashes in the arithmetic circuits is the bottleneck of our system. For example, it takes 27,000 multiplication gates to compute one SHA-256 hash. Therefore, in order to improve the performance of our system, we use the SWIFFT[31] hash function, which is an algebraic hash function that is collision-resistant and is friendly to arithmetic circuits. With the optimizations proposed in jsnark[2], one SWIFFT hash can be implemented with around 3,000 multiplication gates.

Datasets. We use three datasets from the UCI machine learning repository [22]. The small dataset we use is named Breast-Cancer-Wisconsin(Original). It is used for breast cancer diagnosis. Each data has 10 attributes and the prediction is either 0 or 1. We train the model on a training set of 600 data points. The pre-trained decision tree has 61 nodes and 10 levels. The second dataset we use is Spambase. It is used to recognize the spam emails. Each data has 57 attributes and the prediction is either 1 or 0. We train the model on training set of 4,000 data points. The pretrained decision tree has in total 441 nodes and 26 levels. The largest dataset is Forest Covertype. It is used to predict forest cover type from cartographic variables. Each data has 54 attributes and the total number of class is 7. We train the model on a training set of 5,000 data points. The pre-trained decision tree has in total 1,029 nodes and 23 levels.

5.1 Performance of ZKDT

We first present the performance of our zero knowledge decision tree prediction protocol. We vary the length of the prediction paths from 6 to 48. The prediction paths of the first 3 columns are obtained from the decision trees of the three real-world datasets described above, while the last one is obtained from synthetic data. Table 1 shows the performance of our scheme.

Length h	6	12	24	48
#Attributes d	10	54	57	1000
Commit Time (ms)	0.38	6.3	2.8	13.3
Prover Time (s)	0.754	1.577	3.433	7.024
Verifier Time (s)	0.050	0.104	0.221	0.445
Proof size (KB)	140.736	155.936	172.224	189.632

Table 1: Performance of zero knowledge decision tree predictions.

As we can see in the Table 1, the efficiency of our zkDT scheme is reasonable in practice. Though linear to the size of the whole decision trees, the time to commit the decision trees is only on the order of milliseconds. This is because the Commit in our ADT only involves computing hashes, which is very fast in practice. For the prover time and the verification time, it takes 7.02 seconds to generate the proof for a prediction path of length 48, and 0.445 seconds to validate the proof. The proof size is 189KB. Note that we choose our ZKP backend of Aurora [10] to optimize for the prover time. Our zkDT scheme works on all backends and we could use schemes such as Bulletproof [18] and SNARK [34] to reduce the proof size to several KBs or less, with a sacrifice on the prover time.

Moreover, the prover time and the verification time scale roughly linearly with the length of the prediction paths, while the number of attributes d does not affect the performance by much. This is because the bottleneck of the efficiency is computing the hashes in the circuit of the ZKP backend for the path validation, and the number of hashes is linear to the length of the path. Besides, the performance of our scheme fully depends on the parameters of the decision trees and the size of the data samples, but not on the values. Hence, we do not observe any major difference on the performance between the real datasets and the synthetic dataset.

In Appendix B, we compare the performance of our zkDT scheme with the baseline of using RAM-based and circuit-based generic ZKP schemes. Our scheme improves the prover time by orders of magnitude, as we reduce the decision tree prediction to a circuit of optimal size without using generic RAM-to-circuit reduction. We also extend the implementation to zero knowledge random forests trained on the same dataset. Table 2 shows the performance of random forests consisting of different number of decision trees.

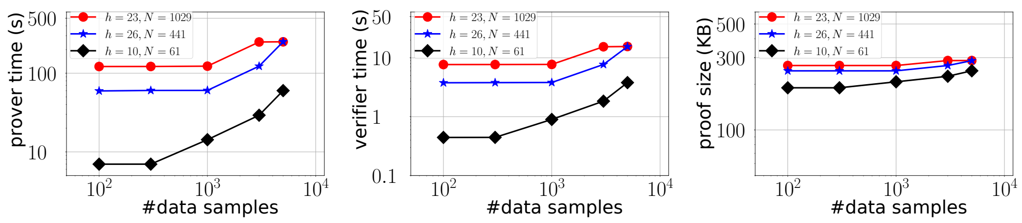
5.2 Performance of ZKDTA

In this section, we further evaluate the performance of our zero knowledge decision tree accuracy scheme. We implement Protocol 2 and test it on decision trees trained on the three datasets described

# of Trees	2	8	32	128
Length h	24	24	12	12
Commit Time (ms)	13.49	54.95	115.14	468.56
Prover Time (s)	6.992	29.317	60.532	253.41
Verifier Time (s)	0.447	1.842	3.83	15.86
Proof size (KB)	189.728	225984	245632	286592

Table 2: Performance of zero knowledge random forest predictions.

[1]We actually use the rank-1-constraint-system (R1CS) to be compatible with the backend.



(a) Prover time (b) Verification time (c) Proof size

Figure 5: Performance of the zero knowledge decision tree accuracy scheme.

before. We vary the number of data samples in the testing dataset as 100, 300, 1000, 3000 and 5,000. We present the prover time, proof size and verification time in Figure 5.

As shown in Figure 5, our zkDTA scheme achieves very good efficiency in practice. On the largest instance with a decision tree of 1029 nodes and 23 levels, it only takes 250 seconds to generate a zero knowledge proof for its accuracy on a large testing dataset of 5,000 data samples. We believe this overhead is close to practical in many applications. The verification time is around 15.6 seconds. In this case, the proof size is 287KB, which is actually smaller than the size of the decision tree and the size of the testing dataset. This means our scheme not only provides soundness and zero knowledge, but also reduces the communicating comparing to the naïve solution of posting the model and the dataset. The gap will further increase because of the succinctness of the proof size.

In addition, Figure 5 shows that the prover time and the verification time for all three models remain mostly unchanged until the size of the testing dataset becomes larger. This is because the bottleneck of our scheme is computing hashes in the circuit of the ZKP backend. The number of hashes is proportional to the total number of nodes in the decision tree, and does not depend on the size of the testing dataset. For example, for the large decision tree model with $N = 1029$ nodes, when the size of the testing dataset is less than 1000, the sub-circuit checking the commitment using hashes consists of around 2^{21} multiplication gates and contributes to more than 75% of the whole circuit. Thus, the performance of the scheme remains mostly the same for dataset with less than 1000 samples. As the size of the testing dataset becomes larger than 1000, other components of the circuit to check decision tree predictions, multisets and permutations start to impact the performance. There are 2^{23} multiplication gates in our largest data point in the figure.

The observation above also justifies the importance of our two optimizations proposed in Section 4. If we simply repeat the zero knowledge predictions multiple times, the number of hashes will increase with the size of the testing dataset. In the largest data point, there are 5,000 samples and each prediction path is of length around 20. The performance would be 100× slower compared to our optimized scheme. Similarly, as shown by the three real-world datasets, the decision trees are usually not balanced. Without our second optimization in Section 4, the number of hashes would be 2^h, which is much larger than N. For the largest decision tree with $N = 1029$ and $h = 23$, our optimization improves the prover time and proof size by around 8000×.

Finally, the accuracy of the three decision trees (from small to large) are 94.89%, 92% and 64.25% respectively on their largest testing datasets with 5,000 samples. They are exactly the same as the original decision trees trained from the datasets, as our zkDTA scheme does not use any approximations and does not introduce any accuracy loss.

5.3 Applications of Our Schemes

Besides ensuring the integrity of decision tree predictions and accuracy in the scenarios motivated in the introduction, our zero knowledge decision tree schemes can also be applied to build a fair and secure trading platform for machine learning models on blokchains. Machine learning models are valuable assets now, and people want to monetize their expertise on machine learning by selling high-quality models. In this scenario, the buyer prefers to test the quality of the machine learning model before making the payment, while the seller does not want to reveal the model first, as the buyer could disappear without any payment after seeing the model. This is the classical problem of fair exchange. One could rely on a trusted party to address this problem. However, it often introduces a heavy burden on the trusted party to validate the quality of the models, enforce the payments and resolve the disputes. Some applications may lack the existence of such a trusted party.

Blockchain is a promising technique to replace the role of the trusted party in this scenario. In a blockchain, all the users validate the data posted on the blocks such that as long as more than 50% of the users are honest, the data on the blockchain is valid and cannot be altered. *Zero knowledge contingent payments* [3, 20] provide a framework for users to trade secret data fairly and securely on blockchains. Instead of posting the data directly on the blockchain, which reveals the data to all users of the blockchain, the seller posts a short zero knowledge proof about the properties of the data. Subsequent protocols enforces the payment and the delivery of the data simultaneously using smart contracts, given that the zero knowledge proof is valid. In order to build a trading platform for machine learning models on blockchains, efficient ZKP protocols for machine learning accuracy are the only missing piece in the framework of zero knowledge contingent payments. Our zero knowledge decision tree schemes in this paper fill in this gap and can be used to build such a fair and secure trading platform. Realizing the system of the trading platform is left as future work.

ACKNOWLEDGMENTS

This material is in part based upon work supported by the National Science Foundation(NSF) under Grant No. TWC-1518899, DARPA under Grant No. N66001-15-C-4066, DARPA under Contract No. HR001120C0087 and Center for Long-Term Cybersecurity (CLTC). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA or CLTC.

REFERENCES

[1] 2014. libsnark. https://github.com/scipr-lab/libsnark.
[2] 2015. jSNARK. https://github.com/akosba/jsnark.
[3] 2016. Zero knowledge contingent payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.
[4] 2019. Human-guided burrito bots raise questions about the future of robo-delivery. https://thehustle.co/kiwibots-autonomous-food-delivery/.
[5] Miklós Ajtai. 1996. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 99–108.
[6] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.
[7] Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 263–280.
[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*. Springer, 701–732.
[9] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. [n.d.]. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*.
[10] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019*. Springer International Publishing, 103–128. https://doi.org/10.1007/978-3-030-17653-2_4
[11] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. [n.d.]. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *Proceedings of the USENIX Security Symposium, 2014*.
[12] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO 2014*. 276–294.
[13] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *International Conference on the Theory and Applications of Cryptographic Techniques*.
[14] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K Jakobsen. 2017. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 336–365.
[15] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. 2018. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 595–626.
[16] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine learning classification over encrypted data.. In *NDSS*, Vol. 4324. 4325.
[17] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. [n.d.]. Verifying computations with state. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP, 2013*.
[18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. [n.d.]. Bulletproofs: Short Proofs for Confidential Transactions and More. In *Proceedings of the Symposium on Security and Privacy (SP), 2018*, Vol. 00. 319–338.
[19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. [n.d.]. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs.. In *CCS 2019*.
[20] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 229–243.
[21] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. [n.d.]. Geppetto: Versatile Verifiable Computation. In *S&P 2015*.
[22] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[23] Amos Fiat and Adi Shamir. [n.d.]. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto 1986*.
[24] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. 2016. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.
[25] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. ACM Press. https://doi.org/10.1145/2810103.2813677
[26] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*. 4672–4681.
[27] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.
[28] Jens Groth. 2009. Linear algebra with sub-linear zero-knowledge arguments. In *Advances in Cryptology-CRYPTO 2009*. Springer, 192–208.
[29] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 305–326.
[30] Joe Kilian. 1992. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In *Proceedings of the ACM Symposium on Theory of Computing*.
[31] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. [n.d.]. SWIFFT: A Modest Proposal for FFT Hashing. In *Fast Software Encryption*. Springer Berlin Heidelberg, 54–72. https://doi.org/10.1007/978-3-540-71039-4_4
[32] Ralph C. Merkle. [n.d.]. A Certified Digital Signature. In *CRYPTO 1989*. 218–238.
[33] Silvio Micali. 2000. Computationally Sound Proofs. *SIAM J. Comput.* (2000).
[34] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *S&P 2013*. 238–252.
[35] Jacob T Schwartz. 1979. Probabilistic algorithms for verification of polynomial identities. In *International Symposium on Symbolic and Algebraic Manipulation*. Springer, 200–215.
[36] Roberto Tamassia. 2003. Authenticated data structures. In *European symposium on algorithms*. Springer, 2–5.
[37] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. 2019. Private evaluation of decision trees using sublinear cost. *Proceedings on Privacy Enhancing Technologies* 2019, 1 (2019), 266–286.
[38] Jaideep Vaidya and Chris Clifton. 2005. Privacy-preserving decision trees over vertically partitioned data. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 139–152.
[39] Riad S Wahby, Srinath TV Setty, Zuocheng Ren, Andrew J Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation.. In *NDSS*.
[40] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 926–943.
[41] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology (CRYPTO)*.
[42] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. [n.d.]. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *S&P 2020*.
[43] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 863–880.
[44] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology ePrint.
[45] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. vRAM: Faster verifiable RAM with program-independent preprocessing. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)*.
[46] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, Xiaodong Lin, Shengshan Hu, and Minxin Du. 2019. VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service. *arXiv preprint arXiv:1909.06961* (2019).
[47] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Manipulation*. Springer, 216–226.

A OTHER VARIANTS OF ZERO KNOWLEDGE DECISION TREES.

Multivariate decision trees. In univariate decision trees, each decision node checks only one attribute for axis-aligned splits. In a linear multivariate decision tree, each decision node divides the input space into two with an arbitrary hyperplane leading to slanting splits. More formally, In a linear multivariate decision tree $\mathcal{T}$, each internal node v has a size-d vector $v.\mathbf{w}$ of weights for d attributes, a threshold $v.\mathsf{thr}$ and two children $v.\mathsf{left}$ and $v.\mathsf{right}$. Similar to the univariate decision tree, each leaf node u stores the classification result $u.\mathsf{class}$. Each data sample is represented as a size-d vector $\mathbf{a}$ of values corresponding to each attribute. The algorithm of multivariate decision tree prediction is shown in Algorithm 3. It starts from the root of $\mathcal{T}$. For each node of v in $\mathcal{T}$, it compares $v.\mathbf{w} \cdot \mathbf{a}^T$

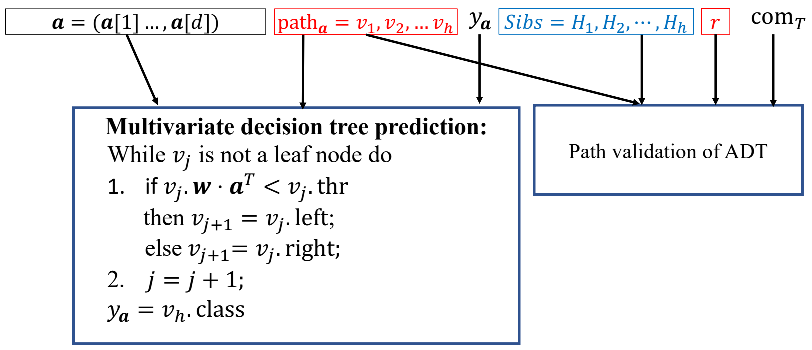

$\boldsymbol{a} = (\boldsymbol{a}[1] \dots, \boldsymbol{a}[d])$
$\text{path}_{\boldsymbol{a}} = v_1, v_2, \dots v_h$
$y_{\boldsymbol{a}}$
$Sibs = H_1, H_2, \cdots, H_h$
r
com_T
Multivariate decision tree prediction:
While v_j is not a leaf node do
1. if $v_j.\boldsymbol{w} \cdot \boldsymbol{a}^T < v_j.\text{thr}$
then $v_{j+1} = v_j.\text{left}$;
else $v_{j+1} = v_j.\text{right}$;
2. $j = j + 1$;
$y_{\boldsymbol{a}} = v_h.\text{class}$
Path validation of ADT

Figure 6: Zero knowledge multivariate decision tree prediction. Public inputs are in black, secret witness is in red, and extended witness for efficiency is in blue.

Algorithm 3 Multivariate Decision Tree Prediction

```
Input: Decision tree $\mathcal{T}$, data sample $\mathbf{a}$
Output: classification $y_{\mathbf{a}}$
1: $v := \mathcal{T}$.root
2: while $v$ is not a leaf node do
3:     if $v.\mathbf{w} \cdot \mathbf{a}^T < v$.thr then
4:         $v := v$.left
5:     else
6:         $v := v$.right
7: return $v$.class
```

with v.thr, and moves to v.left if $v.\mathbf{w} \cdot \mathbf{a}^T < v$.thr, and v.right otherwise. Eventually, the algorithm reaches a leaf node u and the result of the prediction is u.class.

Surprisingly, the hyperplane split in each node of multivariate decision trees makes the design of zero knowledge multivariate decision trees simpler. The prediction of multivariate decision trees is not a RAM based program anymore because it uses the linear combination of d attributes in each node. The circuit for zero knowledge multivariate decision tree predictions is given in Figure 6. Note that comparing to Figure 2, the circuit does not need to perform random access to the attributes using a permutation check.

Regression decision trees. Decision trees where the target variable can take continuous values (typically real numbers), e.g., the price of a house, are called regression decision trees. For regression decision tree predictions, we retain Algorithm 1 except for changing $y_{\mathbf{a}}$ to a real number. The circuit for verifying the regression decision tree prediction is also the same as in Figure 2 but $y_{\mathbf{a}}$ could be a real number.

Random forests. Random forest consists of many individual decision trees. It is an ensembling learning method that can help reduce the variance and aviod the overfitting of a single decision tree model. For the classification problem, each individual decision tree in the random forest outputs a class prediction and the class with the most votes becomes the prediction of the random forest. More formally, suppose a random forest contains m decision trees $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_m$ and $[M]$ is the set of all target classifications, for a data sample $\mathbf{a}$ with d features, we run Algarithm 1 with each decion tree and get the predictions $\mathcal{T}_1(a), \mathcal{T}_2(a), \cdots, \mathcal{T}_m(a) \in [M]$. Then the model outputs the index in $[M]$ with most occurrences in $\mathcal{T}_1(a), \mathcal{T}_2(a), \cdots, \mathcal{T}_m(a)$. The specific algorithm is given in Algorithm 4.

For constructing the circuit to verify the random forest prediction, besides verifying each decision tree's prediction, we add an argmax function on all predictions before outputting the final classification, which is easy to be implemented by the arithmetic circuit.

For the regression problem, random forests report the mean of all values output by individual decision trees, which can be also easily implemented by the arithmetic circuit. We omit the formal construction.

B COMPARISON TO GENERIC ZERO KNOWLEDGE PROOF SCHEMES

We conducted the experiments on the comparison between our zkDT scheme with the baseline of using RAM-based and circuit-based generic ZKP schemes and present the experimental results in this section.

For the RAM-based ZKP scheme, we write the decision tree predictions algorithm in TinyRAM [9], a language similar to assembly language with a simple instruction set. Each iteration in Algorithm 1 takes 20 TinyRAM instructions and there are in total h iterations. We then translate the program to a circuit using the RAM-to-circuit reduction in [11]. Each instruction takes around 4,000 multiplications gates. Finally, we estimate the prover time

Algorithm 4 Random Forest Prediction

```
Input: Random forest with $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_m$, data sample $\mathbf{a}$
Output: classification $y_{\mathbf{a}}$
1: for $i = 1$ to $M$ do
2:     $Occ[i] = 0$
3: for $i = 1$ to $m$ do
4:     Run Algorithm 1 with input of $\mathcal{T}_i$ and $\mathbf{a}$
5:     $Occ[\mathcal{T}_i(\mathbf{a})]$++
6: return $\arg\max_i Occ[i]$
```

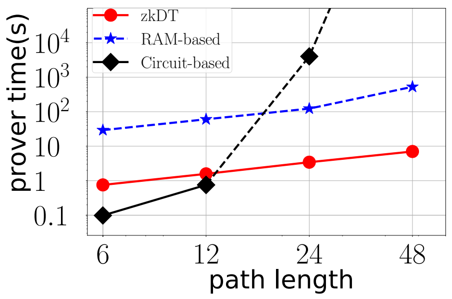


Figure 7: Comparison between zkDT, RAM-based and circuit-based generic ZKP schemes.

using the same ZKP backend of Aurora [10] on the circuit. For the naive circuit-based ZKP scheme, we hardcode the whole decision tree into the circuit and run the prediction algorithm directly using the arithmetic circuit. Then we apply the same ZKP backend.

We vary the path length of the decision tree from 6 to 48 and report the prover time in Figure 7. As shown in the figure, our zkDT scheme is above 100 times faster than the RAM-based ZKP scheme. This is because our scheme does not use the full machinery of the generic RAM-to-circuit reduction, and our resulting circuit in Figure 2 is much smaller. The circuit-based ZKP scheme is faster when the tree size is small. However, the prover time increases exponentially as the length of the path increases. This is because the naive implementation of circuits cannot handle branching efficiently and both branches are included in the circuit for decision tree predictions. The prover time soon becomes slower than our zkDT scheme for paths with more than 12 nodes and is estimated to be 2^{34} seconds for $h = 48$.