Privately Evaluating Decision Trees and Random Forests

David J. Wu[*] Tony Feng[*] Michael Naehrig[†] Kristin Lauter[†]

Abstract

Decision trees and random forests are common classifiers with widespread use. In this paper, we develop two protocols for privately evaluating decision trees and random forests. We operate in the standard two-party setting where the server holds a model (either a tree or a forest), and the client holds an input (a feature vector). At the conclusion of the protocol, the client learns only the model's output on its input and a few generic parameters concerning the model; the server learns nothing. The first protocol we develop provides security against semi-honest adversaries. Next, we show an extension of the semi-honest protocol that obtains one-sided security against malicious adversaries. We implement both protocols and show that both variants are able to process trees with several hundred decision nodes in just a few seconds and a modest amount of bandwidth. Compared to previous semi-honest protocols for private decision tree evaluation, we demonstrate tenfold improvements in computation and bandwidth.

1 Introduction

In recent years, machine learning has been successfully applied to many areas, such as spam classification, credit-risk assessment, cancer diagnosis, and more. With the transition towards cloud-based computing, this has enabled many useful services for consumers. For example, there are many companies that provide automatic medical assessments and risk profiles for various diseases by evaluating a user's responses to an online questionnaire, or by analyzing a user's DNA profile. In the personal finance area, there exist automatic tools and services that provide valuations for a user's car or property based on information the user provides. In most cases, these services require access to the user's information in the clear. Many of these situations involve potentially sensitive information, such as a user's medical or financial data. A natural question to ask is whether one can take advantage of cloud-based machine learning, and still maintain the privacy of the user's data. On the flip side, in many situations, we also require privacy for the model. For example, in scenarios where companies leverage learned models for providing product recommendations, the details of the underlying model often constitute an integral part of the company's "secret sauce," and thus, efforts are taken to guard the precise details. In other scenarios, the model might have been trained on sensitive information such as the results from a medical study or patient records from a hospital; here, directly revealing the model can not only compromise sensitive information, but also violate certain laws and regulations.

In this work, we consider one particular class of machine learning classifiers, and seek to develop efficient protocols for evaluating these classifiers in a privacy-preserving manner. Specifically, we examine decision trees, and their generalization, random forests [1, 2]. Decision trees are simple classifiers that consist of a collection of decision nodes arranged in a tree structure. As the name suggests, each decision node is associated with a predicate or test on the query (for example, a possible predicate could be "$\texttt{age} > 55$"). When the decision tree is binary, each decision node has exactly two children. To evaluate a binary decision tree, we begin at the root. Then, at each decision node, we evaluate the corresponding predicate, and depending on the outcome, proceed either to the node's left child or right child. Thus, every input induces a path through the decision tree. The output of the decision tree is simply the value of the leaf node in the

[*]Stanford University - `dwu4@cs.stanford.edu, tonyfeng@stanford.edu`. Work done while at Microsoft Research.
[†]Microsoft Research - `{mnaehrig, klauter}@microsoft.com`.

induced path. Despite their simple structure, decision trees are widely used in machine learning, and have been successfully applied to many scenarios such as heart disease diagnosis [3] and credit-risk assessment [4].

In this work, we develop a set of practical two-party protocols for privately evaluating decision trees and random forests. In our setting, the server has a trained model of a decision tree or a random forest and the client holds an input to the model. For example, in the case of heart disease diagnosis, the client's input might consist of a feature vector describing the symptoms she is currently experiencing, and the server's model would classify these symptoms as being indicative of heart disease or not. Abstractly, our desired security property is that at the end of the protocol execution, the server should not learn anything about the client's input, and the client should not learn anything about the server's model other than what can be directly inferred from the output of the model. This is a natural setting in cases where we are working with potentially sensitive and private information on the client's side and where we desire to protect the server's model, which might consist of proprietary information.

This problem of privately evaluating decision trees falls into the more general area of privacy-preserving computation. One general approach here leverages homomorphic encryption [5, 6], where the client sends to the server an encryption of its input, and the server evaluates the function homomorphically and sends the encrypted response back to the client. The client then decrypts to learn the output. While these methods have been successfully applied to several problems in privacy-preserving data mining [7, 8], the methods are typically limited to evaluating simple, low-depth functions. Another general approach is based on Yao's garbled circuits [9, 10, 11, 12, 13], where one party prepares a garbled circuit representing the joint function they want to compute and the other party evaluates the circuit. In our setting, it does not suffice to simply construct a circuit representing the decision tree and have the client and server privately compute the output using Yao's approach. This method necessarily reveals the structure of the decision tree to the client. To ensure privacy for the model, we require the stronger primitive of private function evaluation. While it is possible to apply Yao's technique with universal circuits to achieve this stronger notion of privacy, these constructions are less efficient in terms of bandwidth. In our work, we develop a specific protocol tailored for decision tree evaluation.

Our contribution. First, we describe a decision-tree evaluation protocol that is secure against semi-honest adversaries. In the semi-honest model, both the client and server follow the protocol specification, but may try to learn additional information based on their view of the protocol execution. We then show how to extend the semi-honest protocol to a one-sided secure protocol against malicious adversaries. Specifically, this means that even if the client behaves maliciously, it still cannot learn additional information about the server's model. Conversely, if the server was malicious, we still guarantee privacy for the client's input. The one-sided nature of this definition means that a malicious server can cause the client to obtain a corrupted output, but even then, is still unable to learn anything about the client's input. This model is well-suited for cloud-based applications where we assume the server is trying to provide a useful service, and thus, is not incentivized to give corrupt or nonsensical output to the client; however, even if the server acts maliciously, the privacy of the client's input is preserved. We give formal simulation-based proofs of security for both of our protocols in the real-world/ideal-world paradigm [14, 15, 16]. Our protocols leverage two standard cryptographic primitives: additively homomorphic encryption and oblivious transfer.

To assess the practicality of our protocols, we implement both the semi-honest and the one-sided secure protocols using standard libraries. We conduct experiments with decision trees with depth up to 20, as well as decision trees with over 10,000 decision nodes to assess the scalability of our protocols. We also compare the performance of our semi-honest secure protocol against the protocols of [17, 18, 19], and demonstrate over 10x reduction in client computation and bandwidth, or both, while operating at a higher security level (128 bits of security as opposed to 80 bits of security in past works). We conclude our experimental analysis by evaluating our protocols on decision trees trained on several real datasets from the UCI repository [20]. In most cases, our semi-honest decision tree protocol completes on the order of seconds and requires bandwidth ranging from under 100 KB to several MB. This represents reasonable performance for a cloud-based service.

This work also provides the first implementation of a private decision tree evaluation protocol with security against malicious adversaries. In our benchmarks, we additionally show that the performance of our one-sided secure protocol still outperforms existing protocols that only achieve semi-honest security.

2 Preliminaries

We begin by introducing some notation. Let $[n]$ be the set of positive integers $\{1, \ldots, n\}$, and $\mathbb{Z}_p$ be the ring of integers modulo p. For two k-bit strings $x, y \in \{0,1\}^k$, we write $x \oplus y$ for their bitwise xor. Let $\mathcal{D}$ be a probability distribution. We write $x \xleftarrow{\$} \mathcal{D}$ to denote that x is drawn uniformly at random from $\mathcal{D}$. We say that two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are computationally indistinguishable if no probabilistic polynomial time (PPT) algorithm can distinguish them except with negligible probability. A function f is negligible in a parameter λ if for all positive integers c, $f = o(1/\lambda^c)$.

Let $\mathcal{P}$ be a predicate. We write $\mathbf{1}\{\mathcal{P}(x)\}$ to denote the indicator function for the predicate $\mathcal{P}$, that is, $\mathbf{1}\{\mathcal{P}(x)\} = 1$ if and only if $\mathcal{P}(x)$ holds, and 0 otherwise.

2.1 Security Model

Our security definitions follow the real-world/ideal-world paradigm of [14, 15, 16, 21]. We define a two-party protocol π to be a possibly randomized process that maps a pair of inputs to a pair of outputs. We refer to the first party $\mathcal{P}_1$ as the client and the second party $\mathcal{P}_2$ as the server. We define a functionality to be the function f computed by the protocol π. If we let x_i denote the input of $\mathcal{P}_i$ and f_i denote its output, then we can express the functionality as the mapping $(x_1, x_2) \mapsto (f_1(x_1, x_2), f_2(x_1, x_2))$. For private decision tree evaluation, the desired functionality can be expressed as $(x, \mathcal{T}) \mapsto (\mathcal{T}(x), -)$, where x is the client's feature vector and $\mathcal{T}$ is the server's model. We write '–' to indicate that the server receives no output. Informally, our security notion compares a real-world execution of the protocol π against an ideal world instantiation where the parties have access to a trusted party who evaluates f. We say that security holds if these two executions are computationally indistinguishable. See Appendix A for a more formal treatment.

In our work, we consider two classes of adversaries: semi-honest adversaries and malicious adversaries. A semi-honest adversary follows the protocol specification as directed, but may try to learn additional information by analyzing the messages it receives as part of the protocol execution. On the other hand, a malicious adversary can deviate arbitrarily from the protocol specification in order to achieve its goals.

Our first protocol for decision tree evaluation is secure against semi-honest adversaries. Thus, as long as both parties follow the protocol, neither party learns more information at the end of the protocol than desired. While semi-honest protocols are often easier to construct and more efficient, they do not always capture the strong notions of security we desire. Thus, we also consider a stronger, one-sided notion of security (see [21] for a more thorough discussion). In one-sided security, we require a strong simulation-based notion of security to hold against a malicious client and a weaker notion of privacy to hold against a malicious server. This means that even a malicious client cannot learn anything more than $\mathcal{T}(x)$ and what is explicitly leaked by the protocol. On the other hand, while a malicious server cannot learn anything about the client's input x, it can still corrupt the client's output. This is a suitable notion of security in scenarios where the server is not incentivized to cause the client to receive corrupted or wrong outputs, but may try to learn something about the value of x.

2.2 Cryptographic Primitives

In this section, we give a brief overview of the cryptographic tools and primitives that we use to construct our protocols.

Homomorphic encryption. In this work, we require an additively homomorphic encryption scheme such as [6, 22, 23]. A semantically secure public-key encryption system with message space $\mathcal{R}$ (we model $\mathcal{R}$ as a ring) is specified by three algorithms $\mathsf{KeyGen}, \mathsf{Enc}_{\mathsf{pk}}, \mathsf{Dec}_{\mathsf{sk}}$ (for key generation, encryption, decryption, respectively). We write pk and sk to denote the public and secret keys, respectively. We drop the pk and

sk subscripts when the choice of keys is unambiguous. The key-generation algorithm outputs a public-private key pair $(\mathsf{pk}, \mathsf{sk})$. The security requirement is the standard notion of semantic security [24]. In an additively homomorphic encryption system, we require an additional public-key operation $\mathsf{Add}_{\mathsf{pk}}$ such that for all messages $m_0, m_1 \in \mathcal{R}$, the following holds with overwhelming probability:

$$\mathsf{Dec}\left(\mathsf{Add}\left(\mathsf{Enc}(m_0), \mathsf{Enc}(m_1)\right)\right) = m_0 + m_1.$$

In addition to additive homomorphism, we also require that our scheme supports scalar multiplication, that is, given an encryption $\mathsf{Enc}(m)$ of $m \in \mathcal{R}$, there exists a public-key operation that produces an encryption $\mathsf{Enc}(km)$ for all $k \in \mathbb{Z}$.

It turns out that in our constructions, we do not need to decrypt arbitrary ciphertexts; instead, it suffices to be able to test whether a given ciphertext is an encryption of 0. Thus, we instantiate our scheme with the "exponential" variant of the ElGamal encryption scheme [25]. Since we exploit the particular structure of the exponential ElGamal encryption scheme, we sketch out some additional details here. Let $\mathbb{G}$ be an additively written group of prime order p for which the decisional Diffie-Hellman (DDH) assumption [26] holds, and let P be a generator of $\mathbb{G}$. The plaintext space is the ring $\mathbb{Z}_p$ and the ciphertext space is $\mathbb{G}^2$. Then, we have the following:

- KeyGen: Take $s \xleftarrow{\$} \mathbb{Z}_p$ and let $Q \leftarrow sP$. Output the public key $\mathsf{pk} \leftarrow (P, Q)$ and the secret key $\mathsf{sk} \leftarrow s$.
- $\mathsf{Enc}_{\mathsf{pk}}(m)$: Take $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$. Choose $r \xleftarrow{\$} \mathbb{Z}_p$ and output the ciphertext $\mathsf{ct} \leftarrow (r \cdot \mathsf{pk}_1,\ r \cdot \mathsf{pk}_2 + m \cdot \mathsf{pk}_1)$. We write $\mathsf{Enc}_{\mathsf{pk}}(m; r)$ to denote an encryption of m with randomness r.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$: For $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, output $\mathsf{ct}_2 - \mathsf{sk} \cdot \mathsf{ct}_1$.
- $\mathsf{Add}_{\mathsf{pk}}(\mathsf{ct}, \mathsf{ct}')$: Choose $r \xleftarrow{\$} \mathbb{Z}_p$ and output $(\mathsf{ct}_1 + \mathsf{ct}'_1 + r \cdot \mathsf{pk}_1,\ \mathsf{ct}_2 + \mathsf{ct}'_2 + r \cdot \mathsf{pk}_2)$.
- $\mathsf{ScalarMultiply}_{\mathsf{pk}}(\mathsf{ct}, k)$: Choose $r \xleftarrow{\$} \mathbb{Z}_p$ and output $(k \cdot \mathsf{ct}_1 + r \cdot \mathsf{pk}_1,\ k \cdot \mathsf{ct}_2 + r \cdot \mathsf{pk}_2)$.

Observe that the "decryption" function does not produce an element of the plaintext space $\mathbb{Z}_p$: if ct is an encryption of m, the decryption function produces the element $mP \in \mathbb{G}$. Recovering m amounts to solving a discrete log problem in $\mathbb{G}$. In our protocols, it suffices to check if m is zero, which can be performed by checking if mP is the identity in $\mathbb{G}$. Moreover, we will use the fact that if ct is an encryption of a uniformly random message $m \in \mathbb{Z}_p$, then $\mathsf{Dec}_{\mathsf{sk}}(m)$ yields a uniformly random element in $\mathbb{G}$ (since $|\mathbb{G}| = p$).

Oblivious Transfer. Oblivious transfer (OT) [27, 28, 29, 30] is a primitive commonly employed in cryptographic protocols. In standard *1-out-of-n* OT, there are two parties, denoted the sender and the receiver. The sender holds a database $x_1, \ldots, x_n \in \{0,1\}^\kappa$ and the client holds a selection bit $i \in [n]$. At the end of the protocol, the client learns x_i and nothing else about the contents of the database; the server learns nothing.

2.3 Decision Trees and Random Forests

Decision trees are frequently encountered in machine learning and can be used for both classification and regression. We view a decision tree $\mathcal{T} : \mathbb{Z}^n \to \mathbb{Z}$ as implementing a function on an n-dimensional *feature space.*[1] We refer to elements $x \in \mathbb{Z}^n$ as *feature vectors.* Evaluation of a decision tree on an input $x \in \mathbb{Z}^n$ corresponds directly to taking a path through the tree from the root to a leaf. More concretely, for each internal node v_k in the tree, we associate a boolean function $f_k(x) = \mathbf{1}\left\{x_{i_k} < t_k\right\}$. Here, $i_k \in [n]$ is an index into a feature vector $x \in \mathbb{Z}^n$ and t_k is a threshold. For each leaf node ℓ, we associate an output value z_ℓ. To evaluate the decision tree on an input $x \in \mathbb{Z}^n$, we start at the root node, and at each internal node v_k, we evaluate $f_k(x)$. Depending on whether $f_k(x) = 0$ or 1, we either take the left or right branch of the tree.

[1]In practice, the feature space is usually $\mathbb{R}^n$ rather than $\mathbb{Z}^n$; to apply our method to these scenarios, we use a fixed-point encoding of real-valued elements as integers.

We repeat this process until we reach a leaf node ℓ. The value z_ℓ of the leaf node is the predicted response $\mathcal{T}(x)$.

We also review some standard terminology for trees. First, the depth of a tree is the length of the longest path from the root to a leaf. The i^{th} layer of the tree is the set of nodes of distance i from the root. We say a binary tree with depth d is complete if for $0 \leq i \leq d$, the i^{th} layer contains exactly 2^i nodes.

Complete, binary trees. In general, decision trees need not be binary or complete. However, we note that any non-binary tree can be transformed into a binary tree at the expense of increasing the depth of the resulting tree.

Moreover, all incomplete binary trees can be transformed into a complete binary tree by introducing dummy nodes. Let $\mathcal{T}$ be a binary tree with depth d. So long as there are leaf nodes v in the tree at a layer $\ell < d$, we apply the following transformation: replace v with a dummy internal node with two children, each of which is a terminal node with value z. For simplicity, we associate each dummy nodes with the trivial boolean function $f(x) = 0$. Note that this has no effect on the classification, since irrespective of the path taken at the dummy node, the response value is unchanged.

Node indices. We use the following indexing scheme to refer to nodes in a complete binary tree. Let $\mathcal{T}$ be a decision tree with depth d. Set v_1 to be the root node. We label the remaining nodes inductively: if v_i is an internal node, let v_{2i} be its left child and v_{2i+1} be its right child. For convenience, we also define a separate index from 0 to $2^d - 1$ for the leaf nodes. Specifically, if v_i is the parent of a leaf node, then we denote its left and right children by z_{2i-m-1} and z_{2i-m}, where m is the number of internal nodes in $\mathcal{T}$. With this indexing scheme, the leaves of the tree, when read from left-to-right, correspond with the ordering $z_0, \ldots, z_{2^d-1}$.

Paths in a binary tree. It is often useful to associate paths in a complete binary tree with bit strings and vice versa. Specifically, let $\mathcal{T}$ be a complete binary tree with depth d. We can specify a path by a bit string $b = b_1 \cdots b_d \in \{0,1\}^d$, where b_i denotes whether we visit the left child or the right child when we are at a node at level $i-1$. Starting at the root node (level 0), and traversing according to the bits b, this process uniquely defines a path in $\mathcal{T}$. We refer to this path as the path induced by b in $\mathcal{T}$.

Similarly, we define the notion of a *decision string* for an input x on a tree $\mathcal{T}$. Let m be the number of internal nodes in a complete binary tree $\mathcal{T}$. The decision string is the concatenation $f_1(x) \cdots f_m(x)$ of the value of each predicate f_i on the input x. Thus, the decision string encodes information regarding which path the evaluation would have taken at every internal node. By construction, it also uniquely identifies the evaluation path of x in $\mathcal{T}$. Thus, we can also refer to the path induced by a decision string s in $\mathcal{T}$. As part of defining the path s, the decision string also specifies the index of the leaf node at the end of the path. We let $\phi : \{0,1\}^m \to \{0, \ldots, m\}$ be the function that maps a decision string s for a complete binary tree with m decision nodes onto the index of the corresponding leaf node in the path induced by s in $\mathcal{T}$.

Random forests. A natural way of improving the performance of decision tree classifiers is by combining responses (e.g., taking the mean or mode) from many decision trees. In the case of a *random forest*, we train many decision trees, where each tree is trained using a random subset of the features. This has the effect of decorrelating the individual trees in the forest. More concretely, we can describe a random forest $\mathcal{F}$ by an ensemble of decision trees $\mathcal{F} = \{\mathcal{T}_i\}_{i \in [n]}$. If the random forest operates by computing the mean of the individual decision tree outputs, then we have $\mathcal{F}(x) = \frac{1}{n} \sum_{i \in [n]} \mathcal{T}_i(x)$. See [2] for additional details.

3 Semi-honest Decision Tree Evaluation

In this section, we describe our two-party protocol for privately evaluating decision trees in the semi-honest model. We show how to generalize these protocols to random forests in Section 5. In our scenarios, we assume the client holds a feature vector and the server holds a model (either a decision tree or a random forest). The protocol we describe is secure assuming a semantically secure additively homomorphic encryption scheme and a semi-honest secure OT protocol.

3.1 Setup

We make the following assumptions about our model:

- The client has a well-formed ElGamal key-pair. In the semi-honest setting, it suffices that the client generates the key-pair at the beginning of the protocol using the KeyGen algorithm. In the malicious setting, the client also has to include a proof of the well-formedness of its key. Since this issue is orthogonal to the construction of our protocol, we assume that the client already has a valid key-pair at the onset of the protocol.
- The client's private data consists of a feature vector $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, where $x_i \geq 0$ for all i. Let t be the bit-length of each component of the feature vector.
- The server holds a complete binary decision tree $\mathcal{T}$ with m internal nodes, some of which may be dummy nodes (see Section 2.3). Let $i_1, \ldots, i_\ell$ denote the indices of the non-dummy internal nodes in $\mathcal{T}$. For each non-dummy node v_{i_k}, we associate an index j_k and a threshold t_k such that $f_{i_k}(x) = \mathbf{1}\{x_{j_k} < t_k\}$. For the dummy nodes v, we set the boolean function to $f_v(x) = 0$. The output space of $\mathcal{T}$ is $\{0,1\}^\kappa$, that is, the values of $\mathcal{T}$ at the leaves $z_0, \ldots, z_m$ are elements of $\{0,1\}^\kappa$.

Leakage. In constructing our protocol, we aim to minimize the amount of information leaked about the decision tree. However, for practical purposes, our protocol will leak a small amount of information regarding the tree: the depth d of the decision tree and the number ℓ of non-dummy internal nodes. It is possible to hide the number of non-dummy nodes in the tree by treating every internal node in the tree as a non-dummy node (for instance, each dummy node would have an associated index and threshold). This will increase the runtime and bandwidth, but does provide increased privacy.

3.2 Building Blocks

In this section, we describe the construction of our decision tree evaluation protocol in the semi-honest setting. Before presenting its full details (Figure 2), we provide a high-level survey of our methods. As stated in Section 3.1, the decision trees we consider have a very simple structure known to the client: a complete binary tree $\mathcal{T}$ with depth d. Let $z_0, \ldots, z_{2^d-1}$ be the leaf values of $\mathcal{T}$. Suppose also that we allow the client to learn the index $i \in \{0, \ldots, 2^d - 1\}$ of the leaf node in the path induced in $\mathcal{T}$ by x. If the client knew the index i, it can then *privately* obtain the value $z_i = \mathcal{T}(x)$ by engaging in a *1-out-of-*2^d OT with the server. In this case, the server's "database" is the set $\{z_0, \ldots, z_{2^d-1}\}$.

The problem with this simple scheme is that revealing the index of the leaf node to the client reveals too much information about the structure of the tree. We address this by having the server randomize the tree, that is, the server chooses a random subset of the internal nodes in the decision tree and interchanges their left and right subtrees. After this randomization process, we can show that the decision string corresponding to the client's query is uniform over all bit strings with length $2^d - 1$. At this point, we can reveal to the client the decision string corresponding to its input on the permuted tree. The basic idea for our semi-honest secure decision tree evaluation protocol is thus as follows:

1. The server randomly permutes the tree $\mathcal{T}$ to obtain an equivalent tree $\mathcal{T}'$.
2. The client and server engage in a comparison protocol for each decision node in $\mathcal{T}'$. At the end of this phase, the client learns the result of each comparison in $\mathcal{T}'$, and therefore, the decision string corresponding to its input in $\mathcal{T}'$.
3. Using the decision string, the client determines the index i that contains its value $z_i = \mathcal{T}(x)$. The client engages in an OT protocol with the server to obtain the value z_i.

Let $x, y \in \mathbb{Z}, x \neq y$, be t-bit integers with binary representation $x_1 x_2 \cdots x_t$, $y_1 y_2 \cdots y_t$, respectively. Let $(\mathsf{pk}, \mathsf{sk})$ be a public-private key-pair for an additively homomorphic encryption scheme over $\mathbb{Z}_p$. Assume the client holds sk.

- **Client's input and server's output**: None.
- **Server's input**: Encryptions $\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_t)$ under the client's public key pk, the value y, and a bit b.
- **Client's output**: A bit b' such that $b \oplus b' = \mathbf{1}\{x < y\}$.

1. **Server**: Take $s \leftarrow 1 - 2b$. For $i \in [t]$, choose $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and define

$$c_i = r_i \cdot \left[x_i - y_i + s + 3 \cdot \sum\nolimits_{j<i} (x_j \oplus y_j) \right]. \tag{1}$$

   The server homomorphically computes $\mathsf{Enc}(c_i)$ and sends $\{\mathsf{Enc}(c_1), \ldots, \mathsf{Enc}(c_t)\}$ to the client in random order.

2. **Client**: First, decrypt the received ciphertexts to obtain the set $\{c_1, \ldots, c_t\}$. Then, compute and output $b' \leftarrow \mathbf{1}\{\exists i \colon c_i = 0\}$.

Figure 1: Private comparison protocol.

Comparison protocol. The primary building block we require for private decision tree evaluation is a comparison protocol. We use a variant of the comparison protocol from [17, 22, 31]. This protocol is a two-round protocol between a client and a server. The server possesses two t-bit integers: the bitwise encryption of a value x under the client's public key and the value y in the clear. At the end of the protocol, the client and server each possess one bit of a secret sharing of the comparison bit $\mathbf{1}\{x < y\}$. The client should not learn anything about y. The protocol is given in Figure 1. We show that this protocol is correct.

Theorem 1. *Let $x \neq y$ be t-bit integers with binary representation $x_1 x_2 \cdots x_t$ and $y_1 y_2 \cdots y_t$, respectively. Let $(\mathsf{pk}, \mathsf{sk})$ be the public-private key pair of an additively homomorphic public-key encryption scheme with encryption function Enc and plaintext space $\mathbb{Z}_p$, such that $p > 3t - 1$. Then, the protocol in Figure 1 correctly implements the functionality $(-, [(\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_n)), y, b]) \mapsto (b', -)$ where $b \oplus b' = \mathbf{1}\{x < y\}$.*

Proof. Consider the expression in (1):

$$c_i = r_i \cdot \left[x_i - y_i + s + 3 \cdot \sum\nolimits_{j<i} (x_j \oplus y_j) \right] \in \mathbb{Z}_p$$

Since $|x_i - y_i + s| \leq 2$ and $0 \leq \sum_{j<i} (x_j \oplus y_j) \leq 3(t-1)$, we conclude that if $p > 3t - 1$, then $c_i = 0$ over $\mathbb{Z}_p$ if and only if $c_i = 0$ when the computation is performed over $\mathbb{Z}$. Suppose the server's input bit $b = 0$. Then, $s = 1$. We show that $x < y$ if and only if the client's output bit $b' = 1$. This is equivalent to showing that there exists i such that $c_i = 0$. We use the fact that $x < y$ if and only if there exists $i \in [t]$ such that $\forall j < i : x_j = y_j$ and $x_i < y_i$. Consider the value of c_i. Since $x_j = y_j$ for all $j < i$, $\sum_{j<i} (x_j \oplus y_j) = 0$. Moreover, since $x_i < y_i$, it follows that $x_i = 0$ and $y_i = 1$, so $x_i - y_i + s = 0$. We conclude that $c_i = 0$.

For the reverse direction, suppose there exists i such that $c_i = 0$. Since $|x_i - y_i + s| \leq 2$ and $x_j \oplus y_j \geq 0$, $c_i = 0$ only if for all $j < i$, $x_j = y_j$. Since $s = 1$, $x_i - y_i + s = 0$ if and only if $x_i - y_i = -1$, which implies $x_i < y_i$. We conclude that $x < y$. The analysis for the case where $b = 1$ is similar.

To conclude the proof, we show that the server can compute $\mathsf{Enc}(c_i)$. First, the server computes $x_i - y_i + s$ using the additive homomorphism of the encryption scheme. Next, it computes $x_j \oplus y_j$ using the fact that $x_j \oplus y_j = x_j$ if $y_j = 0$ and $1 - x_j$ if $y_j = 1$. Since the server knows y, this just requires additive homomorphism. Finally, multiplication by r_i can be performed if the scheme supports scalar multiplication. $\square$

Remark 1. It is important in the comparison protocol that $x \neq y$. Otherwise, if $x = y$, $c_i \neq 0$ for all i in (1), and so the client always outputs $b' = 0$, which does not satisfy the requirement $b \oplus b' = \mathbf{1}\{x < y\}$. To ensure that the comparison protocol will never be invoked on $x, y \in \mathbb{Z}$ where $x = y$, we use the fact that $x \leq y$ if and only if $2x < 2y + 1$. Thus, when comparing integers x and y, the server first augments the client's input with an extra encryption of 0 in the least significant bit (effectively substituting $2x$ for the client's input) and substitutes $2y + 1$ for its own input. The output of the comparison protocol b' then satisfies $b \oplus b' = \mathbf{1}\{x \leq y\}$.

Decision tree randomization. As mentioned at the beginning of Section 3.2, we apply a tree randomization procedure to hide the structure of the tree. For each decision node v, we interchange its left and right subtrees with equal probability. Moreover, to preserve correctness, if we interchanged the left and right subtrees of v, we replace the boolean function f_v at v with its negation $\tilde{f}_v(x) := f_v(x) \oplus 1$. More precisely, on input a decision tree $\mathcal{T}$ with m internal nodes $v_1, \ldots, v_m$, we construct a permuted tree $\mathcal{T}'$ as follows:

1. Initialize $\mathcal{T}' \leftarrow \mathcal{T}$ and choose $s \xleftarrow{\$} \{0,1\}^m$. Let $v'_1, \ldots, v'_m$ denote the internal nodes of $\mathcal{T}'$ and let $f'_1, \ldots, f'_m$ be the corresponding boolean functions.
2. For $i \in [m]$, set $f'_i(x) \leftarrow f_i(x) \oplus s_i$. If $s_i = 1$, then swap the left and right subtrees of v'_i. Do not reindex the nodes of $\mathcal{T}'$ during this step.
3. Reindex the nodes $v'_1, \ldots, v'_m$ in $\mathcal{T}'$ according to the standard indexing scheme described in Section 2.3. Output the permuted tree $\mathcal{T}'$.

In the above procedure, we obtain a new tree $\mathcal{T}'$ by permuting the nodes of $\mathcal{T}$ according to a bit-string $s \in \{0,1\}^m$. We denote this process by $\mathcal{T}' \leftarrow \pi_s(\mathcal{T})$. By construction, for all $x \in \mathbb{Z}^n$ and all $s \in \{0,1\}^m$, we have that $\mathcal{T}(x) = \pi_s(\mathcal{T})(x)$. Moreover, we define the permutation τ_s that corresponds to the permutation on the nodes of $\mathcal{T}$ effected by π_s. In other words, the node indexed i in $\mathcal{T}$ is indexed $\tau(i)$ in $\mathcal{T}'$. Then, if $\sigma \in \{0,1\}^m$ is the decision string of $\mathcal{T}$ on input x, $\tau(\sigma \oplus s)$ is the decision string of $\pi_s(\mathcal{T})$ on x.

3.3 Semi-honest Secure Decision Tree Evaluation

The protocol for evaluating a decision tree with security against semi-honest adversaries is given in Figure 2. Just to reiterate, in the first part of the protocol, the client and server participate in an interactive comparison protocol that ultimately reveals to the client a decision string for a permuted tree. Given the decision string, the client obtains the response via an OT protocol. We now show that this protocol is correct.

Lemma 1. If the client and server follow the protocol in Figure 2, then at the end of the protocol, the client learns $\mathcal{T}(x)$.

Proof. Invoking Theorem 1, for $k \in [\ell]$, $b_k \oplus b'_k = f_{j_k}(x)$. Since $f_v(x) = 0$ for the dummy nodes v, we conclude that σ is the decision string of x on $\mathcal{T}$. Then, as noted in Section 3.1, $\tau_s(\sigma \oplus s) = \sigma'$ is the corresponding decision string of x on $\pi_s(\mathcal{T}) = \mathcal{T}'$. By correctness of the OT protocol, the client learns the value of $z'_{\phi(\sigma')} = \mathcal{T}'(x)$ at the end of Step 5. Since the tree randomization process preserves the function, it follows that $z'_{\phi(\sigma')} = \mathcal{T}'(x) = \mathcal{T}(x)$. □

We now prove that this protocol is secure against semi-honest adversaries. To simplify the proof, we prove security in the OT-hybrid model (see Appendix A). In other words, we assume that in the final step of the protocol in Figure 2, the client and server have access to a trusted party that implements the OT functionality. By instantiating the OT protocol with a protocol secure against semi-honest adversaries (for instance, the protocols of [29, 30]), and invoking the sequential composability theorem of Canetti [15], we achieve full security against semi-honest adversaries.

Theorem 2. *The protocol in Figure 2 securely computes (Definition 1) the functionality $(x, \mathcal{T}) \mapsto (\mathcal{T}(x), -)$ in the presence of semi-honest adversaries in the OT-hybrid model.*

Proof. The proof is deferred to Appendix B.1. □

Let $(\mathsf{pk}, \mathsf{sk})$ be a public-private key-pair for an additively homomorphic encryption scheme over $\mathbb{Z}_p$. The client holds the secret key sk. Let $t = \lceil \log_2 p \rceil$.

- **Client input**: A feature vector $x \in \mathbb{Z}_p^n$. Let $x_{i,j}$ denote the j^{th} bit of x_i.
- **Server input**: A complete, binary decision tree $\mathcal{T}$ with m internal nodes. Let $i_1, \ldots, i_\ell$ be the indices of the non-dummy nodes, and let $f_{i_k}(x) = \mathbf{1}\{x_{j_k} \leq t_k\}$, where $j_k \in [n]$ and $t_k \in \mathbb{Z}_p$. For the dummy nodes v, set $f_v(x) = 0$. Let $z_0, \ldots, z_m \in \{0,1\}^\kappa$ be the values of the leaves of $\mathcal{T}$. See Section 3.1 for more details.
- **Client output**: A value $z = \mathcal{T}(x)$.
- **Server output**: None.

1. **Client**: For each $i \in [n]$ and $j \in [t]$, compute and send $\mathsf{Enc}(x_{i,j})$ to the server.
2. **Client and Server**: The server chooses $b \xleftarrow{\$} \{0,1\}^\ell$. For each $k \in [\ell]$, the client and server engage in the private comparison protocol of Figure 1 (with the transformation from Remark 1). On the k^{th} iteration, the server's input to the comparison protocol consists of the ciphertexts $\mathsf{Enc}(x_{j_k,1}), \ldots, \mathsf{Enc}(x_{j_k,t})$, the bit b_k, and the threshold t_k. The server can send all of the messages for the ℓ comparisons in a single round. The client then replies with all ℓ responses. Thus, this step just requires two rounds of communication.
3. **Client**: At the end of the comparison protocol, the client obtains shares $b'_1, \ldots, b'_\ell$. The client sends $\mathsf{Enc}(b'_1), \ldots, \mathsf{Enc}(b'_\ell)$ to the server.
4. **Server**: The server chooses $s \xleftarrow{\$} \{0,1\}^m$ and constructs the permuted tree $\mathcal{T}' \leftarrow \pi_s(\mathcal{T})$, where π_s is the permutation associated with the bit-string s (see Section 2.3). Initialize $\sigma \leftarrow 0^m$. For $k \in [\ell]$, update $\sigma_{i_k} \leftarrow b_k \oplus b'_k$. Let τ_s be the permutation on the node indices of $\mathcal{T}$ effected by π_s, and compute $\sigma' \leftarrow \tau_s(\sigma \oplus s)$. The server homomorphically computes $\mathsf{Enc}(\sigma')$ (each bit is encrypted individually) and sends the result to the client.
5. **Client and Server**: The client decrypts the server's message to obtain σ' and then the client computes the index i of the leaf node containing the response (the client computes $i \leftarrow \phi(\sigma')$, with $\phi(\cdot)$ as defined in Section 2.3). Next, it engages in a *1-out-of-*$(m+1)$ OT with the server to learn the value z_i. In the OT protocol, the client supplies the index i and the server supplies the permuted leaf values $z'_0, \ldots, z'_m$ of $\mathcal{T}'$.

Figure 2: Semi-honest secure decision tree evaluation protocol.

3.4 Asymptotic Analysis

In this section, we briefly describe the asymptotic performance of the protocol in Figure 2. Let d be the depth of the tree, n be the dimension of the feature space, ℓ be the number of non-dummy internal nodes, and t be the precision (the number of bits needed to represent each component of the feature vector). Consider the client's computation. First, encrypting its feature vector bitwise requires $O(nt)$ public-key operations. Next, the client receives ℓt ciphertexts from the server for the comparison protocol; it decrypts each set of ciphertexts to determine which ones contain an encryption of 0. This requires $O(\ell t)$ computation. Finally, after the client receives the (encrypted) decision string, it needs to decrypt $O(d)$ bits in the string to determine the index of the leaf node. Instantiating the OT with the Naor-Pinkas OT [28], the OT can be performed in time $O(d)$, for a total complexity of $O(t(n + \ell) + d)$.

On the server side, evaluating the comparisons requires $O(\ell t)$ public-key operations. After receiving the comparison responses, the server must construct the decision string, which has length $2^d - 1$. Using the Naor-Pinkas OT, the server computation in the OT protocol is also $O(2^d)$. The overall computational complexity for the server is then $O\left(\ell t + 2^d\right)$.

If we fix the precision t needed to encode elements of the feature space, then the client's computation is linear in the dimension, the number of comparisons, and the depth of the decision tree. For the server, however, the complexity is linear in the number of comparisons, but exponential in the depth. Thus, our protocols are better-suited for shallow decision trees (say, $d < 20$), which is oftentimes sufficient for practical applications.

4 One-Sided Decision Tree Evaluation

Next, we describe an extension of our proposed decision tree evaluation protocol that achieves stronger security against malicious adversaries. Specifically, we describe a protocol that is fully secure against a malicious client and private against a malicious server. To motivate the construction of the extended protocol, we highlight two possible ways a malicious client might attack the protocol in Figure 2:

- In the first step of the protocol, a malicious client might send encryptions of plaintexts that are not in $\{0,1\}$. The server's response could reveal information about the thresholds in the decision tree.
- When the client engages in OT with the server, it can request an arbitrary index i' of its choosing and learn the value $z_{i'}$ of an arbitrary leaf node, independent of its query.

In the following sections, we develop tools that prevent these two particular attacks on the protocol. We then show that the resulting protocol provides one-sided security against malicious adversaries. We work under the same assumptions as in the semi-honest setting (Section 3.1) with one crucial difference: we no longer distinguish between dummy and non-dummy nodes. In other words, if a node $v \in \mathcal{T}$ is a dummy node, we still associate v with a (dummy) index i_v and threshold t_v. This will have implications on the performance of our protocol, which we elaborate on in Section 6.

4.1 Building Blocks

To extend our protocol to operate in the presence of malicious adversaries, we make use of two additional cryptographic primitives: proofs of knowledge and an adaptation of conditional OT. In this section, we give a brief survey of these methods.

Proofs of knowledge. At the beginning of Section 4, we noted that a malicious client can deviate from the protocol and submit encryptions of non-binary values as its query. To protect against this malicious behavior, we require that the client include zero-knowledge proofs that certify it is submitting encryptions of bits. Since we are using the exponential variant of the ElGamal encryption scheme, proving that a ciphertext encrypts a certain value can be performed efficiently using the Chaum-Pedersen protocol [32]. Using the standard OR proof transformation in [33, 34], we can prove that a ciphertext either encrypts a 0 or a 1. Moreover, we apply the Fiat-Shamir heuristic [35] to make these proofs non-interactive in the random oracle model.

In our exposition, we use the notation introduced in [36] to specify these proofs. We write statements of the form $\mathsf{PoK}\{(r) : c_1 = rP \wedge c_2 = rQ\}$ to denote a zero-knowledge proof-of-knowledge of the integer r that satisfies $c_1 = rP$ and $c_2 = rQ$. All values not enclosed in parenthesis are assumed to be known to the verifier. We refer readers to [37, 38] for a more complete treatment of these topics.

Conditional oblivious transfer. The second problem with the semi-honest protocol we highlighted pertains to the fact that the client is not bound to request the correct index in the OT protocol. To protect against the client requesting an arbitrary index independent of its query, we modify the protocol such that the client is only able to learn the value that corresponds to its query. We use a technique similar to *conditional oblivious transfer* introduced in [39, 40]. Like OT, (strong) conditional OT is a two-party protocol between a sender and a receiver. The receiver holds an input x and the sender holds two secret values k_0, k_1 and an input y. At the conclusion of the protocol, the receiver learns k_1 if (x, y) satisfy some predicate Q, and k_0 otherwise. For instance, a "less-than" predicate would be $Q(x,y) = \mathbf{1}\{x < y\}$. As in OT, the server learns nothing at the conclusion of the protocol. Neither party learns $Q(x,y)$. In our setting, we adapt the comparison protocol in Figure 1 so that at the conclusion of the protocol, in addition to learning the output bit b', the client also learns an associated key $k_{b'}$. Since b' is just a single share of the actual comparison

Let $x, y \in \mathbb{Z}, x \neq y$ be t-bit integers with binary representation $x_1 x_2 \cdots x_t$, $y_1 y_2 \cdots y_t$, respectively. Let $\mathsf{pk} = (P, Q)$ be a public key for an exponential ElGamal cryptosystem with corresponding secret key $\mathsf{sk} = s$ (note that $Q = sP$ in this case). Let $\mathbb{G}$ be the plaintext space where $|\mathbb{G}|$ is a prime p.

- **Client's input and server's output**: None.
- **Server's input**: Encryptions $\{\mathsf{Enc}(x_i)\}_{i \in [t]}$ under the client's public key, the value y, a bit b, and two scalars, $\alpha_0, \alpha_1 \in \mathbb{Z}_p$.
- **Client's output**: A bit b' such that $b \oplus b' = \mathbf{1}\{x < y\}$ and a key $k = k_{b'}$, where $k_i = \alpha_i P$ for $i \in \{0, 1\}$.

1. **Server**: Let $s \leftarrow 1 - 2b$. For $i \in [n]$, choose $r_i, r_i' \xleftarrow{\$} \mathbb{Z}_p$ and define

$$c_i = r_i \cdot \left[x_i - y_i + s + 3 \cdot \sum\nolimits_{j<i} (x_j \oplus y_j)\right] \qquad c_i' = r_i' \cdot \left[x_i - y_i - s + 3 \cdot \sum\nolimits_{j<i} (x_j \oplus y_j)\right] \tag{2}$$

   Choose blinding factors $\rho_1, \ldots, \rho_n, \rho_1', \ldots, \rho_n' \xleftarrow{\$} \mathbb{Z}_p$. Compute the following tuples of ciphertexts:

$$\begin{array}{ll} A = (\mathsf{Enc}(c_1), \ldots, \mathsf{Enc}(c_n)) & A' = (\mathsf{Enc}(c_1'), \ldots, \mathsf{Enc}(c_n')) \\ B = (\mathsf{Enc}(c_1\rho_1 + \alpha_0), \ldots, \mathsf{Enc}(c_n\rho_n + \alpha_0)) & B' = (\mathsf{Enc}(c_1'\rho_1' + \alpha_1), \ldots, \mathsf{Enc}(c_n'\rho_n' + \alpha_1)) \end{array}$$

   Randomly permute the entries in A and apply the same permutation to the entries in B. Repeat with A' and B'. Send A, A', B, B' to the client.

2. **Client**: For each ciphertext $A_i = \mathsf{Enc}(c_i)$ in A, test whether $c_i = 0$. If so, then let $k \leftarrow \mathsf{Dec}(\mathsf{sk}, B_i)$, where B_i denotes the i^{th} component of B. Output the key k and the bit $b' = 1$. Otherwise, for each ciphertext $A_i' = \mathsf{Enc}(c_i')$ in A', test whether $c_i' = 0$. If so, then set $k \leftarrow \mathsf{Dec}(\mathsf{sk}, B_i')$. Output the key k and the bit $b' = 0$. If neither condition holds, then output $\perp$.

Figure 3: Private comparison protocol with conditional key transfer.

bit, it does not reveal any information about the comparison bit to the client. We present our modified comparison protocol in Figure 3.

Theorem 3. *Let $\mathsf{pk} = (P, Q)$ be an ElGamal public key and let $\mathsf{sk} = s$ be the corresponding secret key. Let $x \neq y$ be t-bit integers with binary representation $x_1 x_2 \cdots x_t$ and $y_1 y_2 \cdots y_t$, respectively. Let $\mathbb{Z}_p$ be the underlying plaintext space and suppose that $p > 3t + 1$. Let f be the functionality*

$$\left(-, \left[\{\mathsf{Enc}(x_i)\}_{i \in [t]}, y, b, (\alpha_0, \alpha_1)\right]\right) \mapsto ((b', k_{b'}), -),$$

where $b \oplus b' = \mathbf{1}\{x < y\}$ and $k_i = \alpha_i P$. Then, the protocol in Figure 3 securely computes f in the presence of malicious clients.

Proof (Sketch). We defer the formal proof to Appendix B.2, and give a sketch of the main arguments here. In the private comparison protocol (Section 3.2, Figure 1), the client outputs 1 if and only if one of the server's ciphertexts $\mathsf{Enc}(c_i)$ is an encryption of 0. This enables a simple method for hiding a value α. For each encryption $\mathsf{Enc}(c_i)$, the server chooses a random $\rho_i \xleftarrow{\$} \mathbb{Z}_p$ and homomorphically computes the ciphertext $\mathsf{Enc}(\rho_i c_i + \alpha)$. If $c_i \neq 0$, then $\rho_i c_i + \alpha$ is uniform over $\mathbb{Z}_p$. Otherwise, $\rho_i c_i + \alpha = \alpha$. Thus, if the client has $\{\mathsf{Enc}(\rho_i c_i + \alpha)\}$ in addition to $\{\mathsf{Enc}(c_i)\}$, then the client is able to compute α if and only if there exists i such that $c_i = 0$. If for all i, $c_i \neq 0$, then $\{\mathsf{Enc}(\rho_i c_i + \alpha)\}$ consists of encryptions of random values in $\mathbb{Z}_p$; these values perfectly hide α. Since we instantiate the encryption scheme with exponential ElGamal, the "decryption" operation applied to $\mathsf{Enc}(\alpha)$ yields the group element αP. Thus, the modified comparison protocol operates analogously to the simple comparison protocol in Figure 1, except by including an additional set of ciphertexts, the client is able to learn not only the bit b', but also an associated key $k_{b'} = \alpha_{b'} P$. □

Let $\mathsf{pk} = (P, Q)$ be a public key for an exponential ElGamal cryptosystem with corresponding secret key sk. Let $\mathbb{G}$ be the plaintext space where $|\mathbb{G}|$ is a prime p. We assume the client holds the secret key. Let $t = \lceil \log_2 p \rceil$.

- **Client input**: A feature vector $x \in \mathbb{Z}_p^n$. Let $x_{i,j}$ denote the j^{th} bit of x_i.
- **Server input**: A complete, binary decision tree $\mathcal{T}$ with decision nodes $v_1, \ldots, v_m$. For all $k \in [m]$, the predicate f_k associated with decision node v_k is of the form $f_k(x) = \mathbf{1}\{x_{j_k} \le t_k\}$, where $j_k \in [n]$ is an index and $t_k \in \mathbb{Z}_p$ is a threshold. Let $z_0, \ldots, z_m \in \{0,1\}^\kappa$ be the values of the leaves of $\mathcal{T}$.
- **Client output**: A value $z = \mathcal{T}(x)$.
- **Server output**: None.

1. **Client:** For each $i \in [n]$ and $j \in [t]$, choose $r_{i,j} \xleftarrow{\$} \mathbb{Z}_p$. Then, compute and send encryptions
$$(c_{i,j}, c'_{i,j}) \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_{i,j}; r_{i,j}) = (r_{i,j}P, r_{i,j}Q + x_{i,j}P),$$
along with proofs
$$\pi_{i,j} \leftarrow \mathsf{PoK}\left\{(r_{i,j}) : \left[c_{i,j} = r_{i,j}P \wedge c'_{i,j} = r_{i,j}Q\right] \vee \left[c_{i,j} = r_{i,j}P \wedge c'_{i,j} = r_{i,j}Q + P\right]\right\}. \quad (3)$$
2. **Server:** For each i, j, the server verifies the proof π_{ij}. If the proof fails to verify, it aborts the protocol. Otherwise, the server does the following:
   (a) Choose $s \xleftarrow{\$} \{0,1\}^m$ and compute $\mathcal{T}' \leftarrow \pi_s(\mathcal{T})$, where π_s is the permutation associated with the bit-string s (see Section 2.3). Let τ be the permutation effected by π_s on the nodes of $\mathcal{T}$. Let $s'_1 \cdots s'_m = \tau(s_1 \cdots s_m)$. Similarly, define the permuted node indices $i'_1, \ldots, i'_m$ and thresholds $t'_1, \ldots, t'_m$ in $\mathcal{T}'$.
   (b) Choose blinding factors $\alpha_{1,0}, \ldots, \alpha_{m,0}, \alpha_{1,1}, \ldots, \alpha_{m,1} \xleftarrow{\$} \mathbb{Z}_p$. For $i \in [m]$ and $b \in \{0,1\}$, set $k_{i,b} \leftarrow \alpha_{i,b}P$.
   (c) Let $d = \log_2(m+1)$ be the depth of $\mathcal{T}'$. For each leaf node z_i in $\mathcal{T}'$, let $b_1 \cdots b_d$ be the binary representation of i, and let $i_1, \ldots, i_d$ be the indices of the nodes along the path from the root to the leaf. Compute $\hat{z}_i \leftarrow z_i \oplus \left(\bigoplus_{j \in [d]} h\left(k_{i_j, b_j}\right)\right)$, where $h : \mathbb{G} \to \{0,1\}^\kappa$ is taken from a pairwise independent family of hash functions.
   (d) Send the blinded response vector $[\hat{z}_0, \ldots, \hat{z}_m]$ to the client.
3. **Client and Server:** For each $i \in [m]$, the client and the server engage in the extended private comparison protocol of Figure 3 (with the transformation of the input from Remark 1). On the k^{th} iteration, the server's input to the comparison protocol consists of the encrypted bits $\mathsf{Enc}(x_{j_k,1}), \ldots, \mathsf{Enc}(x_{j_k,t})$, the bit s'_k, the threshold t'_k, and blinding factors $\alpha_{k,0}, \alpha_{k,1}$.
4. **Client:** At the conclusion of the extended private comparison protocol, the client learns bits $b' = b'_1, \ldots, b'_m$ along with keys $k_1, \ldots, k_m$. Let $i_1, \ldots, i_d$ be the indices of the internal nodes in the path induced by b' in a complete binary tree of depth d, and let ℓ be the index of the leaf node at the end of the path. Then, the client computes and outputs $z \leftarrow \hat{z}_\ell \oplus \left(\bigoplus_{j \in [d]} h(k_{i_j})\right)$.

Figure 4: One-sided secure decision tree evaluation protocol.

4.2 One-Sided Secure Evaluation of Decision Trees

The two-round, one-sided secure protocol for decision tree evaluation is described in Figure 4. As in the semi-honest protocol, the client begins by sending a bitwise encryption of its feature vector to the server. In addition to the ciphertexts, the client also submits zero-knowledge proofs that each of its ciphertexts is a bitwise encryption. Next, the server randomizes the tree in the same manner as in the semi-honest protocol (Section 3.2). Moreover, the server associates a randomly chosen key with each edge in the permuted tree.

The server then blinds each leaf node using the keys along the path from the root to the leaf. There is a small technicality here in that the keys here are not bit strings in $\{0,1\}^\kappa$, but rather elements in the elliptic-curve group $\mathbb{G}$. However, as long as $|\mathbb{G}| > 2^\kappa$, we can obtain a uniformly random key in $\{0,1\}^\kappa$ using a pairwise independent family of hash functions and invoking the leftover hash lemma [41]. The server then sends the blinded response vector to the client. Finally, the server computes its response according to the extended comparison protocol of Figure 3. From the extended comparison protocol, the client learns the keys for the edges in the tree that are consistent with its query. Using these keys, the client unblinds the value (and only this value) at the index corresponding to its query. We now state the security theorems for the protocol in Figure 4. We defer the proofs to Appendix C.

Theorem 4 (Client Privacy)**.** *The protocol in Figure 4 is private against a malicious server.*

Theorem 5 (Server Security)**.** *The protocol in Figure 4 securely computes the functionality* $(x, \mathcal{T}) \mapsto (\mathcal{T}(x), -)$ *in the presence of a malicious client.*

4.3 Asymptotic Analysis

We perform a similar analysis of the asymptotic performance for the one-sided secure protocol as we did for the semi-honest secure protocol (Section 3.4). Since the analysis is similar, we just state the results. Using the same parameters as in Section 3.4, the client's computation requires $O(nt + d)$ operations and the server's computation requires $O(2^d t)$ operations. In comparison to the semi-honest secure protocol, the client's computation is lower, and in particular, independent of the number of decision nodes in the one-sided secure protocol. The server's computation, though, is substantially higher. Thus, if the server desires to protect itself against a malicious client, then it potentially must do considerably more work (up to a factor of t).

5 Extensions

In this section, we describe two extensions to our proposed protocol for private decision tree evaluation. First, we provide a generalization of the protocol to support private evaluation of random forests. Next, we describe a way to capture more expressive decision tree models, namely decision trees with categorical variables.

Random forest evaluation. As mentioned in Section 2.3, a random forest classifier is an ensemble classifier that aggregates the responses of multiple decision trees in order to obtain a more robust response. Typically, response aggregation is done by taking a majority vote of the individual decision tree outputs, or taking the average of the responses. A simple, but naïve method for generalizing our protocol to a random forest $\mathcal{F} = \{\mathcal{T}_i\}_{i \in [n]}$ is to run the decision tree evaluation protocol n times, once for each decision tree $\mathcal{T}_i$. At the end of the n protocol executions, the client learns the values $\mathcal{T}_1(x), \ldots, \mathcal{T}_n(x)$, and can then compute the mean, majority, or some other function of the individual decision tree outputs.

The problem is that this simple protocol reveals to the client the values $\mathcal{T}_i(x)$ for all $i \in [n]$. In the case where the output of the random forest is the average (or any affine function) of the individual classifications, we can do better by using additive secret sharing. Specifically, suppose that the value of each leaf of $\mathcal{T}_i$ (for all i) is an element of $\mathbb{Z}_p$. Then, at the beginning of the protocol, the server chooses blinding values $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$. For each tree $\mathcal{T}_i$, the server blinds each of its leaf values $v \in \mathbb{Z}_p$ by computing $v \leftarrow v + r_i$. Since r_i is uniform over $\mathbb{Z}_p$, v is now uniformly random over $\mathbb{Z}_p$. The protocol execution proceeds as before, except that the server also sends the client the value $r \leftarrow \sum_{i \in [n]} r_i$. At the conclusion of the protocol, the client learns the values $\{v_i + r_i\}_{i \in [n]}$ where $v_i = \mathcal{T}_i(x)$. In order to compute the mean of $v_1, \ldots, v_n$, the client computes the sum $\sum_{i \in [n]}(v_i + r_i) - r = \sum_{i \in [n]} v_i$ which is sufficient for computing the mean provided the client knows the number of trees in the forest. We note that this protocol generalizes naturally to evaluating any affine function of the individual responses with little additional overhead. The leakage in the case of affine functions is the total number of comparisons, the depth of each decision tree in the model (or a bound on the depth if all the trees are padded to the maximum depth), and the total number of trees. No information about the response value of any single decision tree in the forest is revealed.

Equality testing and categorical variables. In practice, feature vectors might contain categorical variables in addition to numeric variables. When branching on the value of a categorical variable, the natural operation is testing for set membership. For instance, if x_i is a categorical variable that can take on values from a set $S = \{s_1, \ldots, s_n\}$, then a branching criterion is more naturally phrased in the form $\mathbf{1}\{x_i \in S'\}$ for some $S' \subseteq S$. We leverage this observation to develop a method for testing for set inclusion based on private equality testing when the number of attributes is small. More precisely, to determine whether $x_i \in S'$, we test whether $x = s$ for each $s \in S'$.

We use the two-party equality testing protocol of [42]. Fix a group $\mathbb{G}$ with generator P and let $(\mathsf{pk}, \mathsf{sk})$ be a key-pair for an exponential ElGamal encryption scheme where the client holds the secret key sk. Let $\mathbb{Z}_p$ be the plaintext space for the encryption scheme. Let $x, y \in \mathbb{Z}_p$ denote the client and server's input to the equality testing protocol, respectively. To test whether $x = y$, the client sends $\mathsf{Enc}(x)$ to the server. The server then chooses a random $r \xleftarrow{\$} \mathbb{Z}_p^*$ and homomorphically computes $\mathsf{Enc}(r(x - y))$ and sends it to the client. The key observation is that $r(x - y)$ is 0 if $x = y$, and otherwise, is uniform in $\mathbb{Z}_p$.

We now extend the decision tree evaluation protocol to support categorical variables with up to t classes, where t is the number of bits needed to encode a numeric component in the feature vector. To evaluate a decision function of the form $\mathbf{1}\{x_i \in S'\}$, where $S' = \{y_1, \ldots, y_m\}$, the server constructs the ciphertexts $\mathsf{Enc}(r_j(x_i - y_j))$ for each $y_j \in S'$ as above and additional dummy ciphertexts $\mathsf{Enc}(r_{j+1}), \ldots, \mathsf{Enc}(r_t)$, for $r_{j+1}, \ldots, r_t \xleftarrow{\$} \mathbb{Z}_p^*$. The server sends these ciphertexts to the client in random order. Clearly, $x_i \in S'$ if and only if one of these ciphertexts is an encryption of 0. Moreover, this set of ciphertexts is computationally indistinguishable from the set of ciphertexts the client would receive for a comparison node. Finally, in the decision tree evaluation protocol, we require that the client learns only a share of the value of the decision variable. This is also possible for set membership testing: depending on the value of the server's share of the decision variable, the server can test membership in S' or in its complement $\overline{S'}$. Since $\left|S' \cup \overline{S'}\right| \leq t$, the decision tree evaluation protocols can support these tests with almost no modification. The only difference in the semi-honest setting is that the client encrypts categorical variables directly rather than bitwise. In the one-sided secure setting, the client additionally needs to prove that it sent an encryption of a valid categorical value. To facilitate this, we number the categories from 1 to $|S| \leq n$, where S is the set of all possible categories for a given variable. Then, in addition to providing the encryption $c \leftarrow \mathsf{Enc}(x; r)$ of a value $x \in \{1, \ldots, |S|\}$, the client also includes a proof $\mathsf{PoK}\{(x, r) : (c = \mathsf{Enc}(x; r)) \wedge (1 \leq x \leq |S|)\}$. Since $|S|$ is small, this can be done using the same OR proof transformation of Chaum-Pedersen proofs.

6 Experimental Evaluation

We implemented the semi-honest (Figure 2) and one-sided secure protocol (Figure 4) for decision tree and random forest evaluation (using the extension from Section 5). In the semi-honest setting, we also implemented the extension for categorical variables described in Section 5. Our implementation was written in C++ and we used the MSR-ECC library [43, 44] to implement the elliptic-curved-based exponential ElGamal algorithm. For increased efficiency, we instantiate our protocol with sub-protocols that are provably secure in the random oracle model.[2] We instantiate the random oracle with SHA-256, and leverage the implementation in the OpenSSL library. In the semi-honest protocol, we instantiated the semi-honest secure *1-out-of-n* OT with the Naor-Pinkas OT [28], and implemented it using the the OT library of Asharov *et al.* [30]. For the malicious-secure setting, we use NTL over GMP for the finite field arithmetic needed for the Chaum-Pedersen proofs of knowledge. We compile our code using `g++ 4.8.2` on a machine running Ubuntu

[2]We can obtain security in the standard model in the semi-honest setting by instantiating the OT with an OT that is semi-honest secure in the standard model. Similarly, we can achieve one-sided security in the standard model by instantiating the proofs of knowledge with a (multi-round) interactive protocol secure in the standard model.

Dataset	n	d	m	Method	Security Level	End-to-End Time (s)	Computation (s)		Bandwidth (KB)
							Client	Server	
ECG	6	4	6	Barni *et al.* [18]	80	-	2.609	6.260	112.2
				Bost *et al.* [17]	80	-	2.297	1.723	3555
				Our protocol	**128**	**0.344**	**0.136**	**0.162**	**101.9**
Nursery	8	4	4	Bost *et al.*	80	-	1.579	0.798	2639
				Our protocol	**128**	**0.269**	**0.113**	**0.126**	**101.7**

Table 1: Performance of protocols for semi-honest secure decision tree evaluation. The decision trees have m decision nodes and depth d. The feature vectors are n-dimensional. The "End-to-End Time" column gives the total time for the protocol execution, as measured by the client (including network communication). Performance numbers for the Barni *et al.* and Bost *et al.* methods are taken from [17], which use a similar evaluation environment.

14.04.1. In our experiments, we run the client-side code on a commodity laptop with a multicore 2.30 GHz Intel Core i7-4712HQ CPU and 16 GB of RAM. We run the server on a compute-optimized Amazon EC2 instance with a dual-core 2.60GHz Intel Xeon E5-2666 v3 processor and 3.75 GB of RAM. While many components of our protocol are parallelizable, we do not leverage parallelism in our benchmarks. The network speed in our experiments is around 40-50 Mbps.

We conduct all experiments at a 128-bit security level. This corresponds to using a 256-bit elliptic curve for the exponential ElGamal implementation (we use curve `numsp256d1` from [43, 44]). In the semi-honest setting, we also instantiate the `OT` scheme at the 128-bit security level using the parameters described in [30]. For our first set of benchmarks, we compare our performance against the protocols in [17, 18] on the ECG classification tree from [18] and the Nursery dataset from the UCI Machine Learning Repository [20]. Since the decision tree used in [17] for the Nursery dataset is not precisely specified, we test our protocol against a tree with the same depth and number of comparison nodes as in [17]. In our benchmarks we measure the client computation, server computation, and total communication bandwidth between the client and the server. Our results are summarized in Table 1. The numbers we report for the performance of [17, 18] are taken from [17, Table 4].[3] Our results show that despite running at a higher security level (128 bits vs. 80 bits), our protocol is over 19x faster for the client and over 5x faster for the server compared to the protocols in [17] based on somewhat homomorphic encryption (SWHE). Moreover, our protocol is more than 25x more efficient in terms of communication. Compared to the protocol in [18] based on homomorphic encryption and garbled circuits, of Barni *et al.* [18], our protocol is almost 20x faster for both the client and the server, and requires slightly less communication.

Scalability and sparsity. To better understand the scalability of our protocols to larger decision trees, we perform a series of experiments on randomly generated decision trees with different depths and densities. In the first set of experiments, we consider complete decision trees. While complete decision trees are unlikely to arise in practice, they serve as a "worst-case" bound on the performance of our protocol for trees of a given depth. In our experiments, we fix the dimension of the feature space to $n = 16$ and the precision to $t = 64$. We separately measure the time for the client's and server's computation as well as the total communication between the two parties. These results are shown in Figure 5. For complete trees, the number of comparisons m is exponential in the depth d. Recall from Section 3.4 that the client's complexity is linear in m, or equivalently, exponential in d for complete trees. Likewise, the server's computation is exponential in the depth of the tree. Since the number of decision nodes in a complete tree also grows exponentially in the depth, the computation required for privately evaluating complete decision trees grows *linearly* in the size of the tree. We also note that even for large trees with over ten thousand decision nodes, our protocol still operates on the order of minutes.

[3]While our test environment is not identical to that in [17], they are similar: Bost *et al.* conduct their experiments on a machine with a 2.66 GHz Intel Core i7 processor with 8 GB of RAM.

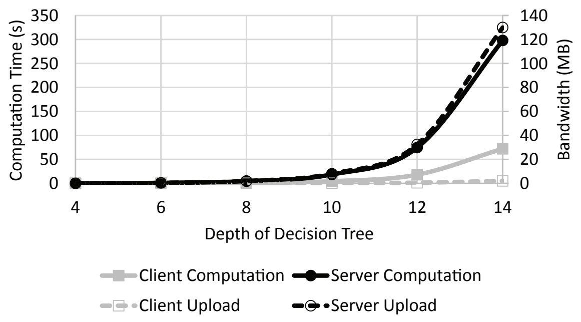

Figure 5: Client and server computation (excluding network communication) and total bandwidth for semi-honest protocol on complete decision trees.

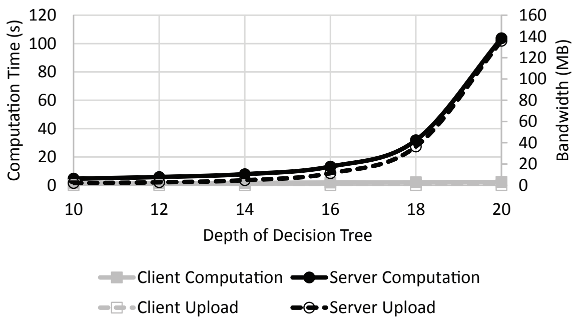

Figure 6: Client and server computation (excluding network communication) and total bandwidth for semi-honest protocol on "sparse" trees.

We briefly compare our protocol against the private decision tree evaluation protocol of Brickell *et al.* [19]. In their benchmarks, evaluating a 1100 node tree required approximately 5 minutes and 25 MB of communication. On a similarly sized tree over an equally large feature space, our protocol completes in 30 seconds and requires about 10 MB of communication, representing a 10x improvement in computation and 2.5x in bandwidth.

In practice, however, it is unlikely that we encounter complete decision trees. To get a sense of the computation time and bandwidth needed to evaluate trees that are sparse, we perform another set of experiments on trees where the number m of decision nodes grows linearly in the depth d of the tree. Here, we set $m = 25d$. We present the results of these experiments in Figure 6. The important observation here is that the client's computation now grows linearly, rather than exponentially in the depth of the tree. This is consistent with the analysis in Section 3.4. Unfortunately, because the server computes a decision string for the complete tree, the server's computation increases exponentially in the depth. However, since the cost of the homomorphic operations in the comparison protocol is greater than the cost of computing the decision string, the protocol is still able to scale to deeper trees and maintain runtimes on the order of minutes. The limiting factor in this case is the exponential growth in the size of the server's response. Even though the tree is sparse, because the decision string communicated by the server encodes information about every node in the padded tree, the amount of communication from the server to the client is mostly unchanged. The communication upstream from the client to the server, though, is significantly reduced (linear rather than exponential in the depth).

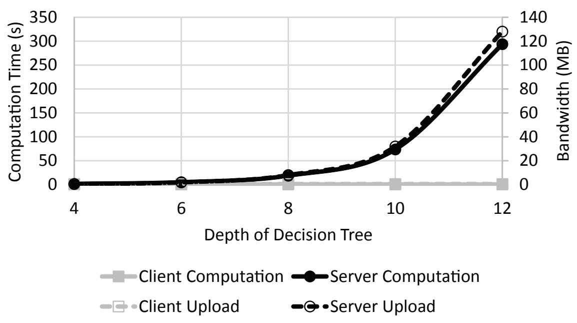

Figure 7: Client and server computation (excluding network communication) and total bandwidth for one-sided secure protocol for decision tree evaluation.

One-sided secure protocol. Next, we consider the performance of the one-sided secure protocol from Figure 4. Since this protocol does not distinguish between dummy and non-dummy nodes, the performance is independent of the number of actual decision nodes in the tree. Thus, we only consider the benchmark for complete, binary trees. Again, we fix the dimension $n = 16$ and the precision $t = 64$ in our experiments. The results are summarized in Figure 7. As noted in Section 4.3, the client's computation grows linearly in the depth of the tree, and so, is virtually constant in these experiments. Notably, in all experiments in Figure 7, the total computation time on the client side is less than half a second. Moreover, the amount of communication from the client to the server depends only on n, t, and is independent of the size of the model. Thus, the client's computation is quite small in the one-sided secure protocol, and therefore, the protocol is well-suited in scenarios where the computational power of the client is limited. The tradeoff though is that the server now performs significantly more work. Nonetheless, even for trees of depth 12 (with more than 4000 decision nodes), the protocol completes in a few minutes. It is also worth noting that the one-sided protocol is almost non-interactive, that is, the client does not have to be online when the server is computing its response. This is in contrast to our semi-honest protocol which involved a small (constant) number of rounds of communication in the course of the protocol execution.

As a final note, we also benchmarked the one-sided secure protocol on the ECG and Nursery datasets. On the ECG dataset, the client's and server's computation took 0.191 and 0.948 s, respectively, with a total communication of 660 KB. On the Nursery dataset, the client's and server's computation took 0.216 s and 0.937 s, respectively, with a total communication of 720 KB. Even in spite of the higher security level and the stronger security guarantees, our protocol remains 2x faster than that of [17] in total computation time and requires 3.5x less communication. Thus, even the one-sided secure protocol is viable in practice, and for the reasons mentioned above (low client overhead and only two rounds of interaction), might be more suitable than the semi-honest secure protocol in certain situations.

Random forests. As described in Section 5, we generalize our decision tree evaluation protocol to support random forests with an affine aggregation function with almost no additional overhead than the cost of evaluating each decision tree privately. The computational and communication complexity of the random forest evaluation protocol is just the complexity of the decision tree evaluation protocol scaled up by the number of trees in the forest. We give some example performance numbers for evaluating a random forest with different number of trees. Each tree in the forest has depth at most 10 and contains exactly 100 comparisons. As before, we take $n = 16$ for the dimension and $t = 64$ for the precision. Our results are summarized in Table 2.

Performance on real datasets. We conclude our analysis by describing experiments on decision trees and random forests trained on real datasets. We benchmark our protocol on five datasets from the UCI

k	End-to-End (s)	Computation (s)		Bandwidth (MB)	
		Client	Server	Client	Server
10	26.161	4.652	19.069	0.247	9.106
25	65.067	11.343	47.756	0.430	22.761
50	130.496	22.252	95.133	0.736	45.520
100	256.362	44.759	190.196	1.346	91.037

Table 2: Semi-honest random forest evaluation benchmarks. Each forest consists of k trees (each tree has depth at most 10 and exactly 100 comparisons). The "End-to-End" measurements include the time for network communication.

Dataset	n	Tree		Forest	
		d	m	d	m
`breast-cancer`	9	8	12	11	276
`heart-disease`	13	3	5	8	178
`housing`	13	13	92	12	384
`credit-screening`	15	4	5	9	349
`spambase`	57	17	58	16	603

Table 3: Parameters for decision trees and random forests trained on UCI datasets [20]: n is the dimension of the data, m is the number of decision nodes in the model, d is the depth of the tree(s) in the model.

repository [20] spanning application domains such as breast cancer diagnosis and credit rating classification. We train our trees using standard Matlab tools (`classregtree` and `TreeBagger`). To obtain more robust models, we introduce a hyperparameter $\alpha \geq 1$ that specifies the minimum number of training examples that must be associated with each leaf node in the tree. We choose α by running 10-fold cross validation [1] for several candidate values of α. In most cases, $\alpha > 1$, which has an added benefit of reducing the depth of the resulting model. Additionally, for two of the datasets (`housing` and `spambase`), we choose the best value for α such that the depth of the resulting trees and forests is within 20. The size and depth of the resulting trees and forests, along with the dimension of the feature space for each of the datasets is given in Table 3.

Of the five datasets we use for the benchmarks, four are binary classification tasks. The exception is the `housing` dataset which is a regression problem. The `heart-disease` and `credit-screening` datasets incorporate a mix of categorical and numeric variables. In our experiments, we operate at a 128-bit security level, and use 64 bits of precision to represent each component of the feature vector. For the random forest experiments, we train a random forest consisting of 10 decision trees, again choosing the α hyperparameter via 10-fold cross-validation. The performance of the semi-honest decision tree evaluation protocol on each of the five datasets is summarized in Table 4 and the performance of the semi-honest random forest evaluation protocol is summarized in Table 5. We remark that while our random forest extension only applies to affine aggregation functions which are suitable for regression problems and not classification problems in general, our classification examples are all examples of binary classification, in which case, taking the mean response is appropriate. One might interpret the average in this case to be a measure of the confidence of the model in the prediction.

The performance benchmarks demonstrate that our semi-honest secure protocols are suitable for evaluating trees and forests that could arise in practice. Even for relatively deep trees over high-dimensional features spaces such as the spam classification (`spambase`) dataset, our semi-honest protocol completes in under 20 seconds. In all cases, the client's computation time in the semi-honest setting is under a second. For all but the largest tree (`spambase`), the total bandwidth is under 2 MB. For smaller trees, the bandwidth is usually on the order of 100-200 KB. With random forest evaluation, both the client and server have to perform additional computation and there is more data exchanged. Nonetheless, the amount of computation required from the client is on the order of a few seconds, and the server's computation is also on the order of

Dataset	End-to-End (s)	Computation (s)		Bandwidth (KB)	
		Client	Server	Upload	Download
`breast-cancer`	0.545	0.190	0.293	73.7	132.0
`heart-disease`	0.370	0.143	0.134	73.3	43.9
`housing`	4.081	0.603	2.464	115.7	1795.2
`credit-screening`	0.551	0.128	0.135	49.9	45.0
`spambase`	16.595	0.628	12.902	463.4	17363.3

Table 4: Performance of semi-honest decision tree evaluation protocol on UCI datasets [20].

Dataset	End-to-End (s)	Computation (s)		Bandwidth (KB)	
		Client	Server	Upload	Download
`breast-cancer`	9.671	1.431	6.888	106.7	4853.1
`heart-disease`	4.691	0.859	3.387	94.9	1758.2
`housing`	15.152	1.913	10.726	152.2	8357.4
`credit-screening`	8.737	1.553	6.446	92.9	3456.5
`spambase`	93.276	2.874	69.839	531.6	89310.7

Table 5: Performance of semi-honest random forest evaluation protocol on UCI datasets [20].

seconds (except in the case of `spambase` where the computation took over a minute). The communication does increase for random forest evaluation, and the amount of data the client needs to download is a couple MB. Excluding `spambase`, we believe this is still a modest amount of communication.

Because we did not implement the extension to categorical variables for the one-sided secure protocol, we do not have complete benchmarks for all five datasets. We do have concrete results for the `housing` and `breast-cancer` datasets. On the `breast-cancer` dataset, end-to-end decision tree evaluation using the one-sided secure protocol completes in 12.3 s including network communication (client computation of just 0.263 s) and 8.2 MB of total communication. For the `housing` dataset, the decision tree evaluation completes in 357.0 s (client computation of 0.384 s) and 256 MB of communication. As noted earlier, the one-sided secure protocol is mostly non-interactive, so the client does not have to be online for most of the server's computation. These results indicate that for low-depth trees ($d < 10$), the one-sided secure protocol remains viable, but the amount of communication does grow rapidly in the depth of the tree.

7 Related Work

We survey some of the related work that has been done in the area of privately evaluating decision trees. The earliest work in this area tended to focus on training decision trees in a privacy-preserving manner [45, 46, 47]. In the case of [46, 47] multiple parties each have their own individual datasets, and the objective is to compute a decision tree on their joint data without revealing their individual datasets.

On the contrary, this work focuses on the problem of privately evaluating decision trees, where it is assumed that one of the parties holds a trained decision tree or random forest. Several practical protocols have been developed for privately evaluating decision trees, or more generally, linear branching programs. In [18, 19], the authors develop protocols for privately evaluating linear branching programs (of which decision trees are a special case) based on a combination of homomorphic encryption and garbled circuits. Because these protocols solve a more general problem of evaluating linear branching programs, they are not as competitive in performance in comparison to our protocol which exploits the simple structure of decision trees. More recently, the work of [17] describes a fairly generic protocol for decision tree evaluation that also splits up the decision tree evaluation protocol into two components: a comparison phase followed by an evaluation phase. In [17], Bost *et al.* view the decision tree as a polynomial in the decision variable and evaluate the polynomial using a SWHE scheme [5, 48, 49]. In all of these works, the authors achieve semi-honest security, although [19, 18] note that their protocols can be made secure in the malicious setting without much difficulty. Nonetheless, we do not know of any existing implementations of a private decision tree evaluation protocol that achieves stronger security.

On the more theoretical side, private decision tree evaluation falls into the category of private function evaluation (where we view the decision tree as the underlying *private* function to be evaluated). In the last few years, several generic approaches for private function evaluation have been proposed [50, 51] which are asymptotically very efficient and can be made robust against malicious adversaries. Restricting to private decision tree evaluation, Mohassel *et al.* describe a protocol for evaluating oblivious decision programs based on OT in [52]. They provide an abstract method for evaluating decision programs in both the semi-honest and malicious setting. However, it is unclear how to integrate comparisons efficiently into their protocol, which would be necessary for evaluating the particular kind of decision trees considered in this work. Then, there are also generic solutions for private function evaluation based on Yao's circuits [9] where the circuit being evaluated is the universal circuit, and the decision tree is provided as an input to the circuit. While the protocols of [50, 51, 52] apply to our setting, we are not aware of any existing implementations of these methods. In the case of using Yao's circuits, Bost *et al.* note in [17] that these methods tend to be prohibitively expensive in practice.

8 Conclusion

In this work, we present two protocols for privately evaluating decision trees and random forests. The first protocol based on additive homomorphic encryption and oblivious transfer achieves security against semi-honest adversaries. Our results show that our protocol achieves at least 5x improvement in computation and communication compared to the recent protocol for private decision tree evaluation based on SWHE by Bost *et al.* [17]. Moreover, we demonstrate that our protocols scale well to trees spanning tens of thousands of internal nodes and depths up to 20; even on these large trees, our protocol completes on the order of a few seconds to a couple of minutes. Moreover, we describe simple extensions for extending our protocol to random forests as well as trees containing categorical variables. Our benchmarks demonstrate that even with these extensions, our protocols still achieve strong performance on decision tree and random forest models trained on real datasets.

Additionally, we extend our semi-honest secure protocol to a protocol that provides security against a malicious client and privacy against a malicious server using efficient Chaum-Pedersen proofs and leveraging a new variant of conditional oblivious transfer. While our one-sided secure protocol is less efficient compared to our semi-honest secure protocol, our benchmarks still indicate that it is viable for shallow decision trees. For the ECG and Nursery trees that have been used as benchmarks in previous works, we note that our one-sided secure protocol still achieves better performance in terms of computation when compared to previous *semi-honest* secure protocols. We do not know of any implementations of a private decision tree evaluation protocol that achieves any kind of malicious security.

We conclude by noting that in our protocols, the server's computation is always exponential in the depth of the tree, which is not ideal for evaluating sparse, but deep trees. This requirement arises because we assume that the general structure of the tree can be revealed to the client (thus, all trees are padded to a complete tree). While asymptotically, the protocol in [17] scales polynomially in the depth of the decision tree, the computational overhead of SWHE renders it impractical for evaluating trees of even moderate depth. We leave as an open question whether more efficient protocols, possibly employing additional rounds of interaction, can be designed for the problem of privately evaluating decision trees and random forests. Another interesting direction to consider would be to perform an empirical analysis of different generic methods for private function evaluation [50, 51] and see how these methods compare against custom protocols for this particular problem.

Acknowledgments

We thank Melissa Chase, Seny Kamara, and Payman Mohassel for many helpful comments and discussion on this work. The first author is supported in part by an NSF Graduate Research Fellowship under grant number DGE-114747 and an Amazon AWS in Education Grant. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. Springer New York Inc., 2001.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[3] A. Singh and J. V. Guttag, "A comparison of non-symmetric entropy-based classification trees and support vector machine for cardiovascular risk stratification," *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2011, p. 7982, 2011.

[4] H. C. Koh, W. C. Tan, and C. P. Goh, "A two-step method to construct credit scoring models with data mining techniques," *International Journal of Business and Information*, vol. 1, pp. 96–118, 2006.

[5] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[6] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1999, vol. 1592, pp. 223–238.

[7] T. Graepel, K. E. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Information Security and Cryptology - ICISC*, 2012, pp. 1–21.

[8] J. W. Bos, K. E. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of Biomedical Informatics*, vol. 50, pp. 234–243, 2014.

[9] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *FOCS*, 1986, pp. 162–167.

[10] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[11] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008, pp. 486–498.

[12] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," Cryptology ePrint Archive, Report 2012/265, 2012, http://eprint.iacr.org/.

[13] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *IEEE Symposium on Security and Privacy*, 2013, pp. 478–492.

[14] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications.* New York, NY, USA: Cambridge University Press, 2004.

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

[16] ——, "Security and composition of cryptographic protocols: a tutorial (part I)," *SIGACT News*, vol. 37, no. 3, pp. 67–92, 2006.

[17] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," *IACR Cryptology ePrint Archive*, vol. 2014, p. 331, 2014.

[18] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A. Sadeghi, and T. Schneider, "Secure evaluation of private linear branching programs with medical applications," in *ESORICS*, 2009, pp. 424–439.

[19] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *CCS*, 2007, pp. 498–507.

[20] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[21] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols - Techniques and Constructions*, ser. Information Security and Cryptography. Springer, 2010.

[22] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *ACISP*, 2007, pp. 416–430.

[23] I. Damgård, M. Jurik, and J. B. Nielsen, "A generalization of Paillier's public-key system with applications to electronic voting," *Int. J. Inf. Sec.*, vol. 9, no. 6, pp. 371–385, 2010.

[24] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, April 1984.

[25] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985, pp. 10–18.

[26] D. Boneh, "The decision Diffie-Hellman problem," in *ANTS*, 1998, pp. 48–63.

[27] M. O. Rabin, "How to exchange secrets with oblivious transfer," *IACR Cryptology ePrint Archive*, vol. 2005, p. 187, 2005.

[28] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *STOC*, 1999, pp. 245–254.

[29] ——, "Efficient oblivious transfer protocols," in *SODA*, 2001, pp. 448–457.

[30] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *ACM Conference on Computer and Communications Security*, 2013, pp. 535–548.

[31] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies*, 2009, pp. 235–253.

[32] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, 1992, pp. 89–105.

[33] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO*, 1994, pp. 174–187.

[34] B. A. Huberman, M. K. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *EC*, 1999, pp. 78–86.

[35] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986, pp. 186–194.

[36] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups (extended abstract)," in *CRYPTO*, 1997, pp. 410–424.

[37] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *STOC*, 1985, pp. 291–304.

[38] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *CRYPTO*, 1992, pp. 390–420.

[39] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," in *EUROCRYPT*, 1999, pp. 74–89.

[40] I. F. Blake and V. Kolesnikov, "Strong conditional oblivious transfer and computing on intervals," in *ASIACRYPT*, 2004.

[41] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1364–1396, 1999.

[42] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *NDSS*, 2011.

[43] J. Bos, C. Costello, P. Longa, and M. Naehrig, "Specification of curve selection and supported curve parameters in MSR ECCLib," Microsoft Research, Tech. Rep. MSR-TR-2014-92, June 2014.

[44] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: An efficiency and security analysis," *IACR Cryptology ePrint Archive*, vol. 2014, p. 130, 2014.

[45] R. Agrawal and R. Srikant, "Privacy-preserving data mining," *SIGMOD Rec.*, vol. 29, no. 2, 2000.

[46] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *CRYPTO*, 2000, pp. 36–54.

[47] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining - Volume 14*, ser. CRPIT '14, 2002.

[48] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, 2012, pp. 309–325.

[49] S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO*, 2014, pp. 554–571.

[50] J. Katz and L. Malka, "Constant-round private function evaluation with linear complexity," in *ASIACRYPT*, 2011, pp. 556–571.

[51] P. Mohassel and S. S. Sadeghian, "How to hide circuits in MPC: An efficient framework for private function evaluation," in *EUROCRYPT*, 2013, pp. 557–574.

[52] P. Mohassel and S. Niksefat, "Oblivious decision programs from oblivious transfer: Efficient reductions," *Financial Cryptography*, vol. 2014, pp. 269–284, 2012.

[53] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank, "On achieving the "best of both worlds" in secure multiparty computation," *SIAM J. Comput.*, vol. 40, no. 1, pp. 122–141, 2011.

A Real/Ideal World Security Model

Our security definitions follow the simulation-based approach of [14, 15, 16, 21]. We follow the conventions in [16, 53] and view the protocol execution as occurring in the presence of an adversary $\mathcal{A}$ and coordinated by an environment $\mathcal{E} = \{\mathcal{E}_\lambda\}$, which is parameterized by a security parameter λ. The environment $\mathcal{E}$ is responsible for choosing the inputs to the execution and plays the role of distinguisher between the real and ideal executions. We specialize our definitions to the setting of a two-party protocol computing a single-output functionality where only the client receives an output. In the following, we write f for the two-party functionality and π for the two-party protocol that computes f.

Real model of execution. In the real-world, the protocol execution proceeds as follows:

1. **Inputs:** For $i \in \{1,2\}$, $\mathcal{E}$ chooses inputs x_i and sends $(1^\lambda, x_i)$ to the i^{th} party $\mathcal{P}_i$. If $\mathcal{P}_i$ is corrupted by the adversary $\mathcal{A}$, then $\mathcal{E}$ gives x_i to $\mathcal{A}$. Finally, $\mathcal{E}$ sends some auxiliary information z to $\mathcal{A}$.
2. **Protocol Evaluation:** The parties begin executing the protocol π. All honest parties behave according to the protocol specification. The adversary $\mathcal{A}$ has full control over the behavior of the corrupted party and sees all messages received by the corrupted party. If $\mathcal{A}$ is *semi-honest*, then $\mathcal{A}$ directs the corrupted party to follow the protocol as specified.
3. **Output:** The honest party computes and gives its output to the environment $\mathcal{E}$. The adversary also computes a function of its view and gives it to $\mathcal{E}$.

At the end of the protocol execution, the environment $\mathcal{E}$ outputs a bit $b \in \{0,1\}$. Let $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$ be the random variable corresponding to the value of this bit.

Ideal model of execution. In the ideal-world execution, the parties have access to a trusted third party (TTP) that evaluates the function f. We now describe the ideal-world execution:

1. **Inputs:** For $i \in \{1,2\}$, $\mathcal{E}$ chooses inputs x_i and sends x_i to $\mathcal{P}_i$. If $\mathcal{P}_i$ is corrupted by the adversary $\mathcal{A}$, then $\mathcal{E}$ gives x_i to $\mathcal{A}$. Finally, $\mathcal{E}$ sends some auxiliary information z to $\mathcal{A}$.
2. **Submission to Trusted Party:** If $\mathcal{P}_i$ is honest, then it gives x_i to the TTP. If a party $\mathcal{P}_i$ is corrupted, then $\mathcal{P}_i$ can submit any input x_i' to the TTP as directed by $\mathcal{A}$. If $\mathcal{A}$ is *semi-honest*, then $x_i' = x_i$.
3. **Answer from Trusted Party:** Upon receiving inputs (x_1, x_2), the TTP computes and sends $f_1(x_1, x_2)$ to the client $\mathcal{P}_1$.
4. **Output:** An honest party gives the message (if any) it received from the TTP to $\mathcal{E}$. The adversary computes a function of its view of the protocol execution and gives it to $\mathcal{E}$.

At the end of the protocol execution, the environment $\mathcal{E}$ outputs a bit $b \in \{0,1\}$. Let $\mathsf{IDEAL}_{f,\mathcal{A},\mathcal{E}}(\lambda)$ be the random variable corresponding to the value of this bit. Informally, we say that a two-party protocol π securely computes a functionality f if for all efficient adversaries $\mathcal{A}$ in the real-world, there exists an adversary $\mathcal{S}$ in the ideal-world (sometimes referred to as a *simulator*) such that the outputs of the protocol executions in the real and ideal worlds are computationally indistinguishable. More formally, we have the following:

Definition 1 (Security). Let π be a two-party protocol for computing a functionality $f = (f_1(x_1, x_2), -)$. Then, π securely computes f in the presence of malicious (resp., semi-honest) adversaries if for all PPT adversaries (resp., semi-honest adversaries) $\mathcal{A}$, there exists a PPT adversary (resp., semi-honest adversary) $\mathcal{S}$ such that for every polynomial-size circuit family $\mathcal{E} = \{\mathcal{E}_\lambda\}$,

$$\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda) \overset{c}{\approx} \mathsf{IDEAL}_{f,\mathcal{S},\mathcal{E}}(\lambda).$$

In this work, we also consider the weaker notion of *privacy*, which captures the notion that an adversary does not learn anything about the inputs of the other parties beyond what is explicitly leaked by the computation and its inputs/outputs. We use the definitions from [53]. Specifically, define the random variable $\mathsf{REAL}'_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$ exactly as $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$, except in the final step of the protocol execution, the environment $\mathcal{E}$ only receives the output from the adversary (and *not* the output from the honest party). Define $\mathsf{IDEAL}'_{f,\mathcal{A},\mathcal{E}}(\lambda)$ similarly. Then, we can define the notion for a two-party protocol to privately compute a functionality f:

Definition 2 (Privacy). Let π be a two-party protocol for computing a functionality $f = (f_1(x_1, x_2), -)$. Then, π privately computes f in the presence of malicious (resp., semi-honest) adversaries if for all PPT adversaries (resp., semi-honest adversaries) $\mathcal{A}$, there exists a PPT adversary (resp., semi-honest adversary) $\mathcal{S}$ such that for every polynomial-size circuit family $\mathcal{E} = \{\mathcal{E}_\lambda\}$,

$$\mathsf{REAL}'_{\pi,\mathcal{A},\mathcal{E}}(\lambda) \overset{c}{\approx} \mathsf{IDEAL}'_{f,\mathcal{S},\mathcal{E}}(\lambda).$$

Hybrid models. When constructing cryptographic protocols, it is often useful to decompose the desired functionality f into a sequence of simpler functionalities $f_1, \ldots, f_n$, and devise protocols $\pi_1, \ldots, \pi_n$ for each of the sub-functionalities. To simplify the proofs of security in this setting, we work in a hybrid model where we assume that during the protocol execution, the parties also have access to a trusted party that evaluates each sub-functionality. If we then instantiate each sub-functionality with a (semi-honest) secure implementation, and make sure that the protocol messages of the sub-functionalities are not interleaved with the messages of the main protocol or other sub-functionalities (e.g., the protocols are composed *sequentially*), then the sequential composition of $\pi_1, \ldots, \pi_n$ (semi-honest) securely realizes f. Refer to [15] for further details.

B Security Proofs

In this section, we give the formal proofs for Theorems 2 and 3.

B.1 Proof of Theorem 2

As noted in Section 3.1, our protocol reveals the depth d of the decision tree, the number of comparisons ℓ, the dimension n of the feature vectors, and the number of bits t used to encode each component of the feature vector. Thus, when constructing the ideal-world adversary, we give the adversary the values d, ℓ, n, t. Finally, in the OT-hybrid model, we can assume that the parties have access to a trusted party for evaluating the OT.

Adversarial server. First, we prove security against a semi-honest server. Intuitively, security against a semi-honest server follows from the fact that the server's view of the protocol execution consists only of ciphertexts, and thus, reduces to a semantic security argument. We now give the formal argument. Let $\mathcal{A}$ be a semi-honest server in the real protocol. We construct an ideal-world simulator $\mathcal{S}$ as follows:

1. At the beginning of the protocol execution, $\mathcal{S}$ receives the input $\mathcal{T}$ from the environment $\mathcal{E}$. The simulator $\mathcal{S}$ also receives the decision tree parameters d, ℓ, n, t. The simulator $\mathcal{S}$ sends the tree $\mathcal{T}$ to the trusted party.

2. Start running $\mathcal{A}$ on input $\mathcal{T}$. Next, $\mathcal{S}$ generates a public-private key-pair $(\mathsf{pk}, \mathsf{sk})$ for the additive homomorphic encryption scheme used in the protocol execution. Then, $\mathcal{S}$ computes and sends nt fresh encryptions $\mathsf{Enc}(0)$ of 0 to the server.

3. After $\mathcal{A}$ replies with the results of the comparison protocol, $\mathcal{S}$ computes and sends ℓ fresh encryptions $\mathsf{Enc}(0)$ of 0 to $\mathcal{A}$.

4. Output whatever $\mathcal{A}$ outputs.

We argue that $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda) \overset{c}{\approx} \mathsf{IDEAL}_{\pi,\mathcal{S},\mathcal{E}}(\lambda)$. Using Lemma 1 and the fact that $\mathcal{A}$ is semi-honest, we have that at the end of the protocol execution in the real world, the client obtains $\mathcal{T}(x)$ where x is the client's input. Since $\mathcal{S}$ is semi-honest, this also holds in the ideal world. Because $\mathcal{T}(x)$ is a deterministic function in the inputs $\mathcal{T}, x$, the joint distribution of the client's output and the adversary's output decomposes. To complete the proof, it thus suffices to show that the view $\mathcal{S}$ simulates for $\mathcal{A}$ is computationally indistinguishable from the view of $\mathcal{A}$ interacting in the real protocol.

The view of $\mathcal{A}$ in the real protocol consists of two components: the bitwise encryptions $\{\mathsf{Enc}(x_{i,j})\}_{i\in[n],j\in[t]}$ of the client's input and the encrypted bit string $\{\mathsf{Enc}(b'_i)\}_{i\in[\ell]}$. When interacting with the simulator $\mathcal{S}$, adversary $\mathcal{A}$ sees nt independent encryptions of 0, followed by ℓ independent encryptions of 0. Security immediately follows by semantic security of the public-key encryption scheme.

Adversarial client. Next, we prove security against a semi-honest client. Let $\mathcal{A}$ be a semi-honest client in the real protocol. We construct a semi-honest simulator $\mathcal{S}$ in the ideal world as follows:

1. At the beginning of the protocol execution, $\mathcal{S}$ receives the input x from the environment $\mathcal{E}$. The simulator $\mathcal{S}$ is also given the decision tree parameters d, ℓ, n, t. The simulator sends x to the trusted party. The trusted party replies with $\hat{z}$.
2. Start running $\mathcal{A}$ on input x. Let pk be the client's public key.
3. Choose a random string $\hat{\sigma} \xleftarrow{\$} \{0,1\}^m$. In addition, choose $b \xleftarrow{\$} \{0,1\}^\ell$. Then, for each $k \in [\ell]$:
   - If $b_k = 0$, set $\hat{c}_{k,j} \xleftarrow{\$} \mathbb{Z}_p$ for all $j \in [t]$.
   - If $b_k = 1$, let $j^* \xleftarrow{\$} [t]$. Then, for all $j \in [t]$ where $j \neq j^*$, set $\hat{c}_{k,j} \xleftarrow{\$} \mathbb{Z}_p$. Set $\hat{c}_{k,j^*} \leftarrow 0$.
4. When $\mathcal{A}$ submits the encryption of its feature vector (Step 1), reply with the ciphertexts $\{\mathsf{Enc}(\hat{c}_{k,j})\}_{k\in[\ell],j\in[t]}$.
5. When $\mathcal{A}$ sends the encrypted bit string (Step 4), reply with the bitwise encryption $\mathsf{Enc}(\hat{\sigma})$ of $\hat{\sigma}$.
6. When $\mathcal{A}$ sends an index to the ideal OT functionality, reply with $\hat{z}$.
7. Output whatever $\mathcal{A}$ outputs.

In the decision tree functionality, the server has no output. Thus, to show security against a semi-honest client, it suffices to show that the output of $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$. We show that the view $\mathcal{S}$ simulates for $\mathcal{A}$ is computationally indistinguishable from the view of $\mathcal{A}$ interacting in the real protocol.

The client's view in the real protocol consists of three components: the ciphertexts $\{\mathsf{Enc}(c_{k,j})\}_{k\in[\ell],j\in[t]}$ from the private comparison protocol, the decision string $\mathsf{Enc}(\sigma')$, and the response z from the trusted OT functionality. We now show that

$$\underbrace{\left\{\{c_{k,j}\}_{k\in[\ell],j\in[t]}, \sigma', z\right\}}_{\text{view in real protocol}} \overset{c}{\approx} \underbrace{\left\{\{\hat{c}_{k,j}\}_{k\in[\ell],j\in[t]}, \hat{\sigma}, \hat{z}\right\}}_{\text{simulated view}}.$$

By correctness of the protocol (Lemma 1), $z = \mathcal{T}(x)$ in the real protocol. In the view $\mathcal{S}$ simulates, $\hat{z}$ is the response from the trusted party, and so $\hat{z} = \mathcal{T}(x) = z$. It suffices then to show that the joint distribution of the remaining components are properly distributed.

Consider the distribution of σ' in the real protocol. By construction, $\sigma' = \tau(\sigma \oplus s)$. Since s is chosen uniformly and independently of σ, $\sigma \oplus s$ is uniform over $\{0,1\}^m$. Since τ is just a permutation on the bits, $\sigma' = \tau(\sigma \oplus s)$ remains independently uniform over $\{0,1\}^m$. Thus, σ' is identically distributed as $\hat{\sigma}$.

Finally, consider the distribution of the $\{c_{k,j}\}$ in the real protocol. These components are computed according to (1) in the private comparison protocol of Figure 1. Since each $c_{k,j}$ is pre-multiplied by a uniformly random $r_{k,j} \in \mathbb{Z}_p$, each $c_{k,j}$ is either 0 or random in $\mathbb{Z}_p$. Fix an index $k \in [\ell]$ and consider the set of values $\{c_{k,j}\}_{j\in[t]}$. By construction of (1), at most one $c_{k,j} = 0$. Let $b'_k = \mathbf{1}\{\exists j : c_{k,j} = 0\}$. From Lemma 1, we have that $b_k \oplus b'_k = \mathbf{1}\{x_{i_k} < t_k\}$. Since the server chooses b_k uniformly and independently of $\mathbf{1}\{x_{i_k} < t_k\}$, b'_k is also uniform over $\{0,1\}$. In particular this means that with equal probability, the set $\{c_{k,j}\}_{j\in[t]}$ contains t uniformly random elements of $\mathbb{Z}_p$ or $t-1$ uniformly random elements of $\mathbb{Z}_p$ and one component equal to 0. Since the ciphertexts in $\{c_{k,j}\}_{j\in[t]}$ are randomly permuted in the real protocol, if $\{c_{k,j}\}_{j\in[t]}$ contains an element $c_{k,j^*} = 0$, it follows that j^* is uniform in $[t]$. Finally, since each comparison is processed independently, $c_{k,j}$ is independent of $c_{k',j'}$ whenever $k \neq k'$. But this is precisely the same distribution from which $\mathcal{S}$ samples the $\hat{c}_{k,j}$, and so we conclude that $\{c_{k,j}\}_{k\in[\ell],j\in[t]} \overset{c}{\approx} \{\hat{c}_{k,j}\}_{k\in[\ell],j\in[t]}$.

We have demonstrated that $\left\{\{c_{k,j}\}_{k\in[\ell],j\in[t]}, \sigma', z\right\} \overset{c}{\approx} \left\{\{\hat{c}_{k,j}\}_{k\in[\ell],j\in[t]}, \hat{\sigma}, \hat{z}\right\}$, and so the view $\mathcal{S}$ simulates for $\mathcal{A}$ is computationally indistinguishable from the view $\mathcal{A}$ sees in the real protocol. Correspondingly, the output of $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$; security follows. $\square$

B.2 Proof of Theorem 3

In this section, we give a proof of Theorem 3, that is, we show that the protocol in Figure 3 securely computes the functionality

$$\left(-, \left[\{\mathsf{Enc}(x_i)\}_{i\in[t]}, y, b, (\alpha_0, \alpha_1)\right]\right) \mapsto ((b', k_{b'})\, -)$$

against malicious adversaries, where $b \oplus b' = \mathbf{1}\{x < y\}$, $k_i = \alpha_i P$, and $\mathsf{pk} = (P, Q)$ is the public key for the encryption scheme. Let $\mathcal{A}$ be a malicious client interacting in the real protocol. We construct an adversarial client $\mathcal{S}$ in the ideal world such that the output of $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$. This suffices to prove security since the server does not produce any output. The simulator $\mathcal{S}$ operates as follows:

1. In the ideal execution, the trusted party gives $\mathcal{S}$ the tuple $(\hat{b}, \hat{k})$, where $\hat{b} \in \{0,1\}$ and $\hat{k} \in \mathbb{G}$.
2. For $i \in [n]$, $\mathcal{S}$ chooses $\hat{c}_1, \ldots, \hat{c}_n, \hat{c}'_1, \ldots, \hat{c}'_n \xleftarrow{\$} \mathbb{Z}_p$ and $\hat{d}_1, \ldots, \hat{d}_n, \hat{d}'_1, \ldots, \hat{d}'_n \xleftarrow{\$} \mathbb{Z}_p$. Next, choose an index $i^* \xleftarrow{\$} [n]$.
   - If $\hat{b} = 1$, update $\hat{c}_{i^*} = 0$ and $\hat{d}_{i^*} = \hat{k}$.
   - If $\hat{b} = 0$, update $\hat{c}'_{i^*} = 0$ and $\hat{d}'_{i^*} = \hat{k}$.
3. Construct the tuples
$$\begin{aligned} \hat{A} &= (\mathsf{Enc}(\hat{c}_1), \ldots, \mathsf{Enc}(\hat{c}_n)) & \hat{A}' &= (\mathsf{Enc}(\hat{c}'_1), \ldots, \mathsf{Enc}(\hat{c}'_n)) \\ \hat{B} &= \left(\mathsf{Enc}(\hat{d}_1), \ldots, \mathsf{Enc}(\hat{d}_n)\right) & \hat{B}' &= \left(\mathsf{Enc}(\hat{d}'_1), \ldots, \mathsf{Enc}(\hat{d}'_n)\right), \end{aligned}$$
   and send $\hat{A}, \hat{A}', \hat{B}, \hat{B}'$ to $\mathcal{A}$. Note that here we overload the encryption function. Specifically, if the public key pk is the tuple (P, Q), then for $k \in \mathbb{G}$, we define $\mathsf{Enc}_{\mathsf{pk}}(k) = (rP, rQ + k)$ where $r \xleftarrow{\$} \mathbb{Z}_p$. When $m \in \mathbb{Z}_p$, we define $\mathsf{Enc}_{\mathsf{pk}}(m)$ as in Section 2.2.
4. Output whatever $\mathcal{A}$ outputs.

We argue that the view of $\mathcal{A}$ interacting in the real protocol is computationally indistinguishable from its view when interacting with $\mathcal{S}$. The view of $\mathcal{A}$ in the real protocol is specified by the tuples A, A', B, B'. We show that

$$\underbrace{\{A, A', B, B'\}}_{\text{view in real protocol}} \overset{c}{\approx} \underbrace{\left\{\hat{A}, \hat{A}', \hat{B}, \hat{B}'\right\}}_{\text{simulated view}}.$$

Consider the joint distribution of $\{c_1, \ldots, c_n, c'_1, \ldots, c'_n\}$ in the real protocol. Let $\beta = b \oplus \mathbf{1}\{x < y\}$. There are two possibilities:

- Suppose $\beta = 1$. There are two possibilities:
  - $b = 0$ and $x < y$. Applying the argument from the proof of Theorem 1, there exists $i^* \in [n]$ such that $c_{i^*} = 0$ and for all $i \neq i^*$, c_i is independently uniform in $\mathbb{Z}_p$. Moreover, in the real protocol, the server randomly shuffles the elements of A so the index i^* is uniform in $[n]$. Using a similar argument, we conclude that all $j \in [n]$, each c'_j is independently uniform in $\mathbb{Z}_p$.
  - $b = 1$ and $y < x$. Again applying the argument from the proof of Theorem 1, we have that in the real protocol, the components of A, A' are distributed exactly as in the first case.

  In the simulation, $\mathcal{S}$ receives $(\hat{b}, \hat{k})$ from the TTP, so by construction, $\hat{b} = b \oplus \mathbf{1}\{x < y\} = \beta$. From the specification of the simulator, we conclude that the joint distribution of the components of A, A' in the real protocol is identically distributed as the components $\hat{A}, \hat{A}'$ in the simulation.

- Suppose $\beta = 0$. There are again two possibilities: either $b = 1$ and $x < y$ or $b = 0$ and $y < x$. Using the same style argument as in the first case, we show that the joint distribution of A, A' in the real protocol is identically distributed as the components $\hat{A}, \hat{A}'$ in the simulation.

Next, consider the components in B and B' conditioned on A and A' in the real protocol. Each component B_i and B_i' depends only on the corresponding component A_i and A_i', respectively. If $c_i \neq 0$, then $c_i \rho_i + \alpha_0$ is uniform over $\mathbb{Z}_p$, since ρ_i is chosen uniformly and independently over $\mathbb{Z}_p$. The same is true for $c_i' \rho_i' + \alpha_1$ when $c_i' \neq 0$. In particular, if $c_i, c_i' \neq 0$, then the corresponding B_i or B_i' is independent of the other components in the joint distribution. Consider the case $c_i = 0$. Then, in the real distribution $B_i = \mathsf{Enc}_{\mathsf{pk}}(\alpha_0) = \mathsf{Enc}_{\mathsf{pk}}(k_0)$ (where we use the overloaded definition of the encryption function). If instead $c_i' = 0$, then $B_i' = \mathsf{Enc}_{\mathsf{pk}}(\alpha_1) = \mathsf{Enc}_{\mathsf{pk}}(k_1)$. In other words, B_i contains an encryption of a uniformly random element if $c_i \neq 0$, and an encryption of k_0 otherwise. An analogous statement holds for B_i'. This is precisely the same distribution from which the simulator is drawing the values $\hat{B}$ and $\hat{B}'$, so we conclude that the view of $\mathcal{A}$ is computationally indistinguishable when interacting with the real protocol and interacting with the simulator. Security follows. □

C Analysis of One-Sided Secure Protocol

In this section, we complete our analysis of the correctness and security of the one-sided secure protocol in Figure 4. First, we show that the protocol is correct.

Lemma 2. If the client and server follow the protocol in Figure 4, then at the end of the protocol, the client learns $\mathcal{T}(x)$.

Proof. From Theorem 3, at the conclusion of Step 3 of the protocol, the client learns bits b_k' for $k \in [m]$ where $s_k' \oplus b_k' = \mathbf{1}\{x_{j_k} < t_k'\}$. Let b' denote the bit string $b_1' \cdots b_m'$. Then, b' is the decision string of x on $\mathcal{T}'$, and so the leaf with index $\phi(b')$ (as defined in Section 2.3) in $\mathcal{T}'$ contains the value $\mathcal{T}'(x) = \mathcal{T}(x)$. Again, invoking Theorem 3, for each $i \in [m]$, the client also obtains the keys $k_{i,b_i'}$. Then, if we denote $\phi(b')$ by ℓ, the client learns the key associated with each edge from the root to the leaf ℓ. This allows the client to compute the value used to blind z_ℓ in the blinded response vector; correctness follows. □

Before presenting the security proof, we first characterize the distribution of values in the blinded response vector $[\hat{z}_0, \ldots, \hat{z}_m]$. In particular, we show that given keys k_{i,b_i}, all but one entry in the blinded response vector is uniformly random over the set of bit strings $\{0,1\}^\kappa$. More formally, take a complete binary tree $\mathcal{T}$ with depth d and let $\{0,1\}^\kappa$ be the output space of $\mathcal{T}$. Set $m \leftarrow 2^d - 1$ to be the number of internal nodes in $\mathcal{T}$. For $j \in [m]$, choose $k_{j,0}, k_{j,1} \xleftarrow{\$} \{0,1\}^\kappa$. Next, for $i = 0, \ldots, m$, let $b_1 \cdots b_d$ be the binary representation of i and let $i_1, \ldots, i_d$ be the indices of the nodes in the path induced by $b_1 \cdots b_d$ in $\mathcal{T}$. Set z_i as follows:

$$z_i \leftarrow \bigoplus_{j \in [d]} k_{i_j, b_j}. \tag{4}$$

Intuitively, we associate a uniformly random key with each edge in $\mathcal{T}$ (or equivalently, two keys with each internal node). The value z_i is the xor of the keys associated with each edge in the path from the root to leaf i. Now, we state the main lemma on the distribution of each z_i conditioned on knowing exactly one of the two keys associated with each internal node.

Lemma 3. Fix a complete binary tree $\mathcal{T}$ with depth d. Set $m \leftarrow 2^d - 1$ and for each $0 \leq i \leq m$, construct z_i according to (4) where $k_{j,b} \xleftarrow{\$} \{0,1\}^\kappa$ for all $j \in [m]$ and $b \in \{0,1\}$. Take any bit-string $b \in \{0,1\}^m$ and let $\ell \leftarrow \phi(b)$ be the index of the leaf node in the path induced by b in $\mathcal{T}$. Then, for all $i \neq \ell$, the conditional distribution of z_i given $\{k_{j,b_j}\}_{j \in [m]}$ is independent of $\{z_j\}_{j \neq i}$ and uniformly random over $\{0,1\}^\kappa$.

Proof. We argue by induction in the depth of the tree. When $d = 1$, there are exactly two keys $k_{1,0}$ and $k_{1,1}$, and moreover, $z_0 = k_{1,0}$ and $z_1 = k_{1,1}$. Since $k_{1,0}$ and $k_{1,1}$ are independently uniform over $\{0,1\}^\kappa$, the claim follows.

For the inductive step, take a tree $\mathcal{T}$ of depth $d > 1$ and a bit-string $b \in \{0,1\}^m$. Let ℓ be the index of the leaf node in the path induced by b in $\mathcal{T}$. Let $\tilde{\mathcal{T}}$ be the complete subtree of $\mathcal{T}$ (with a common root) of depth $d-1$. For $i = 0, \ldots, 2^{d-1}$, let $\tilde{b}_1 \cdots \tilde{b}_{d-1}$ be the binary representation of i and let $\tilde{i}_1, \ldots, \tilde{i}_{d-1}$ be the indices of the nodes in the path induced by $\tilde{b}_1 \cdots \tilde{b}_{d-1}$ in $\tilde{\mathcal{T}}$. Then, define the value $\tilde{z}_i \leftarrow \bigoplus_{j \in [d-1]} k_{\tilde{i}_j, \tilde{b}_j}$. Set $\tilde{m} \leftarrow 2^{d-1} - 1$, and $\tilde{b} \leftarrow b_1 \cdots b_{d-1}$. Let $\tilde{\ell}$ be the index of the leaf node in the path induced by $\tilde{b}$ in $\tilde{\mathcal{T}}$. Invoking the induction hypothesis, we have that for all $i \neq \tilde{\ell}$, the conditional distribution of $\tilde{z}_i$ given $\left\{k_{j,b_j}\right\}_{j \in [\tilde{m}]}$ is uniformly and independently random over $\{0,1\}^\kappa$. In other words, conditioned on the set $\left\{k_{j,b_j}\right\}_{j \in [\tilde{m}]}$, we have that

$$\left[\tilde{z}_0, \ldots, \tilde{z}_{\tilde{\ell}-1}, \tilde{z}_{\tilde{\ell}+1}, \ldots, \tilde{z}_{\tilde{m}}\right] \equiv \left[\tilde{r}_0, \ldots, \tilde{r}_{\tilde{\ell}-1}, \tilde{r}_{\tilde{\ell}+1}, \ldots, \tilde{r}_{\tilde{m}}\right], \tag{5}$$

where $\tilde{r}_0, \ldots, \tilde{r}_{\tilde{m}}$ are independently and uniformly random over $\{0,1\}^\kappa$. Note that (5) holds even if we condition over the full set of keys $\left\{k_{j,b_j}\right\}_{j \in [m]}$, since $\tilde{z}_i$ is independent of k_{j,b_j} for $j > \tilde{m}$.

Each z_i is defined to be the xor of the keys along the path from the root to the i^{th} leaf. We can decompose this as taking the xor of all the keys along the path from the root to the parent of the i^{th} leaf, and xoring with the key associated with the edge from the parent to the leaf. Next, observe that the xor of the keys from the root to the parent of the i^{th} leaf is precisely $\tilde{z}_{\lfloor i/2 \rfloor}$. Finally, let p_i denote the index of the parent node of the i^{th} leaf in $\mathcal{T}$. Then,

$$z_i = \bigoplus_{j \in [d]} k_{i_j, b_j} = \tilde{z}_{\lfloor i/2 \rfloor} \oplus k_{i_d, b_d} = \tilde{z}_{\lfloor i/2 \rfloor} \oplus k_{p_i, (i \bmod 2)} \tag{6}$$

We show that $[z_0, \ldots, z_{\ell-1}, z_{\ell+1}, \ldots, z_m] \equiv [r_0, \ldots, r_{\ell-1}, r_{\ell+1}, \ldots, r_m]$, where $r_0, \ldots, r_m$ are independent and uniform in $\{0,1\}^\kappa$. From (6), we can write $z_i = \tilde{z}_{\lfloor i/2 \rfloor} \oplus k_{p_i, (i \bmod 2)}$. Let A be the subset of $\{z_i\}_{i \neq \ell}$ for which we are given the key $k_{p_i, (i \bmod 2)}$, and let B be the subset of $\{z_i\}_{i \neq \ell}$ such that we are not given the key $k_{p_i, (i \bmod 2)}$. To complete the proof, we use a hybrid argument. In the first hybrid, we show that the conditional distribution of the elements of A and B is identical to the conditional distribution of A and B', where each element in B' is chosen uniformly and independently from $\{0,1\}^\kappa$. In the second hybrid, we show that the conditional distribution of A and B is identical to the conditional distribution of A' and B', where each element in A' is chosen uniformly and independently from $\{0,1\}^\kappa$.

For the first hybrid, take any $z_i \in B$. Since B contains the elements z_i for which we are not given the associated key $k_{p_i, (i \bmod 2)}$, we can write z_i as $\tilde{z}_{i'} \oplus k_{j', 1-b'_j}$, for some $0 \leq i' \leq \tilde{m}$ and $j' \in [m]$. Now, $k_{j', 1-b'_j}$ was chosen independently and uniformly from $\{0,1\}^k$, and thus, is independent of $\left\{k_{j,b_j}\right\}_{j \in [m]}$. Moreover, z_i is the only element that depends on $k_{j', 1-b'_j}$. We conclude then that z_i is independently and uniformly distributed in $\{0,1\}^\kappa$, and the claim follows.

For the second hybrid, take any $z_i \in A$. We can write $z_i = \tilde{z}_{i'} \oplus k_{j', b_{j'}}$ for some $i' \neq \tilde{\ell}$ and $j' \in [m]$. Then, invoking (5), $\tilde{z}_{i'}$ is uniformly random in $\{0,1\}^\kappa$. Moreover, no other element in A depends on $\tilde{z}'_i$, so we conclude that z_i is independent of all the other elements of A. Finally, since the elements of B' are independent, uniform draws from $\{0,1\}^\kappa$, z_i is independent of all the other elements in A and B'. Thus, the conditional distribution of A and B' is identical to the conditional distribution of A' and B', and the claim holds.

By induction on d, we conclude that for all $i \neq \ell$, the conditional distribution of z_i given $\left\{k_{j,b_j}\right\}_{j \in [m]}$ is uniform over $\{0,1\}^\kappa$, and independent of z_j for all $j \neq i$. $\square$

We now show that the protocol in Figure 4 is one-sided secure, that is, secure against a malicious client and private against a malicious server. Similar to the semi-honest setting, we allow the protocol to reveal the depth d of the decision tree. As before, we assume that the decision trees is complete, and so the number of internal nodes m is given by $2^d - 1$.

Proof of Theorem 4 (Client Privacy). Let $\mathcal{A}$ be a PPT server. We construct a simulator $\mathcal{S}$ that interacts only with the ideal functionality such that the output of the simulator is computationally indistinguishable from the output of $\mathcal{B}$ in the real world. As in the case of the semi-honest protocol, our protocol reveals some basic parameters regarding the size of the decision tree: the depth d of the decision tree, the dimension n of the feature vectors, and the precision t of the components in the feature vector. These values are given to the simulator at the beginning of the protocol execution.

1. The environment gives an input $\mathcal{T}$ to the simulator. In addition, the environment gives the simulator the decision tree parameters d, n, t.

2. The simulator starts running $\mathcal{A}$ on input $\mathcal{T}$.

3. For each $i \in [n]$ and $j \in [t]$, set $\tilde{x}_{i,j} \leftarrow 0$ and $\tilde{r}_{i,j} \xleftarrow{\$} \mathbb{Z}_p$. Construct ciphertexts $\tilde{c}_{i,j} \leftarrow \mathsf{Enc}\left(\tilde{x}_{i,j}; \tilde{r}_{i,j}\right)$ and corresponding proofs
$$\tilde{\pi}_{i,j} \leftarrow \mathsf{PoK}\left\{(r_{i,j}) : \tilde{c}_{i,j} = \mathsf{Enc}\left(0; r_{i,j}\right) \vee \tilde{c}_{i,j} = \mathsf{Enc}\left(1; r_{i,j}\right)\right\}.$$
Note that these are the same Chaum-Pederson style proofs as (3) used in the real protocol (Figure 4). Send the ciphertexts $\tilde{c}_{i,j}$ and proofs $\tilde{\pi}_{i,j}$ to $\mathcal{A}$.

4. Output whatever $\mathcal{A}$ outputs.

The view of the adversary $\mathcal{A}$ when interacting in the real protocol is computationally indistinguishable from its view when interacting with the simulator $\mathcal{S}$. This is easy to see since the view of $\mathcal{A}$ consists only of ciphertexts and the associated zero-knowledge proofs. Thus, by semantic security of the underlying encryption scheme, and the zero-knowledge property of the proofs, we conclude that the view of $\mathcal{A}$ in the two cases is computationally indistinguishable, and thus, the output of $\mathcal{A}$ in the real scheme is computationally indistinguishable from the output of $\mathcal{S}$ in the ideal world. $\square$

Proof of Theorem 5 (Server Security). To simplify the proof, we prove security in a hybrid model where we assume we have access to a trusted party for evaluating the functionality in Figure 3. Denote this functionality f_{comp}. Specifically, we assume that in Step 3 of the protocol in Figure 3, the client and server interact with a trusted party for f_{comp}. Then, by instantiating the trusted party with the protocol in Figure 3, and invoking Theorem 3 along with the sequential composability theorem of [15], we achieve security against a malicious client. Let $\mathcal{A}$ be a PPT client in the real world. We describe an ideal-world simulator $\mathcal{S}$:

1. The environment gives an input x to $\mathcal{S}$, along with the decision tree parameters d, n, t. Set $m \leftarrow 2^d - 1$. The simulator starts running $\mathcal{A}$ on input x.

2. Algorithm $\mathcal{A}$ can abort, in which case, the simulator also aborts. Otherwise, for $i \in [m], j \in [t]$, algorithm $\mathcal{A}$ will send a ciphertext $\mathsf{Enc}(\hat{x}_{i,j}; r_{i,j})$ along with a proof $\pi_{i,j}$ that $\hat{x}_{i,j} \in \{0,1\}$.

3. For each $i \in [m], j \in [t]$, verify the proof $\pi_{i,j}$. If any proof fails to verify, abort the protocol, and output whatever $\mathcal{A}$ outputs. Otherwise, apply the knowledge extractor to $\pi_{i,j}$ to extract the randomness $r_{i,j}$ used to encrypt each $\hat{x}_{i,j}$. If the knowledge extractor fails, then output $\perp$. From this, the simulator learns each bit $\hat{x}_{i,j}$ of each component of the client's input vector. Denote the extracted client input by $\hat{x}$. The simulator sends $\hat{x}$ to the TTP, and receives $\mathcal{T}(\hat{x})$.

4. Choose $\tilde{z}_1, \ldots, \tilde{z}_{m+1} \xleftarrow{\$} \{0,1\}^\kappa$ as well as a random bit-string $\tilde{b}' \xleftarrow{\$} \{0,1\}^m$. For each $i \in [m]$, choose $\tilde{\alpha}_i \xleftarrow{\$} \mathbb{Z}_p$, and compute $\tilde{k}_i \leftarrow \tilde{\alpha}_i P$. Let $i_1, \ldots, i_d$ be the indices of the nodes in the path induced by $\tilde{b}'$ in a complete binary tree of depth d, and let ℓ be the index of the leaf node at the end of the path. Update $\tilde{z}_\ell \leftarrow \mathcal{T}(\hat{x}) \oplus \left(\bigoplus_{i \in [d]} h(\tilde{k}_i)\right)$, where $h : \mathbb{G} \to \{0,1\}^\kappa$ is drawn from the same pairwise independent hash family (as in the real scheme).

5. Send the response vector $[\tilde{z}_1, \ldots, \tilde{z}_{m+1}]$ to $\mathcal{A}$. The simulator also simulates the response of the trusted party for evaluating the extended comparison protocol and gives $\tilde{b}'$ and keys $\tilde{k}_1, \ldots, \tilde{k}_m$ to $\mathcal{A}$.

6. Output whatever $\mathcal{A}$ outputs.

We show that on input x from the environment, the output of the simulator $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$ in the real world. First, we argue that if $\mathcal{A}$ aborts the protocol before sending any messages, then the simulator also aborts, so the behavior of $\mathcal{A}$ and $\mathcal{S}$ is identical up to the end of Step 2 of the simulation. Next, if the simulator aborts the protocol because the proofs fail to verify, then in the real protocol, the server would also abort the protocol (since the proofs are sound). Thus, the behavior of the simulator up to this point is identical to the behavior of the server in the real-world, and so, the outputs of the simulator are identically distributed as the outputs of $\mathcal{A}$.

Next, since the proofs are valid, the knowledge extractor fails to extract the client's query x with negligible probability. Suppose the knowledge extractor successfully extracts the client's query x. Then, we argue that the simulator's message to $\mathcal{A}$ is computationally indistinguishable from the server's message to $\mathcal{A}$ in the real protocol. Consider the view of adversary $\mathcal{A}$ in the real protocol. Recall that we have replaced Step 3 with a call to the ideal functionality for evaluating the extended comparison protocol. Then, the view consists of a bit string $b' \in \{0,1\}^m$, keys $k_{1,b'_1}, \dots, k_{m,b'_m}$, and a blinded response vector $[\hat{z}_1, \dots, \hat{z}_m]$. We consider each of these components:

- In the real protocol, each key k_{i,b'_i} is independently uniform over $\mathbb{G}$. This is the same distribution from which $\mathcal{S}$ samples the simulated keys $\tilde{k}_i$. Thus, the keys are properly distributed.
- Next, consider the distribution of b' in the real protocol. By correctness of the extended comparison protocol, $b_k \oplus b'_k = z_k$, where $z_k = \mathbf{1}\{x_{i_k} < t\}$ is the result of the k^{th} comparison. Equivalently, we can write $b' = b \oplus z$, where $z = z_1 \cdots z_m$. In the real protocol, the server chooses b uniformly at random (and independent of z), so we conclude that b' is uniform over $\{0,1\}^m$. Thus, the distribution of $\tilde{b}'$ is indistinguishable from that of b'.
- Finally, consider the blinding factors used to construct the blinded response vector in the real protocol. Since the keys are statistically close to uniform (by applying the leftover hash lemma [41] to an appropriately chosen pairwise independent family of hash functions), we can invoke Lemma 3 to argue that each component $\{z_i\}_{i \neq \ell}$ is blinded by an independently random element in $\{0,1\}^\kappa$. Thus, the elements $[z_1, \dots, z_{\ell-1}, z_{\ell+1}, z_m]$ of the blinded response vector are independently and uniform over $\{0,1\}^\kappa$. Finally, in the real protocol, if the client queried on input $\hat{x}$, then by correctness of the protocol (Lemma 2), $\hat{z}_\ell = \mathcal{T}(\hat{x}) \oplus \left(\bigoplus_{i \in [d]} k_i\right)$. This is precisely the same distribution from which the simulator sampled the values $[\tilde{z}_1, \dots, \tilde{z}_m]$.

We conclude that the view of adversary $\mathcal{A}$ when interacting in the real protocol is computationally indistinguishable from its view when interacting with the simulator $\mathcal{S}$. Security then follows in the hybrid model. As noted earlier, by instantiating the ideal functionality f_{comp} with the extended comparison protocol in Figure 3, we achieve security against malicious clients. $\square$