Privacy-Preserving Decision Tree Training and Prediction against Malicious Server

Adi Akavia[1*], Max Leibovich[1**], Yehezkel S. Resheff[2], Roey Ron[2], Moni Shahar[3], and Margarita Vald[2,4]

[1] University of Haifa, Israel
{adi.akavia, max.fhe.phd}@gmail.com
[2] Intuit Inc., Israel
{hezi.resheff, roey1rg}@gmail.com
[3] Facebook Inc., Israel
monishahar@gmail.com
[4] Tel Aviv University, Israel
margarita.vald@cs.tau.ac.il

Abstract. Privacy-preserving machine learning enables secure outsourcing of machine learning tasks to an untrusted service provider (server) while preserving the privacy of the user's data (client). Attaining good concrete efficiency for complicated machine learning tasks, such as training decision trees, is one of the challenges in this area. Prior works on privacy-preserving decision trees required the parties to have comparable computational resources, and instructed the client to perform computation proportional to the complexity of the entire task.

In this work we present new protocols for privacy-preserving decision trees, for both training and prediction, achieving the following desirable properties:

1. Efficiency: the client's complexity is independent of the training-set size during training, and of the tree size during prediction.
2. Security: privacy holds against malicious servers.
3. Practical usability: high accuracy, fast prediction, and feasible training demonstrated on standard UCI datasets, encrypted with fully homomorphic encryption.

To the best of our knowledge, our protocols are the first to offer all these properties simultaneously.

The core of our work consists of two technical contributions. First, a new low-degree polynomial approximation for functions, leading to faster protocols for training and prediction on encrypted data. Second, a design of an easy-to-use mechanism for proving privacy against malicious adversaries that is suitable for a wide family of protocols, and in particular, our protocols; this mechanism could be of independent interest.

Keywords: privacy-preserving machine learning · decision trees · training · prediction · fully homomorphic encryption · secure outsourcing

* This work was supported in part by Haifa Center for Cyber Law & Policy, BIU Cyber Center, Tel-Aviv Cyber Center, all in conjunction with the Israel National Cyber Directorate; and The Israel Science Foundation (grant No. 3380/19).

** This work was supported in part by BIU Cyber Center.

1 Introduction

Outsourcing computation to a service provider with powerful platforms and advanced analytic skills is desirable by many organizations, in particular, for tasks that require heavy computation and vast amounts of storage. Example of popular tasks that are being outsourced, are training and prediction of machine learning models. However, outsourcing cleartext data poses a threat to the data confidentiality that may conflict with propriety issues, users privacy, or may even be forbidden by law. The machine learning revolution of the last decade boosted the need for outsourcing computation with privacy guarantees. The field of *privacy-preserving machine learning* is a lively research area addressing this privacy threat by employing secure-computation techniques in order to perform machine learning tasks on an untrusted service provider (server) while preserving the privacy of the user's data (client).

Decision trees and *random forests*, in particular, are among the most popular machine learning techniques, used for numerous applications in healthcare, remote diagnostics, spam filtering, etc. Similarly to other machine learning techniques, decision trees are used for prediction and require training. *Prediction* using a decision tree model is the process of assigning a class label (or a likelihood score for each class label), when given an unlabeled data instance. The complexity of prediction is proportional to the tree depth d. *Training* a decision tree model is the process of producing a decision tree, when given a training dataset of labeled examples. The goal of training is to produce a tree that would yield accurate predictions on new unlabeled data instances. The complexity of training is proportional to the training dataset size n and the tree size m. Similarly, for random forests, training produces an ensemble of trees, and prediction aggregates scores from all trees in the forest to produces a prediction.

In *secure outsourcing* for decision trees, a client wishes to outsource the computation of decision tree based prediction or training, while aided by a server so that: (1) the server does not have any input or output in the functionality; and (2) the server has a vast (but bounded) amount of computational resources. In this setting, the goal is to design protocols that *minimize the client's computation at the server's expense while maintaining client's data privacy.* In particular, it is desirable that the client's complexity would be sub-linear in the time it takes to compute the outsourced task. Furthermore, it is desirable to achieve privacy that holds against malicious servers.

In prior works on secure computation of decision trees prediction and training, the proposed protocols involved parties of *comparable computational resources.* Furthermore, most prior works consider only semi-honest servers.[1] See Section 1.3 and Table 1. In particular, in the client-server settings, both client and server have comparable asymptotic complexity; this limits the deployment of prior protocols on weak clients.

[1] A *semi-honest* adversary follows the protocol's specification but may try to learn additional information from its view of the protocol's transcript. A *malicious* adversary may follow any arbitrary attack strategy. Both adversaries are probabilistic polynomial time.

The question motivating this work is whether there are maliciously secure protocols for decision tree training and prediction with sub-linear client complexity. Moreover, can both client and server exhibit reasonable experimental performance on standard dataset?

1.1 Our Contribution

Our contribution is two-fold, comprised of both our protocols and our proposed proof mechanism, as detailed below.

Contribution 1: decision tree prediction and training protocols. In this work we affirmatively answer the above question by presenting the first protocols for secure computation of decision tree based training and prediction that attain all the following: (1) *sub-linear client complexity*; (2) *privacy against malicious adversaries*; and (3) *practical usability* as demonstrated by our experimental results. See Table 1 and Section 6.

Elaborating on the above, the client's complexity in our prediction protocol is proportional only to the size of the input data instance and the outputted prediction,[2] while being independent of the decision tree size (*cf.* at least proportional to the tree's depth or size in prior works); the client's complexity in our training protocol is proportional to the decision tree size, while being independent of the training set size (*cf.* proportional to the product of tree size and training dataset size in prior works);[3] the privacy in both our protocols holds against malicious adversaries (*cf.* only semi-honest adversaries in prior training protocols). We implemented our protocols and ran benchmarks on standard UCI datasets demonstrating the applicability of our protocol by exhibiting:

- *High accuracy* comparable to training and prediction on cleartext data.
- *Fast prediction* (seconds) on encrypted unlabeled data instances.
- *Feasible training* (minutes to hours) on encrypted training dataset with up to 10,000 examples.

These results can easily be extended to random forests.

To formally address the security guarantee of our protocols, we use a general and simple definition that captures privacy against malicious servers for outsourcing protocols, similar to Definition 2.6.2 in [18]. We refer to protocols satisfying our definition as *privacy-preserving outsourcing protocols.*

[2] To emphasize our protocol's complexity independently of its black-box use of a fully homomorphic encryption, we omit the polynomial overhead in the security parameter due to processing ciphertexts.

[3] We address here the online training phase where the decision tree is constructed, assuming a previously uploaded training dataset of encrypted labeled examples.

Table 1. Comparison of decision tree prediction and training protocols for d and m the tree's depth and size respectively, and n the number of labeled examples in the training set. In row [12], t is the binary representation length of data instances.

Prediction					
Scheme	**Communication**		**Complexity**		**Malicious**
	Rounds	**Bandwidth**	**Client**	**Server**	**Adversary?**
[9,5,8,34,21,25]	constant ≥ 2	$\Omega(m)$	$\Omega(m)$	$\Omega(m)$	✗
[38]	constant ≥ 2	$\Omega(m)$	$\Omega(m)$	$\Omega(m)$	✓
[12]	$t+3$	$\Omega(m)$	$\Omega(m)$	$\Omega(m)$	✗
[35]	$4d$	$\Omega(d^2)$	$\Omega(d^2)$	$\Omega(d^2)$	✗
This Work	**1**	$O(\mathbf{1})$	$O(\mathbf{1})$	$O(m)$	✓

Training				
Scheme	**Communication**		**Complexity of**	**Malicious**
	Rounds	**Bandwidth**	**each Data Owner**	**Adversary?**
Prior Work [28,13,37,39,15,32,36,20,29]	$\Omega(d)$	$\Omega(m \log n)$	$\Omega(mn)$	✗
This Work	d	$O(m)$	$O(m)$	✓

Contribution 2: easy-to-use proof mechanism. We devised an easy-to-use mechanism for proving that an outsourcing protocol is privacy-preserving (see Theorem 1, Section 4) that can be applied on a wide family of outsourcing protocols (see Definition 6, Section 4). We exemplify how to use our proof mechanism in the analysis of our decision tree protocols (see Section 5). We believe this mechanism could be of service to the wider community of software engineers developing privacy-preserving outsourcing protocols in attaining rigorous privacy analysis.

Elaborating on the above, our easy-to-use mechanism for proving security is suitable for any client-server outsourcing protocol, parameterized by family of functions $G = \{G_n\}_{n \in \mathbb{N}}$ over some domain D_n, that uses a fully homomorphic encryption (FHE) scheme $\mathcal{E}$ in the following three-stage structure:

1. *Client's input outsourcing phase*, where client's input is encrypted and outsourced to the server.
2. *Server's computation phase*, where the server performs some local computation and is allowed to interact with the client only by sending $(\mathsf{ciphertext}, n)$, and receiving $\mathsf{Enc}(G_n(\mathsf{Dec}(\mathsf{ciphertext})))$ in response.
3. *Client's output phase*, where the server sends to the client a last message that the client processes to produce the output.

We call such protocols $(\mathcal{E}, G)$*-aided outsourcing protocols.*

Our mechanism for proving a $(\mathcal{E}, G)$-aided outsourcing protocol is privacy-preserving requires proving only the following simple and standard properties on the protocol, the family of functions, and the encryption scheme:

- *Correct protocol:* The client and server are probabilistic polynomial-time algorithms that, when adhering to the protocol specifications, produce the correct output.
- *Length preserving functions:* The family of functions G is polynomial-time computable, and it is length preserving in the sense that $|G_n(x_0)| = |G_n(x_1)|$ for all $n \in \mathbb{N}$ and $x_0, x_1 \in \mathsf{D}_n$.
- *Secure encryption:* The encryption scheme $\mathcal{E}$ is CPA-secure, and circuit-private for the family G.

Our main theorem proves that satisfying the above simple properties implies that the protocol is privacy-preserving; See Theorem 1, Section 4.

1.2 Our Techniques

Soft-step function. As a central component in our protocols, we devised a low-degree polynomial approximation for step functions (aka, *soft-step functions*), by using the least squares method. We then show how to utilize this soft-step function for fast and accurate prediction and training over encrypted data. To achieve better accuracy in our algorithms and protocols, the approximation uses a weighting function that is zero in a window around the step and constant elsewhere. See Sections 3,5, and the overview below.

Protocols overview. Our protocols are composed of three main phases: (1) *Client's data outsourcing phase*, where the client generates keys (pk, sk) for a (leveled) FHE scheme $\mathcal{E}$, publishes pk, keeps sk secret, and then the client (or any other data-source) can use the public key pk to encrypt and upload data to the server. (2) *Server's computation phase*, where the bulk of the computation occurs by the server. In our *prediction* protocol, the server homomorphically evaluates the (possibly, encrypted) decision-tree on the encrypted data instance, and sends the encrypted outcome to the client. In our *training* protocol, this phase is an interactive protocol between the client and server, with d rounds of communication (for d the tree's depth), communication bandwidth and client's complexity $O(ksL \cdot m)$, and server's complexity $O(ksL \cdot m \cdot n)$ (for k the dimension of the labeled data examples, aka, the number of features, L the number of class labels, and s the number of thresholds considered for each tree node). (3) *Client's output phase*, where the client receives the encrypted output from the server, and decrypts to produce her output.

We elaborate on the server's computation phase in our training protocol. The standard approach for decision tree training on cleartext data is to construct the tree layer-by-layer, starting from the root. For each node a feature and threshold are chosen to locally optimize the classification of the training set, as measured by the conditional entropy or the Gini Impurity; we use the latter. This (feature,threshold) selection is composed of two steps. First the training set is processed, based on the tree constructed thus far, to generate a small set of data aggregates. Second, the feature and threshold are selected based on these aggregates. In our privacy-preserving protocol, first the server processes the (encrypted) training data and the (encrypted) tree constructed thus far, to produce the (encrypted) small set of data aggregates, and sends these aggregates to the client. The client decrypts, selects the best choice of (feature,threshold), encodes and encrypts them to be sent back to the server. To reduce round complexity, the tree is constructed in a breath-first-search (BFS) manner, so that aggregates for all nodes in the same layer are all sent together in a single message.

The heart of our novel technique lies in how the server computes the new aggregates. In standard protocols for decision tree training, each node is associated with a test "$x[i] < \theta$", directing to its left child all examples with $x[i] \leq \theta$ and to its right child all those with $x[i] > \theta$. When computing on encrypted data, computing this hard-threshold is computationally demanding. Note that threshold can be viewed as a function accepting value 0 (left) on all values smaller than θ, and 1 (right) otherwise (aka, step function). We propose to replace this threshold with a *soft-step function.* The soft-step function is a low-degree polynomial approximating the step function. Our use of this soft-step function considerably improves the computational overhead when computing on encrypted data, without compromising accuracy. Likewise, we employ the soft-step function in our prediction protocol as our central tool for achieving fast homomorphic evaluation of the decision tree.

Easy-to-use mechanism for proving security. The security guarantees of our protocols are obtained via our easy-to-use mechanism presented in Theorem 1. That is, we prove the following simple properties. *Correctness* of our protocols, when assuming honest behavior. *Length preserving* functions are computed in the interaction with the client, specifically, the client's selection of the best (feature,threshold) during training is length preserving. *Secure encryption* is used in the sense that it is CPA-secure and circuit-private for the family of functions computed by the client. Our proof of Theorem 1 shows that the above three properties imply that our protocols are privacy-preserving.

We prove Theorem 1 in two steps. In the first step, we define a stronger variant of CPA-security that we call $\mathsf{Function\text{-}CPA}$, and show that $\mathsf{Function\text{-}CPA}$ is sufficient to guarantee privacy for $(\mathcal{E}, G)$-aided outsourcing protocols. The notion of $\mathsf{Function\text{-}CPA}$ secure encryption extends the CPA game to include queries to a family of functions $G = \{G_n\}_{n \in \mathbb{N}}$ as follows: in addition to the standard CPA game, the adversary has access to a $\mathsf{Enc}(G(\mathsf{Dec}(\cdot)))$ oracle, that receives queries of the form $(\mathsf{ciphertext}, n)$ and response with $\mathsf{Enc}(G_n(\mathsf{Dec}(\mathsf{ciphertext})))$. The security requirement demands that no efficient adversary can win the $\mathsf{Function\text{-}CPA}$ game with probability noticeably greater than $\frac{1}{2}$. Then, in the second step of the proof, we show that any circuit-private CPA-secure FHE scheme is $\mathsf{Function\text{-}CPA}$ secure.

We note that Theorem 1 can be instantiated directly with any $\mathsf{Function\text{-}CPA}$ secure FHE scheme, avoiding the need for circuit-privacy.

1.3 Prior Work

Secure computation of decision tree based prediction and training in prior works addressed parties of *comparable computational resources.*

For prediction, the prior works [9,5,8,38,12,34,21,25,35] addressed settings where the client's input is an unlabeled example, the server's input is a decision tree, the client's output is the prediction for the input example and the server has no output. The *privacy goal*, informally, is that the server (respectively, client) learns no new information on client's input (resp. server's input). The privacy holds only against *semi-honest* adversaries in all prior work, except Wu *et al.* [38] that prove privacy against *malicious* adversaries as well. The *complexity*, for both client and server, grows asymptotically in the complexity of the tree. See Table 1.

For training, the prior works [28,13,37,39,15,32,36,20,29] addressed *federated learning* against semi-honest adversaries. Namely, the training dataset of n labeled examples is distributed among two or more parties (Data Owners) who engage in a secure multi-party computation to construct a decision tree from the union of their datasets. The *privacy goal*, informally, is that no information leaks on their private dataset, beyond what can be inferred from the trained decision tree. The *complexity*, for every party, is proportional to her input dataset; namely, $\Omega(n)$ complexity (on an evenly distributed training dataset and a constant number of parties). See Table 1.

Function approximation for secure outsourcing. A common approach for FHE based secure machine learning is to approximate functions that are computationally heavy for FHE. This approach is widely used for approximating continuous activation functions, such as $ReLU$, $Sigmoid$ and $Tanh$, with low degree polynomials and utilizing these polynomials in training logistic regression models over encrypted dataset, and recently by [19] in neural networks. In particular, the approximation technique in [24,23,26] for training a logistic regression model considered least squares fitting polynomials (i.e., minimizing MSE). Another technique is the minimax approximation, used in [7,19] with Chebyshev polynomials and in [10] with the Remez Algorithm [30]. Experimental results presented in these works show that approximating with polynomials of degree at most 9 is sufficient for the $ReLU$, $Sigmoid$ and $Tanh$ functions.

Paper organization. Section 2 contains basic definitions and notations. In Section 3 we present a machine learning algorithms for training and prediction of approximated decision trees. In Section 4 we present a simple sufficient conditions on protocols to imply privacy. Section 5 presents privacy-preserving protocols for training and prediction of decision trees. Section 6 presents empirical evaluation of our protocols in terms of accuracy , and runtime. Finally, in Section 7 we conclude with a discussion of open problems.

2 Preliminaries

Throughout the rest of the paper, we use the following notation and definitions. For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \dots, n\}$. A function $g : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if it tends to zero faster than any inverse polynomial, *i.e.*, for all $c \in \mathbb{N}$ there exists $k_c \in \mathbb{N}$ such that for every $k > k_c$ it holds that $g(k) < k^{-c}$. We use $\mathsf{neg}(\cdot)$ to denote a negligible function if we do not need to specify its name. A L-dimensional binary vector $y = (y_1, \ldots, y_L)$ is a *1-hot encoding* of $\ell \in [L]$, if the ℓ'th entry is the only non zero entry in y.

A random variable A is a function from a finite set S to the non-negative reals with the property that $\sum_{s \in S} A(s) = 1$. A probability ensemble $X = \{X(a,n)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by a and $n \in \mathbb{N}$. Two distribution ensembles $X = \{X(a,n)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$ and $Y = \{Y(a,n)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X \approx_c Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function neg such that for every $a \in \{0,1\}^*$ and every $n \in \mathbb{N}$,

$$|\Pr[\mathsf{D}(X(a,n)) = 1] - \Pr[\mathsf{D}(Y(a,n)) = 1]| \leq \mathsf{neg}(n)$$

.

2.1 Decision Trees

A decision tree is a mapping $t : \mathbb{R}^k \to \{1, \ldots, L\}$ implemented as a binary tree, where each internal node corresponds to a partitioning of the input space along one dimension, and leaf nodes have a label (value from $\{1, \ldots, L\}$) associated with them. A tree t is evaluated on an input x by traversing a path in the tree starting from the root, using the partitioning rule at each node to decide how to continue. When a leaf is reached, the label associated with it is returned as $t(x)$.

The structure of a decision tree is typically learned in order to fit to a given dataset – a set of n pairs $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ for which we ideally want to have: $\forall i : t(x^{(i)}) = y^{(i)}$. The task of finding the optimal tree, that is the tree of a given depth for which the maximal number of the aforementioned equalities hold, is known to be NP-complete [27]. Heuristics used in practice to obtain decision trees given a dataset rely on optimizing the local quality of each partitioning (i.e. each node), by selecting the dimension and threshold value that divide the data into partitions that are each "as pure as possible". The motivation behind this local criterion is that if all data points that arrive to the same leaf have the same label, then by assigning this label to the leaf we are able to categorize this region perfectly. Several measures of purity are commonly used, and we describe the Gini impurity measure in greater detail in Section 3.

2.2 CPA-Secure Encryption

We give the standard definition for CPA-security (adapted from [22]).

Consider the following experiment defined for public-key encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and adversary $\mathcal{A}$:
The CPA indistinguishability experiment $\mathsf{EXP}^{cpa}_{\mathcal{A},\mathcal{E}}(\lambda)$:

1. $\mathsf{Gen}(1^\lambda)$ is run to obtain keys (pk, sk).

2. Adversary $\mathcal{A}$ is given pk as well as oracle access to $\mathsf{Enc}_{pk}(\cdot)$. The adversary outputs a pair of messages x_0, x_1 with $|x_0| = |x_1|$. (These messages must be in the plaintext space associated with pk.)
3. A random bit $b \in \{0, 1\}$ is chosen, and the ciphertext $c \leftarrow \mathsf{Enc}(pk, x_b)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext. $\mathcal{A}$ continues to have access to $\mathsf{Enc}_{pk}(\dot{)}$.
4. $\mathcal{A}$ outputs a bit b'.
5. The output of the experiment is define as 1 if $b' = b$, and 0 otherwise.

Definition 1 (CPA-security). *A public key encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable encryptions under chosen-plaintext attacks (or is CPA-secure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function* neg *such that:*

$$\Pr[\mathsf{EXP}^{cpa}_{\mathcal{A},\mathcal{E}}(\lambda) = 1] \le \frac{1}{2} + \mathsf{neg}(\lambda)$$

where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used to generate (pk, sk), choose b, and generate the encryptions.

2.3 Fully Homomorphic Encryption

We give standard definitions for fully-homomorphic encryption (FHE) and circuit privacy (adapted from [6] and [17]).

Definition 2 (public-key FHE scheme). *A homomorphic (public-key) encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *with message space $\mathcal{M}$ is a quadruple of PPT algorithms as follows (λ is the security parameter):*

– *Key generation* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*: Outputs a public encryption key pk and a secret decryption key sk.*
– *Encryption* $c \leftarrow \mathsf{Enc}(pk, \mu)$*: Using the public key pk, encrypts a message $\mu \in \mathcal{M}$ into a ciphertext c.*
– *Decryption* $\mu \leftarrow \mathsf{Dec}(sk, c)$*: Using the secret key sk, decrypts a ciphertext c to recover the message $\mu \in \mathcal{M}$.*
– *Homomorphic evaluation* $\hat{c} \leftarrow \mathsf{Eval}(C, (c_1, \ldots, c_\ell), pk)$*: Using the public key pk, applies a circuit $C : \mathcal{M}^\ell \to \mathcal{M}$ to $c_1, \ldots, c_\ell$, and outputs a ciphertext $\hat{c}$.*

The scheme is said to be secure *if it is CPA-secure. It is* fully homomorphic*, if for any efficiently computable circuit C and any set of inputs to the circuit $x_1, \ldots, x_\ell$, letting $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, and $c_i \leftarrow \mathsf{Enc}(pk, x_i)$, it holds that:*

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(C, (c_1, \ldots, c_\ell), pk)) \ne C(x_1, \ldots, x_\ell)] = \mathsf{neg}(\lambda)$$

The scheme is compact *if the decryption circuit's size only depends on λ.*

Definition 3 (Circuit Private Homomorphic Encryption [16]). *We say that a homomorphic encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is circuit-private for circuits in $\tilde{G}$ if, for any keypair (pk, sk) output by* $\mathsf{Gen}(1^\lambda)$*, any* *circuit $\tilde{G}_i \in \tilde{G}$, and any fixed ciphertexts $\mathsf{c}_1, \ldots, \mathsf{c}_\ell$ that are in the image of* Enc *for plaintexts $x_1, \cdots, x_\ell$, the following distributions (over the random coins in* Enc *and* Eval*) are indistinguishable:*

$$\mathsf{Enc}\big(pk, \tilde{G}_i(x_1, \cdots, x_\ell)\big) \approx \mathsf{Eval}\big(\tilde{G}_i, (\mathsf{c}_1, \ldots, \mathsf{c}_\ell), pk\big)$$

3 Decision Trees with Low Degree Approximation

In this section we present algorithms for training and prediction of decision trees. The algorithms are tailored to being evaluated over encrypted data, in the sense of avoiding complexity bottlenecks of homomorphic evaluation.

The key component in our algorithms is a low degree polynomial approximation for the step function "$x < \theta$" (aka, soft-step function); See Section 3.1. The obtained low degree approximation is used to replace the step function at each tree node in our new prediction and training algorithms, presented in Sections 3.2-3.3.

3.1 Low Degree Approximation of a Step Function

We construct a low-degree polynomial approximation for a step function. Specifically, we consider the step function $I_0 \colon \mathbb{R} \to \{0, 1\}$ with threshold zero, defined by: $I_0(x) = 1$ if $x \geq 0$ and $I_0(x) = 0$ otherwise.

There are several convenient methods for replacing piece-wise continuous functions with limited-degree polynomial approximations. One approach is to starts off with considering the appropriate space of functions as a metric space, and then finding a polynomial of the desired degree that minimizes the deviation from the target function in this metric. Natural choices of metrics are the uniform error, integral square error, and integral absolute error. We aim to replace a step function with a *soft-step function*, *i.e.*, a polynomial approximation. In choosing these polynomials we opt for the mean square integral solution, due to its extendability. That is, the soft-step function would be the solution to the following optimization problem:

$$\phi = \min_{p \in P_n} \int_{-2}^{2} \left(I_0(x) - p(x)\right)^2 \, dx \tag{1}$$

where P_n is the set of polynomial functions of degree at most n over the reals. Setting the interval of the approximation to be $[-2, 2]$ is sufficient once we have pre-processed all data to be in the range $[-1, 1]$. A soft-step at $\theta \in [-1, 1]$ is of the form $\phi(x - \theta)$, and thus $x - \theta \in [-2, 2]$.

However, in many cases the sensitivity to error in the approximation is not uniform in the domain. Errors at an interval around the threshold may harm the overall result of the algorithm less, compared to errors away from the threshold value. Adding an importance-weighting of the approximation interval leads to the following optimization problem:

$$\phi = \min_{p \in P_n} \int_{-2}^{2} \left(I_0(x) - p(x)\right)^2 \, w(x) \, dx \tag{2}$$

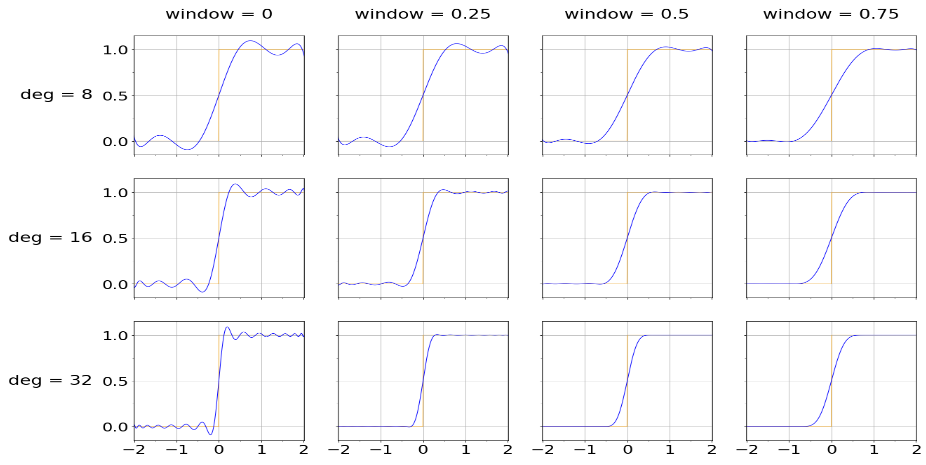

Fig. 1. Polynomial approximations of the step function in $[-2, 2]$, with varying polynomial degrees (rows) and width of neglected window around the transition (columns). We approximate with polynomials that have zero terms in all coefficients of even degree greater than 1.

with a weighting function $w(x) \geq 0 \; \forall x \in [-2, 2]$ and $\int_{-2}^{2} w(x)\, dx = 1$. We note that the unique solution to this problem is obtained by the projection of I_0 onto P_n, in the w-weighted norm (see Chapter II in [31]), or alternatively by applying polynomial regression (minimizing MSE) on a discrete set of points sampled proportionally to w.

Experiments with polynomials which are solutions to (2) with various weight functions and degrees show that by neglecting an interval around the threshold we are able to obtain tighter approximations to the linear phases of the step function, which overall benefit the trees constructed with them. More specifically, by using weighting functions that are zero in a window around the step and constant otherwise, a trade-off is obtained between the slope of the transition and tightness of approximation at the edges of the interval (Figure 1). For larger slopes (smaller neglected windows) the approximation polynomial reaches the $0 - 1$ plateau faster, but at the price of overshooting and oscillations around the linear parts of the step function. While this can be remedied by choosing very high degree polynomial, computational considerations (especially with FHE arithmetic) lead us to favor smaller degree polynomials.

Approximating in L_∞-norm: The problem of polynomial approximation in L_∞-norm is of great importance in signal processing. The standard way of solving such problems is by applying the Remez Algorithm [30]. This approach required higher degree polynomials to obtain the same level of errors in our case, and therefore we chose the L_2 optimization.

3.2 Prediction of Decision Trees with Soft-Step Function

We present our algorithm for prediction of decision trees, where the step function is replaced with its soft-step function counterpart. Moreover, the procedure we describe in this section can also be used as a "recipe" for converting an existing tree model to a polynomial form of low degree. This recipe by itself might be of independent interest.

The methodology described in this section is performed on a given, already trained, tree. Traditionally, a tree model is evaluated on a new example $x \in \mathbb{R}^k$ by traversing a single path from root to leaf according to the input data. Interpreted as a product of indicators, traversal from node v can be written recursively as:

$$\begin{aligned} v.value(x) = I_{v.\theta}(x[v.feature]) \cdot v.right.value(x) + \\ (1 - I_{v.\theta}(x[v.feature])) \cdot v.left.value(x) \end{aligned} \tag{3}$$

where $v.right.value(x)$ and $v.left.value(x)$ are the recursive evaluations of the right and left sub-trees respectively on the input data x. The step function $I_{v.\theta}(x[v.feature])$ is 1 if the statement evaluated at the node is true, and otherwise 0, *i.e.*, if the value of the designated feature specified by $v.feature$ in x is larger than the threshold $v.\theta$. Replacing the step function with its polynomial approximation (*i.e.*, the soft-step function ϕ obtained via Equation 2), this becomes:

$$\begin{aligned} v.value(x) = \phi\left(x[v.feature] - v.\theta\right) \cdot v.right.value(x) + \\ \left(1 - \phi\left(x[v.feature] - v.\theta\right)\right) \cdot v.left.value(x) \end{aligned} \tag{4}$$

Due to the symmetry of $\phi()$ it holds that,

$$\phi\left(v.\theta - x[v.feature]\right) = \left(1 - \phi\left(x[v.feature] - v.\theta\right)\right) \tag{5}$$

Using Equation 4 recursively from the root node, and adding a stopping criterion when reaching a leaf leads to the full tree evaluation procedure, as described in Algorithm 1. Informally, this is a Depth-First-Search that computes a linear combination of the values associated with each of the tree leaves, with coefficients based on the product of soft-step function evaluated along the path leading from the root to that leaf. We note that Algorithm 1 naturally extends to evaluation of base-classifiers for random forests and tree boosting, hence our method supports these common tree-based machine learning methods as well.

3.3 Training Decision Trees with Soft-Step Function

In this section we specify our training algorithm for decision trees. The training algorithm is given a dataset $(\mathcal{X}, \mathcal{Y})$ containing n labeled examples, each $x \in \mathcal{X}$ associated with a label from $[L]$. For $x \in \mathcal{X}$ we will use y_x as shorthand for the

Algorithm 1 Tree Prediction Function

A node v is a data structure containing the following fields: $feature$ and θ denoting the feature and the threshold associated with the node, $leaf\_value$ is the value in case v is a leaf. In addition, $v.right$ and $v.left$ denote the right and left sub-trees of v, respectively. $x \in [-1,1]^k$ is a single data instance.

```
function Tree_Predict($v$, $x$)
    if $v$ is a leaf then
        return $v.leaf\_value$
    else
        return $\phi\big(x[v.feature] - v.\theta\big) \cdot \mathsf{Tree\_Predict}(v.right, x) +$
                $\phi\big(v.\theta - x[v.feature]\big) \cdot \mathsf{Tree\_Predict}(v.left, x)$
    end if
end function
```

label associated with x. The indicator function $\mathsf{isEQ}(y_x, \ell)$ is 1 if y_x is equal to ℓ, and 0 otherwise.

For the root node, the standard training considers the split obtained from using the i-th feature with the threshold θ. The examples in the dataset are divided into two sets: $\{x \in \mathcal{X} \mid x[i] \geq \theta\}$ and $\mathcal{X} \setminus \{x \in \mathcal{X} \mid x[i] \geq \theta\}$. The number of examples $|\{x \in \mathcal{X} \mid x[i] \geq \theta\}| = \sum_{x \in \mathcal{X}} I_\theta(x[i])$. The training procedure aims to build trees with local objectives based on the number of examples of each label that flow to each side at a split, to this end for the right hand side we define:

$$\begin{aligned} \mathsf{right}[i,\theta][\ell] &= \sum_{x \in \mathcal{X}} I_\theta(x[i]) \cdot \mathsf{isEQ}(y_x, \ell) \\ \mathsf{total\_right}[i,\theta] &= \sum_{\ell \in L} \mathsf{right}[i,\theta][\ell] \end{aligned} \tag{6}$$

where the left hand side is defined in similar manner with $(1 - I_\theta(x[i]))$. Note that $\mathsf{right}[i,\theta]$, and $\mathsf{left}[i,\theta]$ are L-dimensional vectors that contain the total number of examples $x \in \mathcal{X}$ of each of the labels that flow to each side of the split.
Based on Equation 6, the weighted Gini Impurity for this split is calculated (the Gini Impurity is weighted by the volume of data that the split sends to each of the children nodes):

$$\tilde{I}_G[i,\theta] = \sum_{side \in \{\mathsf{right},\mathsf{left}\}} \left(1 - \sum_{\ell \in L} \left[\frac{\mathsf{side}[i,\theta][\ell]}{\mathsf{total\_side}[i,\theta]}\right]^2\right) \cdot \mathsf{total\_side}[i,\theta] \tag{7}$$

At each node in the training procedure, the feature and threshold are chosen in order to minimize the weighted Gini impurity of the resulting split. In order to minimize the number of splits to consider, a common approach is to select the possible threshold values on a grid. A scaling pre-processing procedure guarantees that all features value are within the correct range.

The split finding procedure proceeds as follows: for each feature i and each candidate threshold θ on the grid, compute $\tilde{I}_G[i,\theta]$, and select the best feature and threshold, *i.e.*, those that minimize $\tilde{I}_G[i,\theta]$. Once the best feature and threshold value are determined, denoted by i^* and θ^*, we can move on to the successive sub-trees. The training procedure associates with each node a set of indicators $W = \{w_x\}_{x\in\mathcal{X}}$, so that w_x is a bit indicating if example x is participating in the training of a sub-tree rooted at this node, and W is updated for the children nodes as follows:

$$\forall x \in \mathcal{X} : w_x^{\mathsf{right}} = w_x \cdot I_{\theta^*}(x[i^*]) \tag{8}$$

is the set of indicators of the right-hand sub-tree, since w_x^{right} is 1 iff both w_x is 1 (meaning this is an example x that was considered in the current sub-tree), and the example x flows right in this split. Likewise, it is defined for the left-hand sub-tree. Therefore, the analog of Equation 6 for training internal nodes is:

$$\begin{aligned}\mathsf{right}[i,\theta][\ell] &= \sum_{x\in\mathcal{X}} w_x \cdot I_\theta(x[i]) \cdot \mathsf{isEQ}(y_x,\ell)\\ \mathsf{left}[i,\theta][\ell] &= \sum_{x\in\mathcal{X}} w_x \cdot (1 - I_\theta(x[i])) \cdot \mathsf{isEQ}(y_x,\ell)\end{aligned} \tag{9}$$

In our approach, we avoid the (expensive) comparison operation by replacing the step function $I_\theta(\cdot)$ with the low-degree polynomial approximation $\phi(\cdot)$ in Equations 6, 7,8, 9, see Algorithm 2 for details.

Notice that our approximated version of Equation 8 has real valued weights instead of boolean indicators. This means that *every example* reaches every node, and is evaluated at all nodes. Rather than hard splitting the data (partitioning to right and left children at each split) we have a soft partition of the data, where the two children nodes get a part of each data point, weighted differently (but with weights that sum to 1). In order to efficiently keep track of the weight of each data example at each node, we keep a weights set W while traversing the tree during training. All weights are initialized to 1, and recursively multiplied by the polynomial approximation at the current node before passing on to the children nodes. The details of the training algorithm are presented in Algorithm 2.

4 Privacy-Preserving Outsourcing

In this section we present our main theorem, that provides an easy mechanism for proving security of outsourcing protocols. As an intermediate step towards establishing our mechanism, we identify and formally define a stronger notion of secure public-key encryption called *Function-CPA* (that might be of independent interest), and show that FHE encryption schemes satisfy it. In addition, we present a definition of privacy-preserving outsourcing computation in Section 4.1 that is similar to Definition 2.6.2 of [18], but with adjustment to the outsourcing setting.

Algorithm 2 Tree Training Function

$(\mathcal{X},\mathcal{Y})$ is a dataset and W is a set of weights, s.t. $\forall x \in \mathcal{X}$: $w_x \in \mathbb{R}$ and $y_x \in [L]$ are the weight (initially 1) and the label associated with the example $x \in [-1,1]^k$, respectively. We denote by k and L the number of features and labels in the associated problem, respectively. We denote by S the set of considered thresholds. The parameter $\mathsf{maximal\,depth}$ is the depth of the trained tree, the variable *depth* is initialized to 1. The function $\mathsf{Gini}(\cdot)$ in Figure 2 computes the weighted Gini impurity and return the best threshold and feature.

```
function Tree_Train($(\mathcal{X},\mathcal{Y}), w, depth$)
    if reached maximal depth then                                        ▷ leaf node
        $label \leftarrow \arg\max_{\ell\in[L]} \sum_{x\in\mathcal{X}} w_x \cdot \mathsf{isEQ}(y_x,\ell)$
    else
        for each feature $i$ do                              ▷ search best split for this node
            for each threshold $\theta$ do
                for each label $\ell$ do
                    $\mathsf{right}[i,\theta][\ell] \leftarrow \sum_{x\in\mathcal{X}} w_x \cdot \phi(x[i]-\theta) \cdot \mathsf{isEQ}(y_x,\ell)$
                    $\mathsf{left}[i,\theta][\ell] \leftarrow \sum_{x\in\mathcal{X}} w_x \cdot \phi(\theta-x[i]) \cdot \mathsf{isEQ}(y_x,\ell)$
                end for
            end for
        end for
        $i^*,\theta^* \leftarrow \mathsf{Gini}(\{\mathsf{right}[i,\theta],\mathsf{left}[i,\theta]\}_{i\in[k],\theta\in S})$         ▷ See Figure 2

        $\forall x \in \mathcal{X} : w_x^{\mathsf{right}} \leftarrow w_x \cdot \phi(x[i^*]-\theta^*)$
        Tree_Train($(\mathcal{X},\mathcal{Y}), \{w_x^{\mathsf{right}}\}_{x\in\mathcal{X}}, depth+1$)          ▷ build right-side sub-tree

        $\forall x \in \mathcal{X} : w_x^{\mathsf{left}} \leftarrow w_x \cdot \phi(\theta^*-x[i^*])$
        Tree_Train($(\mathcal{X},\mathcal{Y}), \{w_x^{\mathsf{left}}\}_{x\in\mathcal{X}}, depth+1$)           ▷ build left-side sub-tree
    end if
end function
```

4.1 Privacy-Preserving Outsourcing Computation

We consider a natural setting of the standard models for secure computation called the *secure outsourcing model*. In particular, we have a server, that might be malicious, who interacts with a client and performs some computation for it.

The model is motivated by settings where a weak client, denoted by Clnt, delegates some computation to a powerful server Srv. In contrast to classical delegation, where integrity of computation is the main concern, our setting focuses on protecting client's private information from the server who wishes to learn as much as possible about the client's input.

Secure outsourcing protocols have an analogous motivation to zero-knowledge proofs. Zero-knowledge proofs guarantee the privacy of the prover's secret information, while enabling the prover to prove statements about this information. Similarly, secure outsourcing protocols aim to guarantee privacy of client's data, while enabling the server to perform heavy computation related to this client's

Procedure: Gini impurity computation.

We denote by k and L the number of features and labels in the associated problem, respectively. We denote by S the set of considered thresholds. The function computes the weighted Gini impurity and returns the best threshold and feature.
Given a set of L-dimensional vectors $\{\mathsf{right}[i,\theta], \mathsf{left}[i,\theta]\}_{i\in[k],\theta\in S}$ proceed as follows:

1. for each threshold $\theta \in S$ and each feature $i \in [k]$ compute

$$\mathsf{total\_right}[i,\theta] \leftarrow \sum_{\ell\in L} \mathsf{right}[i,\theta][\ell]$$

$$\mathsf{total\_left}[i,\theta] \leftarrow \sum_{\ell\in L} \mathsf{left}[i,\theta][\ell]$$

$$\begin{aligned}\tilde{I}_G[i,\theta] \leftarrow & \left(1 - \sum_{\ell\in L}\left[\frac{\mathsf{right}[i,\theta][\ell]}{\mathsf{total\_right}[i,\theta]}\right]^2\right)\cdot \mathsf{total\_right}[i,\theta] \\ & + \left(1 - \sum_{\ell\in L}\left[\frac{\mathsf{left}[i,\theta][\ell]}{\mathsf{total\_left}[i,\theta]}\right]^2\right)\cdot \mathsf{total\_left}[i,\theta]\end{aligned}$$

2. compute the selected feature and threshold at the node, *i.e.*, return

$$i^*, \theta^* \leftarrow \arg\min_{i,\theta} \tilde{I}_G[i,\theta]$$

Fig. 2. The weighted Gini impurity computation.

data. We capture client's data privacy via the inability of any server to distinguish executions done over different client's data. More formally:

Interactive Outsourcing Protocols. An interactive outsourcing protocol $\langle\mathsf{Clnt}, \mathsf{Srv}\rangle$ for computing a function F proceeds in the following manner:

– At the beginning of the protocol, Clnt and Srv receive the security parameter, and in addition Clnt receives input x and Srv has no input.
– Then Clnt and Srv execute the protocol, and upon receiving the last message from Srv, Clnt produces an output y, and Srv has no output.

An execution of this protocol with client's input x is denoted throughout this paper by $\langle\mathsf{Clnt}(x), \mathsf{Srv}\rangle$. A interactive outsourcing protocol is a protocol that satisfies the completeness and privacy conditions defined below.

Definition 4 (View and output). *Let $\langle\mathsf{Clnt}, \mathsf{Srv}\rangle$ be an interactive outsourcing protocol. Then:*

– Srv*'s view of $\langle\mathsf{Clnt}(x), \mathsf{Srv}\rangle$ is the random variable $View_{\mathsf{Srv}}(\mathsf{Clnt}(x), \mathsf{Srv}) = (m_1, \ldots, m_t; r)$ consisting of all the messages $m_1, \ldots, m_t$ exchanged between* Clnt *and* Srv *together with the string r containing all the random bits that* Srv *has read during the interaction.*

– Clnt*'s output in* $\langle \mathsf{Clnt}(x), \mathsf{Srv}\rangle$ *is a random variable denoted by* $Out_{\mathsf{Clnt}}(\mathsf{Clnt}(x), \mathsf{Srv})$.

Formally, we define privacy for outsourcing computation as follows:

Definition 5 (Privacy-preserving outsourcing protocol). *An interactive outsourcing protocol* $\langle \mathsf{Clnt}, \mathsf{Srv}\rangle$ *is* privacy-preserving *for a function* $F : \mathsf{A} \rightarrow \mathsf{B}$ *if* Srv *and* Clnt *are* PPT *machines and there exists a negligible function* $\mathsf{neg}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds:*

Completeness: *For all* $x \in \mathsf{A}$,

$$\Pr[Out_{\mathsf{Clnt}}(\mathsf{Clnt}(x), \mathsf{Srv})(1^{\lambda}) = F(x)] = 1 - \mathsf{neg}(\lambda).$$

Privacy: *For all* $x_0, x_1 \in \mathsf{A}$ *with* $|x_0| = |x_1|$*, for any* PPT *distinguisher* $\mathcal{D}$ *and every* PPT *server* S^* *it holds that:*

$$|\Pr[\mathcal{D}(View_{S^*}(\mathsf{Clnt}(x_0), S^*)(1^{\lambda})) = 1] - \Pr[\mathcal{D}(View_{S^*}(\mathsf{Clnt}(x_1), S^*)(1^{\lambda})) = 1]| \leq \mathsf{neg}(\lambda)$$

where the probability is taken over the random coins of Clnt *and* Srv.

We note that although Definition 5 is defined with respect to deterministic functions, it can be easily extended to handle randomized functionalities. In Sections 5.1 and 5.2 we present our protocols for prediction and training for tree based models and prove their security in terms privacy-preserving outsourcing protocols.

4.2 An Easy-to-Use Mechanism

In this section we present our main theorem, a mechanism for proving privacy for a broad family of protocols, we call this family *function-aided outsourcing protocols*. We present simple sufficient conditions on protocols for being privacy-preserving outsourcing protocols. We first define *function-aided outsourcing protocols* as follows:

Definition 6 (($\mathcal{E}, G$)-aided outsourcing protocol). *Let* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme with a message space* $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$*. Let* $G = \{G_n : \mathsf{D}_n \rightarrow \mathcal{M}\}_{n\in\mathbb{N}}$ *be a family of functions. An interactive outsourcing protocol* $\Pi = \langle \mathsf{Clnt}, \mathsf{Srv}\rangle$ *for computing a function* F *is called* $(\mathcal{E}, G)$-aided outsourcing protocol *if it has the following three stage structure:*

1. ***Client's input outsourcing phase:*** Clnt *runs* $(sk, pk) \leftarrow \mathsf{Gen}(1^{\lambda})$*, and encrypts its input* x*, i.e.,* $\mathbf{c_x} \leftarrow \mathsf{Enc}_{pk}(x)$*. Then* Clnt *sends* $\mathbf{c_x}$ *and* pk *to* Srv.
2. ***Server's computation phase:*** Srv *performs some computation, and in addition it may interact with* Clnt *by sending it tuples of the form* $(\mathbf{e}, i)$ *(where* $\mathbf{e}$ *is an encryption and* $i \in \mathbb{N}$*) and receiving in response* $\mathsf{Enc}_{pk}(G_i(\mathsf{Dec}_{sk}(\mathbf{e})))$*. In case of an error the response is* $\mathsf{Enc}_{pk}(G_i(x))$ *for an arbitrary* $x \leftarrow \mathsf{D}_i$.
3. ***Client's output phase:*** *Server sends to Client the last message of the protocol. Upon receiving this message, the Client produces an output.*

To formally state our main theorem, we first need to define the class of *admissible* functions for *function-aided outsourcing protocols.*

Definition 7 (Admissible functions). *A family of functions $G = \{G_n \colon \mathsf{D}_n \to \mathcal{M}\}_{n \in \mathbb{N}}$ is called* admissible *if for all $n \in \mathbb{N}$ the following holds:*

- *G_n is polynomial-time computable by a Turing machine M.*
- *$|G_n(x_0)| = |G_n(x_1)|$ for all $x_0, x_1 \in \mathsf{D}_n$.*
- *for all $x \in \mathsf{D}_n$ it holds that $|x| = n$.*

Let $\tilde{G}$ denote the class of polynomial-size circuits corresponding to G. In more detail, defining $\tilde{G}$ is motivated for the purpose of later employing the FHE notion of circuit-privacy. The correspondence here is in the sense that for every $n \in \mathbb{N}$ and $x \in \mathsf{D}_n$, $G_n(x) = \tilde{G}_n(x)$. Finally, $\tilde{G}$ is efficiently computable given a Turing machine for G due to classical results in complexity theory; See [4].

Theorem 1 (Main theorem). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an FHE encryption scheme with a message space $\mathcal{M} = \{\mathcal{M}_n\}_{n \in \mathbb{N}}$. Let $G = \{G_n \colon \mathsf{D}_n \to \mathcal{M}\}_{n \in \mathbb{N}}$ be a family of functions. Let Π be $(\mathcal{E}, G)$-aided outsourcing protocol for a function $F : \mathsf{A} \to \mathsf{B}$. Π is a privacy-preserving outsourcing protocol for F if the following holds:*

1. *Π has PPT Client and Server and satisfies completeness with respect to F as defined in Definition 5.*
2. *G is family of admissible functions.*
3. *$\mathcal{E}$ is CPA-secure encryption scheme that satisfies circuit-privacy for circuits in $\tilde{G}$.*

For the proof of our main theorem, we define and a new security property of public-key encryption, called *function-chosen-plaintext attack* (or $\mathsf{Function\text{-}CPA}$ security). This notion aims at capturing a stronger variant of CPA-security where the adversary is allowed to query not only encryption queries but also decrypt-function-encrypt queries. Concretely, the security is defined with respect to a family of functions, where the adversary may submit a ciphertext together with a function identifier and receive in response an encryption that is produced as follows: the submitted encryption is first decrypted, then the requested function is calculated on the plaintext and the result is being encrypted and returned to the adversary. More formally, we define the $\mathsf{Function\text{-}CPA}$ security via $\mathsf{Function\text{-}CPA}$ experiment as follows:

Given a public-key encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with a message space $\mathcal{M} = \{\mathcal{M}_n\}_{n \in \mathbb{N}}$, a family of admissible functions $G = \{G_n \colon \to \mathcal{M}\}_{n \in \mathbb{N}}$, and an adversary $\mathcal{A}$, consider the following experiment:

Function-CPA indistinguishability experiment $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda)$:

1. $\mathsf{Gen}(1^\lambda)$ is run to obtain a key-pair (pk, sk)

2. Adversary $\mathcal{A}$ is given pk as well as oracle access to an encryption oracle $\mathsf{Enc}_{pk}(\cdot)$ and a decryption-function-encryption oracle $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$.
   The queries to the $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ oracle are pairs of the form: a ciphertext $\mathbf{e}$ and a function index i; the response returned from the oracle is $\mathbf{e}' \leftarrow \mathsf{Enc}_{pk}(G_i(\mathsf{Dec}_{sk}(\mathbf{e})))$ (in case of an error, the oracle returns the encryption of $G_i(x)$ for an arbitrary $x \leftarrow \mathsf{D}_i$).
   The adversary outputs a pair of messages $x_0, x_1 \in \mathcal{M}$ with $|x_0| = |x_1|$.
3. A random bit $b \in \{0,1\}$ is chosen, and then the ciphertext $\mathbf{c} \leftarrow \mathsf{Enc}_{pk}(x_b)$ is computed and given to $\mathcal{A}$. We call $\mathbf{c}$ the challenge ciphertext. $\mathcal{A}$ continues to have access to $\mathsf{Enc}_{pk}(\cdot)$ and $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$.
4. $\mathcal{A}$ outputs a bit b' .
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition 8 (Function-CPA). *A public-key encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with a message space $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$ is* Function-CPA secure *with respect to a family of admissible functions $G = \{G_n : \mathsf{D}_n \to \mathcal{M}\}_{n\in\mathbb{N}}$ if for all* PPT *adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{neg}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, the following holds:*

$$\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{neg}(\lambda).$$

where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used to generate (pk, sk), choose b, and the encryptions.

Now we are ready to show that any CPA-secure circuit private FHE encryption scheme is also Function-CPA-secure.

Theorem 2. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a CPA-secure* FHE *encryption scheme with a message space $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$. Let $G = \{G_n \colon \mathsf{D}_n \to \mathcal{M}\}_{n\in\mathbb{N}}$ be family of admissible functions. If $\mathcal{E}$ satisfies circuit-privacy for circuits in $\tilde{G}$ then $\mathcal{E}$ is Function-CPA secure public-key encryption scheme with respect G.*

Proof. Given a family of admissible functions $G = \{G_n \colon \mathsf{D}_n \to \mathcal{M}\}_{n\in\mathbb{N}}$. Given a CPA-secure FHE encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with message space $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$ and satisfying circuit-privacy for circuits in $\tilde{G}$.

Let $\mathcal{A}$ be some PPT adversary for $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}$. We construct an adversary $\mathcal{A}_{cpa}$ for $\mathsf{EXP}^{cpa}_{\mathcal{A}_{cpa},\mathcal{E}}$ that behaves as follows: The adversary $\mathcal{A}_{cpa}$ will run internally $\mathcal{A}$ and relay messages between the challenger and $\mathcal{A}$, with the exception that $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ queries are answered using Eval. That is, $\mathcal{A}_{cpa}$ does the following:

– upon receiving pk from challenger, forward it to $\mathcal{A}$.
– any $\mathsf{Enc}_{pk}(\cdot)$ type query from $\mathcal{A}$ is redirected to the challenger and the response is given back to $\mathcal{A}$. Any $(\mathbf{e}, i)$ tuple for $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ query is answered by computing $\mathbf{e}' \leftarrow \mathsf{Eval}_{pk}\big(\tilde{G}_i, \mathbf{e}\big)$ (if Eval fails encrypt $G_i(x)$ for an arbitrary $x \leftarrow \mathsf{D}_i$).
– once $\mathcal{A}$ generates x_0, x_1 forward them to the challenger and return the response $\mathbf{c} \leftarrow Enc_{pk}(x_b)$ to $\mathcal{A}$.

– output the b' that $\mathcal{A}$ outputs.

The adversary $\mathcal{A}_{cpa}$ is PPT due to $\mathcal{A}$ being a PPT adversary, $\tilde{G}$ an efficiently computable family of circuits, and the efficiency of Eval.

Note that all the interaction of $\mathcal{A}$ is perfectly simulated besides the queries to $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ that are simulated using Eval. However, circuit privacy of $\mathcal{E}$ guarantees that the response is indistinguishable from decrypting, applying G_i and encrypting the result. More formally, we define a series of hybrid executions that gradually move between $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}$ experiment (where $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ oracle is used) to $\mathsf{EXP}^{cpa}_{\mathcal{A}_{cpa},\mathcal{E}}$ experiment (where Eval is used). Let q denote an upper bound on the number of queries done by $\mathcal{A}$, we define $q+2$ hybrids as follows:

Hybrid $\mathsf{H}_{\textsf{Function-CPA}}$ is defined as the execution of $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}$.

Hybrid H_0 is defined as Hybrid $\mathsf{H}_{\textsf{Function-CPA}}$ with $\mathsf{Enc}_{pk}(\tilde{G}(\mathsf{Dec}_{sk}(\cdot)))$ oracle using circuit family $\tilde{G}$ instead of function family G. We denote this experiment as $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},\tilde{G}}$.

Hybrid H_i is defined for $i \in [q]$. The hybrid H_i is defined as $\mathsf{EXP}^{Fcpa}_{\mathcal{A}_i,\mathcal{E},\tilde{G}}$, where $\mathcal{A}_i$'s last i queries are answered using Eval instead of oracle $\mathsf{Enc}_{pk}(\tilde{G}(\mathsf{Dec}_{sk}(\cdot)))$.

Note that Hybrid H_q is equivalent to the CPA-experiment $\mathsf{EXP}^{cpa}_{\mathcal{A}_{cpa},\mathcal{E}}$.

We will show that each consecutive pair of hybrids are indistinguishable and, since we have only polynomially many hybrids, the indistinguishability will follow. The only difference between $\mathsf{H}_{\textsf{Function-CPA}}$ and H_0 is the use of circuits $\tilde{G}$ in H_0 instead of functions G in $\mathsf{H}_{\textsf{Function-CPA}}$. By the construction of $\tilde{G}$, each function in G is replaced with an equivalent circuit (in term of functionality) and used instead of the function in H_0. Therefore,

$$\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda)=1] = \Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},\tilde{G}}(\lambda)=1].$$

In each pair of adjacent hybrids H_{i-1} and H_i for $i \in [q]$ the difference is that in H_i the i'th query is done using Eval instead $\mathsf{Enc}_{pk}(\tilde{G}(\mathsf{Dec}_{sk}(\cdot)))$ oracle. In this case the indistinguishability follows from $\mathcal{E}$ being circuit private for $\tilde{G}$. Therefore,

$$|\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A}_i,\mathcal{E},\tilde{G}}(\lambda)=1] - \Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A}_{i-1},\mathcal{E},\tilde{G}}(\lambda)=1]| \le \mathsf{neg}(\lambda).$$

Since q is polynomial in λ and $\Pr[\mathsf{EXP}^{cpa}_{\mathcal{A}_{cpa},\mathcal{E}}(\lambda)=1] = \Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A}_q,\mathcal{E},\tilde{G}}(\lambda)=1]$ we conclude that:

$$\Pr[\mathsf{EXP}^{cpa}_{\mathcal{A}_{cpa},\mathcal{E}}(\lambda)=1] \le \frac{1}{2} + \mathsf{neg}(\lambda)$$

As required.

Proof (Proof of Theorem 1). By Theorem 2 it follows that $\mathcal{E}$ is $\mathsf{Function\text{-}CPA}$ secure with respect to G. For Theorem 2 to follow it is sufficient to show that

$\mathit{\Pi}$ satisfies the privacy condition of Definition 5. Assume by contradiction that privacy does not hold for $\mathit{\Pi}$. That is, there exists efficiently samplable $x_0, x_1 \in \mathsf{A}$ with $|x_0| = |x_1|$, a PPT distinguisher $\mathcal{D}$, a malicious server S, and a polynomial $p(\cdot)$ such that:

$$\begin{aligned}&\Pr[\mathcal{D}(View_S(\mathsf{Clnt}(x_1), S)(1^\lambda)) = 1]\\ &- \Pr[\mathcal{D}(View_S(\mathsf{Clnt}(x_0), S)(1^\lambda)) = 1] \geq p(\lambda)\ .\end{aligned} \tag{10}$$

We construct an adversary $\mathcal{A}$ that violates the $\mathsf{Function\text{-}CPA}$ security of $\mathcal{E}$. That is, $\mathcal{A}$ participates in $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}$ as follows:

1. upon receiving pk output x_0, x_1.
2. upon receiving $\mathbf{c}_x \leftarrow \mathsf{Enc}_{pk}(x_b)$ behave exactly as S behaves while executing $\mathit{\Pi}$ starting from the *Client's input outsourcing phase* on $(\mathbf{c}_x, pk)$ from Clnt. Every intermediate outgoing message $(\mathbf{e}, i)$ to Clnt, is redirected to $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ oracle and the response is treated as if it was coming from Clnt.
3. let $S_\mathcal{A}$ denote the part of $\mathcal{A}$ associated with the execution of $\mathit{\Pi}$ (*i.e.*, behaving as the adversary S), and let $View_{S_\mathcal{A}}$ be its view.
4. run the distinguisher $\mathcal{D}$ on $View_{S_\mathcal{A}}$ and output whatever $\mathcal{D}$ outputs.

Adversary $\mathcal{A}$ is PPT due to x_0, x_1 being efficiently samplable and S and $\mathcal{D}$ being PPT.

We denote by $\langle \mathsf{Chal}(x_0), \mathcal{A}\rangle$ the execution of $\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}$ with bit $b = 0$ being selected by the challenger (similarly for $b = 1$). Note that the interaction of $\mathcal{A}$ with $\mathsf{Enc}_{pk}(G(\mathsf{Dec}_{sk}(\cdot)))$ oracle is identical to the interaction of S with Clnt in Step 2 of $\mathit{\Pi}$. Moreover, since $\mathcal{A}$ behaves exactly as S in $\mathit{\Pi}$ it holds that for every $b^* \in \{0, 1\}$,

$$\Pr[\mathcal{D}(View_S(\mathsf{Clnt}(x_{b^*}), S)(1^\lambda)) = 1] = \Pr[\mathcal{D}(View_{S_\mathcal{A}}(\mathsf{Chal}(x_{b^*}), \mathcal{A})) = 1] \tag{11}$$

From Equations 10 and 11 it follows that:

$$\begin{aligned}&\Pr[\mathcal{D}(View_{S_\mathcal{A}}(\mathsf{Chal}(x_1), \mathcal{A})(1^\lambda)) = 1]\\ &- \Pr[\mathcal{D}(View_{S_\mathcal{A}}(\mathsf{Chal}(x_0), \mathcal{A})(1^\lambda)) = 1] \geq p(\lambda)\end{aligned} \tag{12}$$

Therefore, we obtain that:

$$
\begin{aligned}
&\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda)=1]\\
&\quad=\frac{1}{2}\cdot\big(\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda)=1|b=1]+\Pr[\mathsf{EXP}^{Fcpa}_{\mathcal{A},\mathcal{E},G}(\lambda)=1|b=0]\big)\\
&\quad=\frac{1}{2}\cdot\big(\Pr[\mathcal{D}(View_{S_{\mathcal{A}}}(\mathsf{Chal}(x_1),\mathcal{A})(1^{\lambda}))=1]+\Pr[\mathcal{D}(View_{S_{\mathcal{A}}}(\mathsf{Chal}(x_0),\mathcal{A})(1^{\lambda}))=0]\big)\\
&\quad=\frac{1}{2}+\frac{1}{2}\cdot\big(\Pr[\mathcal{D}(View_{S_{\mathcal{A}}}(\mathsf{Chal}(x_1),\mathcal{A})(1^{\lambda}))=1]-\Pr[\mathcal{D}(View_{S_{\mathcal{A}}}(\mathsf{Chal}(x_0),\mathcal{A})(1^{\lambda}))=1]\big)\\
&\quad\geq\frac{1}{2}+\frac{1}{2}\cdot p(\lambda)
\end{aligned}
$$

Where the last inequality follows from Equation 12. Combining this with $\mathcal{A}$ being PPT we derive a contradiction to $\mathcal{E}$ being Function-CPA secure with respect to G.

5 Prediction and Training on Encrypted Data

In this section we present our secure protocols for prediction and training of tree based models. The protocols are an adaptation of Algorithms 1, 2 in Section 3 to the interactive setting. We easily derive the security of our protocols using Theorem 1.

5.1 Decision-Tree based Prediction over Encrypted Data

In this section we show how the tree prediction algorithm (Algorithm 1) can be executed as a protocol between a client and a server on client's input data instance, while maintaining client's input privacy. First, we present our privacy-preserving outsourcing protocol for prediction on cleartext trees (Figure 3). Then we extend our protocol to handle encrypted trees, providing privacy guarantee for both the data instance and the tree model. We note that our protocol can be executed on any tree based model such as Random Forest or Boosted Tree algorithms for instance.

We show that the Protocol $\mathsf{P}=\langle\mathsf{Clnt}_{\mathsf{P}},\mathsf{Srv}_{\mathsf{P}}\rangle$ (Figure 3) is a privacy preserving outsourcing protocol for *Tree Prediction Function* (Algorithm 1). More formally,

Theorem 3. *If* $\mathcal{E}=(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec},\mathsf{Eval})$ *is a CPA-secure* FHE *encryption scheme, then the protocol* $\mathsf{P}=\langle\mathsf{Clnt}_{\mathsf{P}},\mathsf{Srv}_{\mathsf{P}}\rangle$ *(Figure 3) is single-round privacy-preserving outsourcing protocol for* Tree Prediction Function *(Algorithm 1).*

Proof. We use Theorem 1 to prove that $\mathsf{P}=\langle\mathsf{Clnt}_{\mathsf{P}},\mathsf{Srv}_{\mathsf{P}}\rangle$ is privacy-preserving outsourcing protocol. The protocol $\mathsf{P}=\langle\mathsf{Clnt}_{\mathsf{P}},\mathsf{Srv}_{\mathsf{P}}\rangle$ is $(\mathcal{E},G)$-aided outsourcing protocol for *Tree Prediction Function*, with $G=\emptyset$. Therefore, G is admissible

Shared parameters: Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be FHE encryption scheme and let ϕ be the soft-step function from Equation 1 in Section 3. We denote by k and L the number of features and labels in the associated problem, respectively.
A tree $\mathsf{T} = (V, E)$ and security parameter λ. In the tree T each node $v \in V$ is a data structure containing the following fields: $feature$ and θ denoting the feature and the threshold associated with the node, and $leaf\_value$ that is a L-dimension vector of values in $\mathbb{R}$ if v is a leaf and $\perp$ otherwise. In addition, each node has $v.right$ and $v.left$ that denote the right and left children of v, respectively.,
Client's input: a normalized data instance $x \in [-1, 1]^k$ where k is the number of features associated with the instance.
Client's output: a label $\ell \in L$.

The protocol $\mathsf{P} = \langle \mathsf{Clnt_P}, \mathsf{Srv_P} \rangle$ for prediction over trees proceeds as follows:

1. **Client's input outsourcing phase:**
   (a) $\mathsf{Clnt_P}$ runs $\mathsf{Gen}(1^\lambda)$ to obtain a secret key, public key pair (sk, pk).
   (b) $\mathsf{Clnt_P}$ encrypts each entry in x and obtains ciphertext vector $\mathbf{c_x}$. Then $\mathbf{c_x}$ is sent together with pk to $\mathsf{Srv_P}$.
2. **Server's computation phase:** the server $\mathsf{Srv_P}$ recursively computes the subroutine $\mathsf{Enc\_Predict}(v, \mathbf{c_x})$ in Figure 4 starting from the root node. Let **res** be the recursion result, *i.e.*, a L-dimensional vector **res** returned at the root.
3. **Client's output phase:** $\mathsf{Srv_P}$ sends **res** to $\mathsf{Clnt_P}$, who decrypts it and outputs $label \leftarrow \arg\max_{\ell \in L} \mathsf{p\_res}_\ell$ (where $\mathsf{p\_res}$ is the decryption **res**).

Fig. 3. The prediction protocol $\mathsf{P} = \langle \mathsf{Clnt_P}, \mathsf{Srv_P} \rangle$ for trees. The protocol instructs the sever to traverse the tree and homomorphically calculate a value for each path in the tree, and returns to client the weighted sum of the leafs. The predicted label for the input is the arg max of the weighted sum.

Subroutine $\mathsf{Enc\_Predict}(v, \mathbf{c_x})$ where $\mathbf{c_x}$ is a ciphertext and v is a note in V.

1. if v is not a leaf, compute homomorphically and return

$$\begin{aligned} &\mathsf{Eval}_{pk}\big(\phi, \mathbf{c_x}[v.feature] - v.\theta\big) \cdot \mathsf{Enc\_Predict}(v.right, \mathbf{c_x}) \\ &+\mathsf{Eval}_{pk}\big(\phi, v.\theta - \mathbf{c_x}[v.feature]\big) \cdot \mathsf{Enc\_Predict}(v.left, \mathbf{c_x}) \end{aligned}$$

2. otherwise return $v.leaf\_value$.

Fig. 4. The subroutine $\mathsf{Enc\_Predict}(\cdot, \cdot)$ operating recursively on a node and ciphertext pair. The subroutine is an adjustment of Algorithm 1 to operate over encrypted data.

and any FHE scheme is circuit-private for (the corresponding, empty, family of circuits) $\tilde{G}$. Completeness of P follows from the correctness of $\mathcal{E}$ as $\langle \mathsf{Clnt_P}, \mathsf{Srv_P} \rangle$ computes exactly the same function as described in Algorithm 1.

To conclude the proof it remains to show that $\mathsf{Clnt_P}$, and $\mathsf{Srv_P}$ are PPT, which follows from the construction. More formally, $\mathsf{Srv_P}$'s complexity is proportional to the number of nodes in the tree, denoted by m. The number of basic operations (additions and multiplications) performed by $\mathsf{Srv_P}$ is constant for each internal node, and $O(L)$ for each leaf. Therefore, the total number of basic operation is $O(m \cdot L)$, and since each operation takes polynomial-time the $\mathsf{Srv_P}$ is PPT. The client $\mathsf{Clnt_P}$ only performs a single Enc and Dec operation, which takes polynomial-time in the input size.

Maintaining the tree private. For certain applications it is required that the tree, used for prediction, remains hidden from the server. The protocol in Figure 3 can be easily modified to keep the data instance as well as the tree private. Concretely, in a setting where privacy of the tree in needed, the tree will be transmitted encrypted to the server (not necessary by the client), and the same protocol as in Figure 3 will be executed with the following changes:

– the tree $\mathsf{T} = (V, E)$ is encrypted as follows: for each $v \in V$ the threshold $v.\theta$ is encrypted and $v.feature$ is first transformed into a 1-hot encoding vector of dimension k and then each entry in this vector is encrypted. We denote by $\tilde{v}.\theta$ the encrypted threshold associated with node v, and by $\tilde{\mathsf{v}}.\mathsf{feature}$ the encrypted 1-hot encoding associated with node v.
– step 1 in Figure 4: let $\mathbf{c_x}$ be an encryption of input data x as described in Figure 3. Compute:

$$\mathsf{Eval}_{pk}\big(\phi, \big(\sum_{i\in[k]} \mathbf{c_x}[i] \cdot \tilde{\mathsf{v}}.\mathsf{feature}[i]\big) - \tilde{v}.\theta\big)$$

We note that the proof of Theorem 3 can be applied to show that the modified protocol is privacy-preserving outsourcing protocol for *Tree Prediction Function* (Algorithm 1 in Section 3). Moreover, the modified protocol requires the server to compute only k additional multiplications and additions per node, and the overall multiplicative depth required for the tree grows only by 1.

5.2 Decision-Tree Training over Encrypted Data

In this section we present our privacy-preserving outsourcing protocol for training trees (and Random Forests). Our protocol is a careful adaptation of Algorithm 2 to the client-server setting in a way that the computational burden is almost fully on the server, with constant communication complexity and a number of rounds proportional to tree depth. Concretely, the client is responsible for computing, at each node, the final step of the impurity measure, which depends on the number of features, labels, and considered thresholds by the algorithm.

Although the protocol in Figure 5 is presented with respect to Gini impurity measure, it can be instantiated with other standard impurity measures, such as *entropy*.

Now we are ready to show that Protocol $\mathsf{T} = \langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ is a privacy preserving outsourcing protocol for *Tree Training Function* (Algorithm 2, Section 3). More formally, denoting by $\widetilde{\mathsf{Gini}}$ the class of polynomial-size circuits corresponding to the Gini function in in Figure 2,

Shared parameters: Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a FHE encryption scheme and let ϕ be the soft-step function from Equation 1 in Section 3. We denote by k and L the number of features and labels in the associated problem, respectively. We denote by S the set of considered thresholds. We use the notation and variable names from Algorithm 2, where $[\![\cdot]\!]$ denotes the encrypted version.
A security parameter λ and $\mathsf{maximal\_depth}$ bound.
Client's input: a set of n examples $\mathcal{X}$ and the corresponding labels $\mathcal{Y}$ where each example $x \in \mathcal{X}$ is in $[-1,1]^k$ and the corresponding $y_x \in \mathcal{Y}$ is a 1-hot encoding of the label.
Client's output: tree $\mathsf{T} = (V, E)$, where each node $v \in V$ is a data structure containing the following fields: $feature \in [k]$ and $\theta \in S$ denoting the feature and the threshold associated with the node, and $leaf\_value$ that is a L-dimension vector of values in $\mathbb{R}$ if v is a leaf. In addition, each note has $v.right$ and $v.left$ that denote the right and left children of v, respectively.
The protocol $\mathsf{T} = \langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ for training over trees proceeds as follows:

1. **Input outsourcing phase:**
   (a) $\mathsf{Clnt_T}$ runs $\mathsf{Gen}(1^\lambda)$ to obtain a secret key, public key pair (sk, pk).
   (b) $\mathsf{Clnt_T}$ encrypts each $x \in \mathcal{X}$ and the corresponding $y_x \in \mathcal{Y}$ (*i.e.*, each entry in x and y). $\mathsf{Clnt_T}$ obtains a set of ciphertexts $\mathsf{CTXT} = \{\mathbf{c_x}, \mathbf{c_{y_x}}\}_{x\in\mathcal{X}, y_x\in\mathcal{Y}}$ where each ciphertext $\mathbf{c_x}$ is an encryption of example x and $\mathbf{c_{y_x}}$ is an encryption of x's label y_x. Then CTXT is sent together with pk to $\mathsf{Srv_T}$.
2. **Computation phase:** We denote by W_v a set of weights $\{[\![w_x]\!]\}_{x\in\mathcal{X}}$ associated with each node v, where each $[\![w_x]\!] \in W_{root}$ is initialized to $\mathsf{Enc}_{pk}(1)$. In addition, we initialize a variable d to 0, this variable represents the currently constructed tree depth.
   For each depth $d \leq \mathsf{maximal\_depth}$ and for each node v to be constructed at depth d, the server $\mathsf{Srv_T}$ computes the sub-protocol $\mathsf{Enc\_Train}(\mathsf{CTXT}, W_v, d)$ in Figure 6. The set of weights W_v for node v is calculated at creation of v's parent node and contains $[\![w_x^{\mathsf{right}}]\!]$ if v is the right child and $[\![w_x^{\mathsf{left}}]\!]$ otherwise. Once v is created, v's parent node is updated to contain a pointer to v, in the appropriate right or left field.
3. **Output phase:** $\mathsf{Srv_T}$ sends to $\mathsf{Clnt_T}$ the trained tree $[\![\mathsf{T}]\!]$, where each node consists of $[\![i^*]\!]$ and $[\![\theta]\!]$ associated with the node. $\mathsf{Clnt_T}$ decrypts $[\![\mathsf{T}]\!]$ and outputs the cleartext tree.

Fig. 5. The training protocol $\mathsf{T} = \langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ for constructing a single tree. The protocol can be executed in parallel to train a Random Forest.

Theorem 4. *If $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is CPA-secure* FHE *encryption scheme that satisfies circuit-privacy for* $\widetilde{\mathsf{Gini}}$*, then the protocol* $\mathsf{T} = \langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ *(Figure 5) is a d-round privacy-preserving outsourcing protocol for* Tree Training Function *(Algorithm 2), where d is the trained tree depth.*

Proof. Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a CPA-secure FHE encryption scheme that satisfies circuit-privacy for $\widetilde{\mathsf{Gini}}$. The protocol $\mathsf{T} = \langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ is an $(\mathcal{E}, \widetilde{\mathsf{Gini}})$-aided outsourcing protocol for *Tree Training Function*, for $\widetilde{\mathsf{Gini}}$ defined in 2. We show that it satisfies the requirements of Theorem 1.

Sub-protocol $\mathsf{Enc\_Train}(\mathsf{CTXT}, W, d)$ where CTXT is a set of ciphertexts containing encrypted examples and labels, W is a set of encrypted weights $\{[\![w_x]\!]\}_{x\in\mathcal{X}}$ for the node to be constructed, and d is the depth of the constructed node. We use the notation and variable names from Algorithm 2, where $[\![\cdot]\!]$ denotes the encrypted version.
For the currently constructed node v at depth d, the sub-protocol proceeds as follows:

1. if d reached $\mathsf{maximal\_depth}$, $\mathsf{Srv_T}$ computes homomorphically $v.leaf\_value = \sum_{x\in X}[\![w_x]\!]\cdot \mathbf{c_{y_x}}$.
2. otherwise, $\mathsf{Srv_T}$ proceed as follows:
   (a) for each threshold $\theta \in S$ and each feature $i \in [k]$ homomorphically compute and send to $\mathsf{Clnt_T}$ the following L-dimension vectors of ciphertexts:
$$[\![\mathsf{right}[i,\theta]]\!] \leftarrow \sum_{x\in X}[\![w_x]\!]\cdot \mathsf{Eval}_{pk}\big(\phi, \mathbf{c_x}[i]-\theta\big)\cdot \mathbf{c_{y_x}}$$
$$[\![\mathsf{left}[i,\theta]]\!] \leftarrow \sum_{x\in X}[\![w_x]\!]\cdot \mathsf{Eval}_{pk}\big(\phi, \theta-\mathbf{c_x}[i]\big)\cdot \mathbf{c_{y_x}}$$
   (b) $\mathsf{Clnt_T}$ decrypts $[\![\mathsf{right}[i,\theta]]\!]$ and $[\![\mathsf{left}[i,\theta]]\!]$ for each threshold $\theta\in S$ and each feature $i \in [k]$ and computes the Gini function in Figure 2. Let i^* be the 1-hot encoding representation of the selected feature and let θ^* be the selected threshold.
   (c) $\mathsf{Clnt_T}$ encrypts i^* and θ^* and sends the ciphertexts to $\mathsf{Srv_T}$ (if any error occurs during decrypt or during calculation then it encrypts $\mathsf{Gini}(x)$ for an arbitrary x of the appropriate length). Let $[\![i^*]\!]$ be the encryption of i^*, *i.e.*, a k-dimensional vector of ciphertexts, and let $[\![\theta^*]\!]$ be the encryption of θ^*.
   (d) $\mathsf{Srv_T}$ sets $v.feature = [\![i^*]\!]$ and $v.\theta = [\![\theta^*]\!]$ and for each $x \in \mathcal{X}$ computes:
$$[\![w_x^{\mathsf{right}}]\!] \leftarrow [\![w_x]\!]\cdot \mathsf{Eval}_{pk}\big(\phi, \big(\sum_{j\in[k]}\mathbf{c_x}[j]\cdot[\![i^*]\!][j]\big)-[\![\theta^*]\!]\big)$$
$$[\![w_x^{\mathsf{left}}]\!] \leftarrow [\![w_x]\!]\cdot \mathsf{Eval}_{pk}\big(\phi, [\![\theta^*]\!]-\big(\sum_{j\in[k]}\mathbf{c_x}[j]\cdot[\![i^*]\!][j]\big)\big)$$

Fig. 6. The sub-protocol $\mathsf{Enc\_Train}(\cdot,\cdot,\cdot)$ operating on a set of ciphertexts of examples and corresponding labels, set of weights, and the current treated depth. The subroutine is an adaptation of Algorithm 2 to a protocol that operates over encrypted data.

– $\mathsf{Clnt_T}, \mathsf{Srv_T}$ are PPT:
  - $\mathsf{Srv_T}$'s complexity: In each node for each potential threshold (we have $|S|$ potential thresholds), and feature (k features), the $\mathsf{Srv_T}$ performs L homomorphic multiplication per example and n homomorphic additions, where n is the number of examples. Another k homomorphic multiplications and additions are performed after receiving a response from the $\mathsf{Clnt_T}$. Therefore $\mathsf{Srv_T}$ performs $O(|S| \cdot k \cdot L \cdot n)$ homomorphic operations per node, and has $O(m \cdot |S| \cdot k \cdot L \cdot n) \cdot poly(\lambda)$ total complexity (where $poly(\lambda)$ accounts for the complexity of each homomorphic operation).
  - $\mathsf{Clnt_T}$'s complexity: for each node, $\mathsf{Clnt_T}$ performs $k \cdot |S|$ operations of Dec and Enc and computes Gini on cleartext. The total complexity of the client is $O(m \cdot k \cdot |S|) \cdot poly(\lambda)$ (where $poly(\lambda)$ accounts for the complexity of each encryption and decryption operation). Note that the complexity is independent of the number of examples n in the input dataset.

– Completeness: follows from the correctness of $\mathcal{E}$ as $\langle \mathsf{Clnt_T}, \mathsf{Srv_T} \rangle$ computes exactly the same function as described in Algorithm 2.

– Gini is a family of admissible function: each G_n in Gini has the same output length for all inputs, *i.e.*, a k-dimensional vector i^* and a value $\theta^* \in S$. Moreover, the calculation in Gini consists only of (plaintext) addition, multiplication and division, which is polynomial-time computable.

6 Experimental Results

We empirically evaluated our decision trees algorithms and protocols for both accuracy and run-time performance. In Section 6.1 we report the accuracy of our decision tree algorithms that employ a soft-step function (presented in Section 3), in comparison to the accuracy of standard decision trees. In Section 6.2 we report the concrete run-time performance of our privacy-preserving training and prediction protocols (presented in Section 5).

Our accuracy and run-time performance evaluation is done with respect to a single decision tree, and can naturally be extended to multiple trees (as in random forests) where trees are trained/evaluated in parallel, each on a separate CPU core. Random forests are commonly used to achieve accuracy improvement.

Our decision trees algorithms use in each node a soft-step function (instead of the hard threshold). The soft-step function is realized by a degree 15 polynomial:

$$\begin{aligned}\phi(x) = &- 0.01404 \cdot x^{15} + 0.24219 \cdot x^{13} - 1.69826314 \cdot x^{11} + 6.21807861 \cdot x^9 \\ &- 12.6979 \cdot x^7 + 14.32256 \cdot x^5 - 8.35912664 \cdot x^3 + 2.49621375 \cdot x + 0.5\end{aligned}$$

To construct this polynomial, we first selected the polynomial that best approximates the step function as specified in Equation (2); we use in this equation a weighting function $w\colon [-2, 2] \to [0, 1]$ defined to be zero in the interval $[-0.2, 0.2]$ and a constant positive value elsewhere. Next, the resulting polynomial is squashed and moved so that its image is in the appropriate range: $\phi([-2, 2]) = [0, 1]$. For the training algorithm and the corresponding training protocol we use thresholds on a 0.05 grid in the $[-1, 1]$ interval.

The datasets on which we perform our empirical analysis, for both accuracy and run-time estimation, are taken from the standard UCI repository datasets [14]; See Table 2. These datasets range in size from very small (iris with 4 features and 150 examples) to the moderate size common in real-world applications (forest cover with 54 features and over half a million examples).

	data set name	# examples	# features	# labels
1	iris	150	4	3
2	wine	178	13	3
3	breast cancer	569	30	2
4	digits	1203	64	10
5	forest cover	581012	54	7

Table 2. UCI Datasets [14].

6.1 Evaluating Accuracy of our Decision-Tree Algorithms

We evaluate the accuracy of our decision trees algorithms that use the soft-step function (aka, Algorithms 1 and 2) by comparing their predictive power to standard trees, on the benchmark datasets (Table 2).

Accuracy was measured using a 3-fold cross-validation procedure. Each dataset was randomly partitioned into three equal-size parts. Each of the three parts serves as a test-set for a classifier trained on the remaining two. The *overall accuracy* is calculated as the percentage of correct classification on test examples (each example in the data is taken exactly once, so the accuracy reported is simply the percentage of examples that were correctly classified).

Our experiments consisted of executing training to produce both soft and standard trees, and executing prediction when using both soft and standard trees as the model. We compared all four possible combinations of the former:

- standard-standard: both training and prediction are done using a standard decision tree. (This is done using the python scikit-learn DecisionTreeClassifier object).
- standard-soft: training is conducted in the standard way, prediction uses soft-step function using Algorithm 1. This option simulates a scenario where a standard tree model is used on encrypted data without re-training.
- soft-standard: training using Algorithm 2, but once the tree structure is obtained, for prediction we first transform the soft tree to a standard tree (by replacing each soft-step function by its hard threshold counterpart), and then execute the prediction on the resulting standard tree. This simulates the situation where training must be done on encrypted data, but the resulting model is not kept secret, and henceforth may be used in the clear, and so with standard hard thresholds.
- soft-soft: both training and prediction are done with our soft tree method.

The results (Figure 7) show an overall comparable accuracy when comparing all four training-testing alternatives in all four datasets tested, and for trees of depth up to 5. Depth 0 trees are taken to be the majority-class baseline (*i.e.*, a prediction of the majority class regardless of the input data, which is the accuracy that is obtained without training a classifier, but with access to the proportion of examples having each label $\ell \in [L]$). Our results indicate that our soft tree algorithm (Algorithm 1) is a valid replacement for standard decision trees in terms of classification accuracy. This remains true with any combination of training and prediction with standard and soft tree variants.

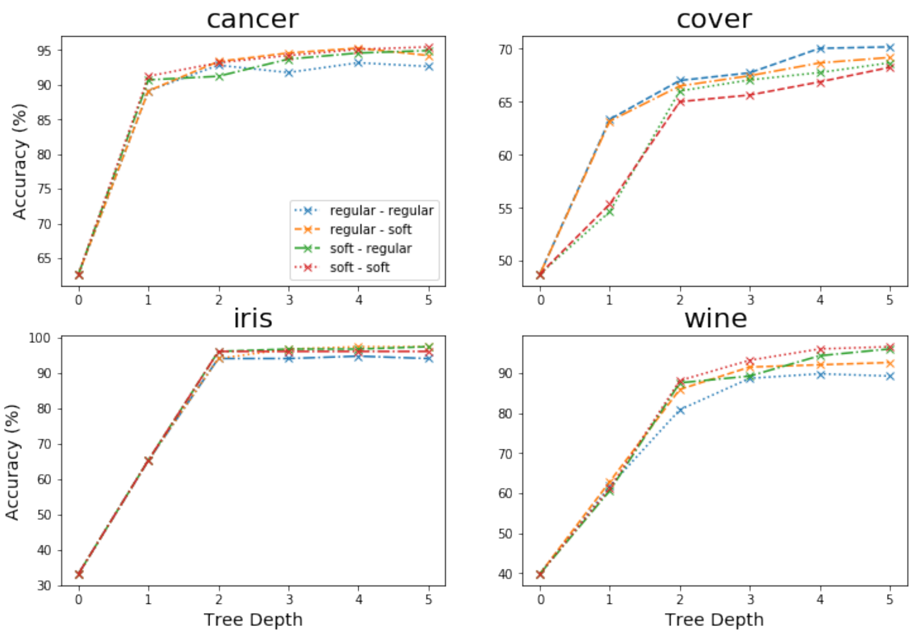

Fig. 7. Accuracy on UCI datasets of approximated vs. standard decision trees (regular in legend).

6.2 Evaluating Running-Time on Encrypted Data

Both our training and prediction protocols were implemented on top of Microsoft SEAL [33] using CKKS homomorphic encryption scheme [11] that supports arithmetic over encrypted floating-point real numbers. SEAL supports batching (also called, packing) a vector of messages into each ciphertexts with support for (homomorphic) coordinate-wise operations on the packed messages.

The number of messages that can be packed in each ciphertext is denoted by slots; the value of slots depends on the parameters used to initialize the scheme.

The chosen SEAL parameters for the experiments were the following: security level of 80 bits, irreducible polynomial degree poly_modulus_degree of 8192 (implies number of slots in a single ciphertext 4096), and total number of bits of primes is coeff_modulus = 340. The coeff_modulus were calculated using lwe-estimator [1,2,3]. The corresponding ciphertext size is 0.56MB. Our protocols were executed using soft-step function of degree 15 and tree depth 4.

Tree training experimental results. We executed the training protocol described in Figure 5 on an AWS x1.16xlarge as the server. Our implementation construct the tree in a BFS manner.

The input is encrypted using batching to pack many examples and their corresponding labels into a small number of ciphertext. Specifically, for an input of n labeled examples, each of dimension k (its number of features) and with L possible label values, we proceed as follows. First, for each feature, we batch the values associated with this feature in the input examples. Next, for each input example we represent the corresponding label as a 1-hot encoding. Then, for each index in L, we batch the values associated with this index in the labels encoding. This allows the total amount of ciphertexts outsourced to the server to be $\frac{n}{\mathsf{slots}} \times k$ for the input examples and $\frac{n}{\mathsf{slots}} \times L$ for the corresponding labels.

The implementation uses in memory storage to store the soft-step function results, for each feature, for each threshold on the grid, and for each label index. Our storage based implementation approach required the use of x1.16xlarge machine, that includes 976 GiB of RAM.

The input examples that were used for training were a random sub-sampling of the corresponding UCI Datasets that were also used to test accuracy (see Table 2). The timing results are given in Table 3.

dataset	training	#	#	Server time; training (minutes)		
name	examples	features	labels	storage preparation	tree construction	total time
iris	100	4	3	4	15	19
wine	119	13	3	13	46	59
cancer	381	30	2	27	80	107
digits	1,203	64	10	77	589	665
cover	10,000	54	7	163	1058	1220

Table 3. Training depth 4 trees with 15-degree polynomial approximation. The servers run-time results for each dataset are divided as following: (1) storage preparation: the time it took to compute on the encrypted input examples the soft-step function; (2) tree construction: the time it took the server to construct the decision tree; (3) total time: servers total training time.

Tree prediction experimental results. We executed the tree prediction protocol (Figure 3) on a single core of a PC with an Intel Core i7-4790 CPU and 16GB of memory. The prediction was on an encrypted unlabeled input example and cleartext decision tree. Since prediction consists of a single full-traversal of the tree, and each node considers a single entry in the input, the server's complexity is independent of the number of features in the input example. Hence, we used randomly generated examples and trees for the prediction timing experiment. The resulting prediction time over an encrypted sample was 2.3 seconds.

7 Conclusions

In this work, first we identify a wide family of outsourcing protocols for which we can guarantee privacy against malicious servers via a simple mechanism described in Theorem 1. Second, we design protocols for outsourcing decision tree based prediction and training, where the client's complexity is independent of the tree size and the training dataset size respectively, and where privacy holds against malicious servers. We implemented our protocols and performed extensive experiments on standard UCI datasets encrypted with fully homomorphic encryption, demonstrating high accuracy, fast prediction, and feasible training.

We leave as open problem the necessity of circuit-privacy for our mechanism. The circuit-privacy requirement seems to be an artifact of our choice to prove Theorem 1 using a generic reduction from Function-CPA to standard CPA security. An alternative approach would be to directly prove Function-CPA security for a concrete FHE scheme; this might eliminate the need for circuit-privacy.

8 Acknowledgements

We wish to thank Yaron Sheffer for helpful discussions.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard (November 2018)
2. Albrecht, M.R.: On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 103–129. Springer International Publishing, Cham (2017)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
4. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, New York, NY, USA, 1st edn. (2009)
5. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: European Symposium on Research in Computer Security. pp. 424–439. Springer (2009)

6. Benarroch, D., Brakerski, Z., Lepoint, T.: Fhe over the integers: decomposed and batched in the post-quantum regime. In: IACR International Workshop on Public Key Cryptography. pp. 271–301. Springer (2017)
7. Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., Vaikuntanathan, V.: Optimized homomorphic encryption solution for secure genome-wide association studies. Cryptology ePrint Archive, Report 2019/223 (2019), `https://eprint.iacr.org/2019/223`
8. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS. vol. 4324, p. 4325 (2015)
9. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 498–507. ACM (2007)
10. Chen, H., Gilad-Bachrach, R., Kyoohyung, H., Huang, Z., Jalali, A., Laine, K., Lauter, K.: Logistic regression over encrypted data from fully homomorphic encryption. BMC Medical Genomics **11** (10 2018). https://doi.org/10.1186/s12920-018-0397-z
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. pp. 409–437 (2017)
12. De Cock, M., Dowsley, R., Horst, C., Katti, R., Nascimento, A.C., Poon, W.S., Truex, S.: Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. IEEE Transactions on Dependable and Secure Computing **16**(2), 217–230 (2017)
13. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14. pp. 1–8. Australian Computer Society, Inc. (2002)
14. Dua, D., Graff, C.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
15. Emekci, F., Sahin, O.D., Agrawal, D., El Abbadi, A.: Privacy preserving decision tree learning over multiple parties. Data Knowl. Eng. **63**(2), 348–361 (Nov 2007). https://doi.org/10.1016/j.datak.2007.02.004, `http://dx.doi.org/10.1016/j.datak.2007.02.004`
16. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009), `crypto.stanford.edu/craig`
17. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. In: Stoc. vol. 9, pp. 169–178 (2009)
18. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols: Techniques and Constructions. Springer-Verlag, Berlin, Heidelberg, 1st edn. (2010)
19. Hesamifard, E., Takabi, H., Ghasemi, M., Wright, R.: Privacy-preserving machine learning as a service. Proceedings on Privacy Enhancing Technologies **2018**, 123–142 (06 2018). https://doi.org/10.1515/popets-2018-0024
20. de Hoogh, S., Schoenmakers, B., Chen, P., op den Akker, H.: Practical secure decision tree learning in a teletreatment application. In: International Conference on Financial Cryptography and Data Security. pp. 179–194. Springer (2014)
21. Joye, M., Salehi, F.: Private yet efficient decision tree evaluation. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 243–259. Springer (2018)
22. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)

23. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.: Logistic regression model training based on the approximate homomorphic encryption. BMC Medical Genomics **11** (10 2018). https://doi.org/10.1186/s12920-018-0401-7
24. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR Medical Informatics **6** (08 2017). https://doi.org/10.2196/medinform.8805
25. Kiss, Á., Naderpour, M., Liu, J., Asokan, N., Schneider, T.: Sok: Modular and efficient private decision tree evaluation. PoPETs **2019**(2), 187–208 (2019). https://doi.org/10.2478/popets-2019-0026, `https://doi.org/10.2478/popets-2019-0026`
26. Kyoohyung, H., Hong, S., Cheon, J., Park, D.: Logistic regression on homomorphic encrypted data at scale. Proceedings of the AAAI Conference on Artificial Intelligence **33**, 9466–9471 (07 2019). https://doi.org/10.1609/aaai.v33i01.33019466
27. Laurent, H., Rivest, R.L.: Constructing optimal binary decision trees is np-complete. Information processing letters **5**(1), 15–17 (1976)
28. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Annual International Cryptology Conference. pp. 36–54. Springer (2000)
29. Lory, P.: Enhancing the efficiency in privacy preserving learning of decision trees in partitioned databases. In: Domingo-Ferrer, J., Tinnirello, I. (eds.) Privacy in Statistical Databases. pp. 322–335. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
30. Remez, E.Y.: Sur la détermination des polynômes d'approximation de degré donnée. Comm. Soc. Math. Kharkov **10**(4163), 196 (1934)
31. Rivlin, T.J.: An introduction to the approximation of functions. Courier Corporation (2003)
32. Samet, S., Miri, A.: Privacy preserving id3 using gini index over horizontally partitioned data. In: Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications. pp. 645–651. AICCSA '08, IEEE Computer Society, Washington, DC, USA (2008). https://doi.org/10.1109/AICCSA.2008.4493598, `https://doi.org/10.1109/AICCSA.2008.4493598`
33. Microsoft SEAL (release 3.3). `https://github.com/Microsoft/SEAL` (2019), microsoft Research, Redmond, WA.
34. Tai, R.K., Ma, J.P., Zhao, Y., Chow, S.S.: Privacy-preserving decision trees evaluation via linear functions. In: European Symposium on Research in Computer Security. pp. 494–512. Springer (2017)
35. Tueno, A., Kerschbaum, F., Katzenbeisser, S.: Private evaluation of decision trees using sublinear cost. PoPETs **2019**(1), 266–286 (2019). https://doi.org/10.2478/popets-2019-0015, `https://doi.org/10.2478/popets-2019-0015`
36. Vaidya, J., Clifton, C., Kantarcioglu, M., Patterson, A.S.: Privacy-preserving decision trees over vertically partitioned data. ACM Transactions on Knowledge Discovery from Data (TKDD) **2**(3), 14 (2008)
37. Wang, K., Xu, Y., She, R., Yu, P.S.: Classification spanning private databases. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA. pp. 293–298. AAAI Press (2006), `http://www.aaai.org/Library/AAAI/2006/aaai06-047.php`
38. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. Proceedings on Privacy Enhancing Technologies **2016**(4), 335–355 (2016)

39. Xiao, M.J., Huang, L.S., Luo, Y.L., Shen, H.: Privacy preserving id3 algorithm over horizontally partitioned data. In: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies. pp. 239–243. PDCAT '05, IEEE Computer Society, Washington, DC, USA (2005). https://doi.org/10.1109/PDCAT.2005.191, http://dx.doi.org/10.1109/PDCAT.2005.191