Privacy-Preserving Decision Tree Classification Using VBB-Secure Cryptographic Obfuscation

Shalini Banerjee, Steven D. Galbraith, and Giovanni Russello

University of Auckland, New Zealand
{shalini.banerjee, s.galbraith, g.russello}@aucklanduni.ac.nz

Abstract. The use of data as a product and service has given momentum to the extensive uptake of complex machine learning algorithms that focus on performing prediction with popular tree-based methods such as decision trees classifiers. With increasing adoption over a wide array of sensitive applications, a significant need to protect the confidentiality of the classifier model and user data is identified. The existing literature safeguards them using *interactive* solutions based on expensive cryptographic approaches, where an encrypted classifier model interacts with the encrypted queries and forwards the encrypted classification to the user. Adding to that, the state-of-art protocols for protecting the privacy of the model do not contain *model-extraction attacks*.
We design an efficient *virtual black-box* obfuscator for binary decision trees and use the random oracle paradigm to analyze the security of our construction. To thwart model-extraction attacks, we restrict to evasive decision trees, as black-box access to the classifier does not allow a PPT adversary to extract the model. While doing so, we present an encoder for hiding parameters in an *interval-membership function*. Our exclusive goal behind designing the obfuscator is that, not only will the solution increase the class of functions that has cryptographically secure obfuscators, but also address the open problem of non-interactive prediction in privacy-preserving classification using computationally inexpensive cryptographic hash functions.

Keywords: program obfuscation · privacy-preserving classification · decision trees · hash functions.

1 Introduction

The growing sophistication in prediction and analysis by complex machine learning models has resulted in their adoption in critical and sensitive applications in the domains of healthcare, e-commerce, agriculture, cybersecurity, etc. [32] [10] [19] [31]. Among a wide range of classifier models, tree-based designs such as decision trees have attained utmost popularity [39] due to their less ambiguous nature, ease of deployment and robustness [33]. Intensively used in many disciplines, decision tree classifiers predict various sensitive results like whether a user should invest his money in stock market, or what disease is a patient diagnosed with. These models are trained from sensitive training data using customized learning algorithms and thus are mostly proprietary and confidential to the model-provider. Cloud-based services from Machine learning-as-a-service (MLaaS) platforms have added flexibility, scalability and ubiquity [18] to the existing design by allowing model-providers to outsource models to cloud servers, who in turn manage classification services for the users. The prediction services are usually provided at some cost and thus learning the model allows an adversary to bypass the query charges. Additionally, the adversary can be motivated to steal the training data, as the model itself might leak information on the training set points, as evident from model inversion attacks [2] [20]. Moreover, prediction queries by the user and final classification contains sensitive information about the user, and should not be disclosed to the model-provider/server. This calls for privacy-preserving solutions, where the model is hidden from anyone but the model-provider and prediction queries/classification are private to the user, such that no leakage of useful information happens during the classification phase [12].

1.1 Related work on privacy-preserving classification.

The existing solutions for safeguarding model and user queries/classification usually follow a paradigm, where the model is encrypted and outsourced to a server, where it processes the encrypted input and forwards encrypted classification to the user [42] [27].

Brickell *et al.* [13] suggest an interactive two-party protocol employing additive homomorphic encryption and ml oblivious transfers (where l is the bit-length of each input feature and m is the number of decision nodes), restricting the user from performing multiple queries on the encrypted tree. In their well-known work, Bost *et al.* [12] present a comparison protocol between model-provider and the user for each node in the decision tree using fully-homomorphic encryption (FHE) method. Tai *et al.* [34] make use of multiple communication rounds to transfer the path costs and encrypted labels to the client. Tueno *et al.* in [36] claim designing a non-interactive FHE/SHE based proposal, where entire evaluation takes place at the server. However, we believe that the protocol is interactive as the user needs to communicate with the server to submit encrypted query and receive encrypted classification. In order to reduce the high computation and communication costs introduced by the FHE techniques, the authors of [17] design a solution (SortingHat) that secures the prediction queries and classification results, but do not guarantee the privacy of the model. The propose a homomorphic comparison algorithm that takes encrypted prediction queries as input and compares them with the model in plaintext, claiming the practicability of their design in terms of the overheads incurred. However taking into account the potential attack vectors, we argue that deploying such a design is infeasible.

The approaches discussed above involve multiple rounds of communication [42] [27], and rely upon expensive cryptographic approaches (FHE [11] [12], garbled programs [5][6], etc.). Also, none of the approaches, to the best of our knowledge, provide a complete non-interactive solution towards privacy-preserving classification with decision trees [1], which is *the central idea of this paper*. Furthermore, we achieve this using cryptographic hash functions, which are inexpensive compared to prevalent FHE, oblivious transfer protocols, Paillier encryption schemes, garbled circuits, etc.

The other important concern is that, almost all decision trees are susceptible to generic *model-extraction attacks*, where an adversary on submitting a total of $m.\log_2(b/\epsilon)$ queries can learn the model from its black-box access, where m is the number of internal nodes and ϵ is the minimum width of an interval in a node [35]. The existing literature focuses on identifying these attacks by observing API calls and issuing warnings [24] [30] or adding perturbations [26] [41] . However, since there are no restrictions on the number of prediction queries made by a user [29], limiting them is not reasonable approach towards thwarting such attacks. We define *evasive* binary decision trees (see Definition 4.4) and claim that if a decision tree is not evasive, then maintaining the privacy of the model is impossible, and hence there is no choice but to restrict to evasive decision trees.

The goal of this paper is to design non-interactive solutions for privacy-preserving classification with evasive decision trees by employing *cryptographic obfuscation*, a brief study of which follows this section.

1.2 Obfuscation

Obfuscation is a powerful tool that guarantees security against reverse engineering of classified information in a program. The secrets, also called *assets*, are defined by the owner of the program and needs to be kept hidden from anyone who has access to the source code or binaries. An obfuscator takes a program P and outputs a semantically equivalent version $O(P)$, such that access to the obfuscated version gives no information on the assets that needs to be protected, yet facilitating to produce the same output as the original program.

The seminal work of Canetti *et al.* in [14] lays the groundwork of program obfuscation. The obfuscated program stores a hashed secret value and outputs 1, if user input x satisfies the condition $h(x) = c$. Barak *et al.* in [4] defines virtual black-box (VBB) notion, which states that an attacker with access to obfuscated code has no advantage in learning the asset(s) over a simulator having oracle access to a program. The authors also claim that achieving this property is too ambitious a goal for the generic class of programs as there exists a class of *unobfuscatable* programs that leak important asset that black-box access does not reveal. This result motivated constructing special-purpose obfuscators, emphasizing upon a special class of functions, called *evasive functions* (point-functions [15][37], pattern-matching with wildcards [8][9], compute-and-compare programs [23][38], fuzzy-matching for Hamming distance [21], hyperplane membership [16]) which gives a strong intuition that learning assets of a program by identifying the accepting inputs is computationally hard.

1.3 Our Contributions.

We make use of *cryptographic obfuscation* to address the following: (a) completely eliminate user interaction with the model-provider/server; (b) design solutions based on inexpensive cryptographic methods, as opposed to the prevalent FHE, oblivious transfer protocols, Paillier encryption schemes, garbled programs, etc. and (c) design solutions for a special class of decision trees, for which it is computationally hard to find an accepting input, such that a polynomial adversary cannot extract the model, except with negligible probability. On the whole, we contribute towards designing a VBB obfuscator for encoding *evasive decision trees* (see Definition 4.4). We focus on trees at constant depth, but the techniques will apply to more general classes of decision trees. Note that, we do not consider privacy-preserving methods to construct the model and how the model is obtained is out of the scope of this study. A technical briefing of our construction is as follows:

Technical Overview. Let n, ℓ, $d \in \mathbb{N}$. Consider a function $C : \mathbb{N}^n \rightarrow \{0,1\}$ that classifies input $(x_i)_{i\in[n]}$, $x_i \in \{0,1\}^\ell$ based on a full binary tree of depth d. At each internal node v_j, Boolean function $g_j : \{0,1\}^\ell \rightarrow \{0,1\}$ outputs 1 if and only if $x_i \leq t_j$, where t_j is an integer in $[0, 2^\ell)$. we aim to encode $[\![t_j, i]\!]$ at each v_j, along with the label of terminal nodes $(s_1, \ldots, s_{2^d+1})$, where $s_k \in \{0,1\}$. We assume that x_i is compared at most twice along a path (from root node at level 0 till it reaches terminal node at level $d-1$), i.e. $x_i \leq c_i + w_i$, $x_i > c_i$, where c_i, $c_i + w_i \in (t_1, \ldots, t_{2^d})$ and $w_i \leq \ell - \frac{\lambda}{n}$. This defines an integer interval $(c_i, c_i + w_i]$ where x_i is true, such that $C((x_i)_{i\in[n]}) = 1$, if $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$.

We explain the obfuscation as follows: Boolean functions $x_i \leq (c_i + w_i)$ and $x_i > c_i$ can be reduced to intervals $[0, c_i+w_i+1)$and $[c_i+1, 2^\ell)$ respectively where x_i is true. We divide the intervals into disjoint sub-intervals of the form $[a, a+2^p)$, where $p \in \{0, \ldots, \ell-1\}$. Note that the intersection of the two sub-intervals produces sub-intervals, the union of which is equivalent to the $(c_i, c_i+w_i]$, which we want to encode. Consider $f_i : \{0,1\}^\ell \rightarrow \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$ for $i \in \{0, 1, \ldots, \ell-1\}$. We generate the encodings $\mathcal{A}^i$ of the intersection of sub-intervals (of the form $[a, a+2^p)$) by calculating $H(f_p(a))$, where $H : \{0,1\}^* \rightarrow \{0,1\}^\omega$ is a hash function. Finally, for each encoding in $\mathcal{A}^i$, we concatenate n entries sorted in order of i and hash them using $H_c : \{0,1\}^* \rightarrow \{0,1\}^q$ and publish the set of hashes. Note that, reordering the nodes in order of i along each accepting path hides the structure, though the size of the obfuscated program may reveal the number of different accepting paths. To classify input $(x_i)_{i\in[n]}$, one computes the set of encodings for every $i \in [n]$, by calculating $H(f_p(x_i))$, where $p \in \{0, \ldots, \ell-1\}$ and for each such encoding, concatenates n entries sorted in order of i, and hashes them using H_c. For an accepting input, one of the generated hashes will be contained in the set hashes published by the obfuscator.

1.4 Organization.

This paper is organized into the following sections. Section 2 briefly describes the basic terminologies and notations that are used throughout the paper. Section 3 states the definitions of obfuscations and the security notions specific to our construction. Section 4 introduces the formal definitions of decision trees along with conditions for evasiveness. Section 5 gives a description of the proposed construction. Section 6 provides the proof for VBB security of our proposal. An overall conclusion is presented in Section 7.

2 Terminologies and Notations

Below are the standard notations and terminologies that will be used throughout the paper.

We denote the set of positive integers from 1 to $n \in \mathbb{N}$ as $S = \{x \in \mathbb{N} : 1 \leq x \leq n\}$. We denote by $|S|$, size of the set S. We use the standard notations to denote intervals as (a, b), $(a, b]$, $[a, b)$ and $[a, b]$, for a, $b \in \mathbb{N}$. We denote l-bit *binary encoding* of n as $r_{\ell-1}.2^{\ell-1} + \cdots + r_0.2^0$ for $n \in \mathbb{N}^+$, where $r_i \in \{0, 1\}$. We denote hamming weight of n as $wt(n) = \sum_{i=0}^{\ell-1} r_i$. For a program C, we denote its size by $|C|$. We rely upon the notion of *computational security* and follow the *asymptotic approach* throughout the paper. We provide the honest parties and the adversaries with a security parameter $\lambda \in \mathbb{N}$. We model the adversaries as a family of probabilistic polynomial time (PPT) programs, running in time $a.\lambda^c$, for some constants a, c. A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* in n, if is grows slower than n^{-c}, for every constant c. We measure negligibility with respect to the security parameter λ. We use $\|_{i=1}^{n} a_i$ to denote concatenation of a sequence of strings $(a_1, \ldots, a_n)$.

3 Obfuscation Definitions

In this section, we present the standard definition of obfuscation and discuss *virtual black box* (VBB) security that our obfuscator satisfies. We give formal definitions of *evasive functions*, as our obfuscator targets this specific class of functions.

Definition 3.1 (Obfuscation [4]). *Let $\mathcal{F} = \{\mathcal{F}\}_{\lambda \in \mathbb{N}}$ be a family of polynomial sized programs parameterized by inputs of length $n(\lambda)$, where λ is a security parameter. A probabilistic polynomial time (PPT) algorithm $\mathcal{O}$ is an obfuscator for $\mathcal{F}$, if is satisfies the following conditions:*

- *functionality preservation : For every $n \in \mathbb{N}$ and every $F \in \mathcal{F}_\lambda$, there exists a negligible function $\mu(\lambda)$, such that $\mathcal{O}(F)$ describes a program that computes the same function as F with a probability $1 - \mu(\lambda)$.*
- *polynomial slowdown : For every $n \in \mathbb{N}$ and every $F \in \mathcal{F}_\lambda$, there exists a polynomial p such that the running time of $\mathcal{O}(F)$ is bounded by $p(|F|)$, where $|F|$ denotes the size of the program.*

We next discuss the VBB property of an obfuscator, with the following intuition: if there exists a PPT algorithm that computes a predicate from an obfuscated function, then there exists another PPT algorithm that computes the same predicate from oracle access to the function with almost same probability [4].

Definition 3.2 (Virtual Black-Box Obfuscator [3] [4]). *Let $\mathcal{F} = \{\mathcal{F}\}_{\lambda \in \mathbb{N}}$ be a family of polynomial sized programs parameterized by inputs of length $n(\lambda)$, where λ is a security parameter. Let $\mathcal{O}$ be a PPT algorithm that takes as input $F \in \mathcal{F}$ and outputs a program $\mathcal{O}(F)$ (which not necessarily belongs to $\mathcal{F}$). The algorithm $\mathcal{O}$ is a VBB obfuscator for the family of programs $\mathcal{F}$ if it satisfies the syntactic properties (functionality preservation and polynomial slowdown properties stated in Definition 3.1) and an additional property as follows: For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to F, such that for every (non-uniform) polynomial size predicate $\varphi : F \to \{0,1\}$, there exists a negligible function $\mu(\lambda)$ such that:*

$$\Big| Pr[F(\mathcal{A}(\mathcal{O}(F)) = \varphi(F)] - Pr[\mathcal{S}^F(1^\lambda) = \varphi(F)] \Big| \leq \mu(\lambda) \tag{1}$$

where the first probability are taken over the coin tosses of $\mathcal{A}$ and $\mathcal{O}$ and he second probability is taken over the coin tosses of $\mathcal{S}$.

Evasive Functions. These are a special class of Boolean functions with the condition that for a random program from the collection, a PPT algorithm finds it hard to map an accepting input.

Definition 3.3 (Evasive Program Collection [3]). *A collection of programs $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ parameterized by inputs of length $n(\lambda)$ is called evasive, if there exists a negligible function $\mu(\lambda)$, such that for every $\lambda \in \mathbb{N}$ and every input $x \in \{0,1\}^{n(\lambda)}$:*

$$\Pr_{F \leftarrow \mathcal{F}_\lambda}[F(x) = 1] \leq \mu(\lambda)$$

where the oracle access to the program allows at most $p(n)$ queries.

4 Decision Trees

In this section we briefly introduce binary decision trees and present some related formal definitions that will be used throughout the paper. Without loss of generality, we define decision tree as a full binary tree and restrict classification labels to be in $\{0,1\}$, i.e. the tree provides binary classification. In scenarios, where the binary tree is not full, dummy nodes should be added [40].

Definition 4.1 (Decision Trees). *Let $n, d, \ell \in \mathbb{N}$ and $(x_i)_{i=1}^n = (x_1, \dots x_n) \in \mathbb{N}^n$ be a finite sequence of input elements, where x_i is an integer between 0 and $2^\ell - 1$ and represents the value of some attribute for all $i \in [n]$. Classification of an input $(x_1, \dots, x_n)$ is defined as evaluation of the classification function*

$C : \mathbb{N}^n \to \{0,1\}$, $(x_1, \ldots, x_n) \mapsto C(x_1, \ldots, x_n)$ *based on a model decision tree defined as follows:*

A full binary tree where $\mathcal{D} = (v_1, \ldots, v_{2^d})$ *denotes decision nodes in level-order sequence, and* d *is the depth of the tree. Let* $\mathcal{S} = (s_1, \ldots, s_{2^d+1})$ *be the sequence of labels of terminal nodes where* $s_k \in \{0,1\}$*. Each node* $v_j \in D$ *associates a Boolean function* $g_j : \{0,1\}^\ell \to \{0,1\}, (x_i) \mapsto g_j(x_i)$ *such that* $g_j(x_i) = 1$*, if* $x_i \le t_j$*, which specifies the branch to walk, where* $t_j \in [0, 2^\ell)$ *denotes the threshold value at* v_j*. At input* (x_i)*, the function* g_j *iterates* $(d-1)$ *times starting from root node at level* 0 *such that function* g_j *at level* $\lfloor \log_2 j \rfloor$ *determines which function* g_{j+1} *at the next level* $\lfloor \log_2 (j+1) \rfloor$ *to be used, till it reaches the terminal nodes at level* $(d-1)$ *which are labeled by one of the classes in* $\{0,1\}$*.*

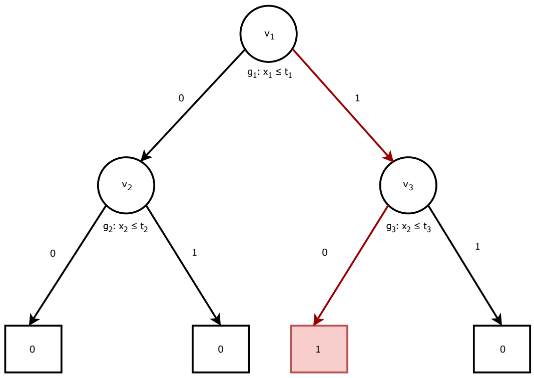

Fig. 1: Binary classification with a decision tree: the circular nodes represent decision nodes, and the square nodes represent terminal nodes. Decision nodes are numbered in level-order sequence. The path in red represents the accepting path with terminal node labeled 1.

Guided by our objective to prevent an adversary from extracting the model, we make an effort to define *assets* in a classification program. Each decision node v_j associates Boolean function g_j that checks whether the inequality $x_i \le t_j$ holds or not. We identify t_j and index i in $(x_i)_{i \in [n]}$, along with the labels of terminal nodes $\mathcal{S}$, as critical and central to the predictive behavior of the model. In what follows, we define Structure of Decision Trees as the assets that we want to protect in a decision tree classification program.

Definition 4.2 (Structure of Decision Trees). *For each decision node* $v_j \in D$*, there exists a tuple* $[\![t_j, i]\!]$*, where* t_j *denotes the corresponding threshold value and* i *denotes the index in the sequence* $(x_1, \ldots, x_n)$*. Structure of a decision tree, denoted by* $str(C)$ *defines the sequence of* 2^d *tuples and the sequence of labels of the terminal nodes* $\mathcal{S}$*.*

We assume x_i to be compared at most twice along an accepting path. Let $w_{max} \in \mathbb{N}$ and c_i, $c_i + w_i$ be integers between 0 and $2^\ell - 1$ and $w_i \in [0, w_{max}]$.

Consider $x_i \leq c_i + w_i$ and $x_i > c_i$ along a path from root at level 0 till terminal nodes at level $d-1$. The collection of inequalities define an interval $(c_i, c_i + w_i]$, where x_i is true. Finally, $C((x_i)_{i\in[n]}) = 1$ if $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$.

Definition 4.3 (Decision Region). *Let n, ℓ, $w_{max} \in \mathbb{N}$. Let c_i, $c_i + w_i$ be integers between 0 and $2^\ell - 1$ and $w_i \in [0, w_{max})$. Define decision region to be the hyper-rectangular region formed by n overlapping intervals $(c_i, c_i + w_i]$ determined by the classification function C, such that $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$.*

Evasive Function Family. As stated in Definition 4.1, a binary classification program maps input $(x_i)_{i\in[n]}$ to one of the classes $\{0, 1\}$. From [35], it is evident that an adversary, by identifying the accepting/rejecting inputs, can extract the model (learn the assets in the classification program using binary search) within polynomial attempts. From our discussion in Section 1, we can conjecture that it is impossible to protect the privacy of the model from generic model-extraction attacks, and this is our intuition behind restricting to a special class of classification programs, for which it is computationally hard to find an accepting input. We call this *evasive* collection which states that for every input, a random program selected from this collection evaluates to 0 with overwhelming probability. In what follows, we define evasive decision tree collection, with the exclusive goal of obfuscating this class of programs, such that adversary cannot learn the assets from the input/output behavior of the programs. Throughout the paper, we assume that an adversary knows the domain of inputs, but not the accepting inputs.

Definition 4.4 (Evasive Decision Tree Collection). *Let $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of polynomial-size classification functions where C_λ defines a set of decision tree classifiers with a security parameter λ. Every $C \in C_\lambda$ maps an input sequence $\{x_i\}_{i\in[n]}$ to a single output bit, where $n = n(\lambda)$ and $d = d(\lambda)$. We say, C is evasive, if there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, for every input $(x_i)_{i\in[n]}$:*

$$\Pr_{C \leftarrow C_\lambda} [\, C\,(\,\{x_i\}_{i\in[n]}) =\, 1] \leq\, \mu(\lambda)$$

In short, Definition 4.4 points out that for every $\{x_i\}_{i\in[n]}$, a program C chosen randomly from the collection C_λ evaluates to 1 with negligible probability.

Any distribution $X_n \in [0, 2^\ell)^n$ defines a distribution C_λ, such that $C \leftarrow C_\lambda$ computes whether an input $(x_i)_{i\in[n]}$ is accepted or not. This is equivalent to choosing $(c_1, \ldots, c_n) \leftarrow X_n$ and $(w_1, \ldots, w_n) \in [0, w_{max})^n$, such that $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. For the program collection to be evasive, it is necessary that this probability is negligible. Thus we require X_n to have large entropy. As uniform distributions provide the highest entropy, we have the best possible conditions guaranteed under these distributions. However, while dealing with real-world applications, we might come across scenarios where the sampling is done from non-uniform distributions. Intuitively, the scenarios that lead to non-evasiveness are: (1) the decision regions are too big; (2) the number of points in the space $[0, 2^l)^n$ representing $(c_1, \ldots, c_n)$ are too few; (3) the decision regions overlap with each other. Figure 2 shows two example distributions that lead towards non-evasiveness in decision trees.

Hence for an evasive collection, the distribution X_n needs to have a large number of points representing $(c_1, \ldots, c_n)$. Adding to that, the distribution needs to be 'well-spread', such that the overlapping between points are relatively small. We further this discussion with the calculation of the required parameters for

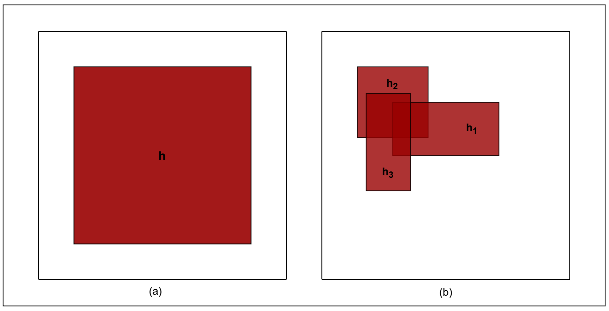

Fig. 2: Two example cases of distributions which lead towards non-evasiveness: (a) Decision region h_1 is very big. (2) Overlapping decision regions h_1, h_2 and h_3.

identifying an evasive program collection. We start with uniform distribution on $[0, 2^l)^n$, where n, ℓ are polynomials in λ.

Lemma 4.1. *Let $n, \ell \in \mathbb{N}$. Let c_i is integer between 0 and $2^\ell - 1$ and w_i is an integer between 0 and $w_{max} - 1$. The maximum number of elements in the decision region defined by $(c_i, c_i + w_i]$, for $i \in [n]$, is at most $(w_{max})^n$.*

Proof. For an c_i selected uniformly in $[0, 2^\ell)$, w_i has to be selected such that $w_i < w_{max}$, for some $w_{max} \in \mathbb{N}$. It can be readily seen that the number of ways of selecting the elements along an interval is w_{max}. For all the n intervals chosen uniformly and independently of each other, the number of possible ways is at most $(w_{max})^n$.

Lemma 4.2. *Let $\lambda \in \mathbb{N}$ be the security parameter and $n, \ell, w_{max} \in \mathbb{N}$, where $w_{max} \leq 2^{(\ell - \frac{\lambda}{n})}$. Fix an input $(x_i)_{i \in [n]}$. Then the probability that $(x_i)_{i \in [n]}$ belongs to the decision region defined by $(c_i, c_i + w_i]$, for $i \in [n]$, where c_i is chosen uniformly from $[0, 2^\ell)$, w_i is chosen uniformly from $[0, w_{max})$, is not more than $2^{-\lambda}$.*

Proof. The total number of points in the space $[0, 2^\ell)^n$ is given by $2^{\ell n}$. The input $(x_i)_{i \in [n]}$ is contained in the decision region defined by $(c_i, c_i + w_i]$, $i \in [n]$, when $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$, where c_i and w_i are chosen uniformly from $[0, 2^\ell)$ and $[0, w_{max})$ respectively.

Now, for a fixed input $(x_1, \ldots, x_n)$ to be contained in the decision region, the c_i's need to be selected such that $c_i \in (x_i - w_i, x_i]$, for every $i \in [n]$. It can be readily seen from Lemma 4.1 that the number of ways of selecting $(c_1, \ldots, c_n)$ such that the w_i's are less than w_{max} are $(w_{max})^n$ and thus the probability that $(x_i)_{i\in[n]}$ belongs to the decision region defined by $(c_i, c_i + w_i)$ is given by $\frac{(w_{max})^n}{2^{\ell n}}$. For $w_{max} \leq 2^{(\ell - \frac{\lambda}{n})}$, the above probability is at most $2^{-\lambda}$.

Lemma 4.3. *Let λ be the security parameter and ℓ, n are polynomials in λ. Let $w_{max} = w_{max}(\lambda)$ be a polynomial such that $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$. Let $2^{-\lambda}$ be a negligible function. Let X_n be an uniform distribution on $[0, 2^\ell)^n$ and C_λ be the corresponding distribution on decision trees that checks if $(x_i)_{i\in[n]}$ belongs to the decision region defined by the c_i's and w_i's in the distribution. Then C_λ is an evasive program collection.*

Proof. A uniform distribution on $[0, 2^\ell)^n$ defines a C_λ, and we need to show that for every $\lambda \in \mathbb{N}$ and every $(x_i)_{i\in[n]}$, $\underset{C \leftarrow C_\lambda}{Pr}[C((x_i)_{i\in[n]}) = 1] \leq \mu(\lambda)$. For a $C \leftarrow C_\lambda$, probability that $(x_i)_{i\in[n]}$ gets accepted is equivalent to the probability of choosing intervals $(c_i, c_i + w_i]$, where $c_i \leftarrow [0, 2^\ell)$, $w_i \leftarrow [0, w_{max}]$, such that $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. It is evident from Lemma 4.2 that the probability is at most $2^{-\lambda}$, which is a negligible function in λ.

We next discuss what it means to *learn* a decision tree classification program. An attacker aims to reverse-engineer the program to identify $str(C)$ and thus unlearnability means that an attacker, with oracle access to the C, cannot determine $str(C)$ with overwhelming probability.

Definition 4.5 (Unlearnable Decision Trees). *A collection of classification functions $\mathcal{C}$ is unlearnable, if for every polynomial time adversary $\mathcal{A}$ with oracle access to C, there exists a negligible function μ, such that for every $\lambda \in \mathbb{N}$:*

$$\underset{C \leftarrow C_\lambda}{Pr}[(\mathcal{A})^{C(1^\lambda)} = str(C)] \leq \mu(\lambda)$$

5 Constructing the Obfuscator for Evasive Decision Trees

This section presents our construction for obfuscating evasive decision trees along with an introduction to some basic assumptions and an insight to the building blocks availed for designing the proposal.

5.1 Setup.

Without loss of generality, we assume decision trees to be full binary trees that perform binary classification on an input $(x_i)_{i\in[n]}$, where $x_i \in \{0,1\}^\ell$. We consider a decision tree classification program $C \leftarrow C_\lambda$ with depth d. We denote internal nodes by $(v_1, \ldots, v_{2^d})$ and terminal nodes by $\mathcal{S} = (s_1, \ldots, s_{2^d+1})$. An *accepting path* P_{s_τ} is defined to be the sequence of tuples $[\![t_j, i, b]\!]$, such that v_j

is an ancestor node of $s_\tau \in \mathcal{S}$, where $s_\tau = 1$ and $b \in \{0,1\}$ denotes the output of Boolean function $g_j(x_i)$. We assume each element in the input sequence to be used at most twice along an accepting path. This assumption in reasonable, since any collection of inequalities in the form $x_i \leq t_j$ or $x_i > t_j$ defines an interval and so is defined by a pair of comparisons. We assume that Evaluation procedure (Algorithm 5.6) is oblivious to the tuples in P_{s_τ} and is only allowed to know d.

5.2 Building Blocks.

We want the obfuscated tree to be *unlearnable* i.e. we aim to hide $str(C)$ from a PPT adversary. To achieve the same, we develop a library of building blocks that enables encoding arbitrary integer intervals, which we leverage to build our obfuscator.

Reducing Inequality into Intervals. A decision node associates function $g : \{0,1\}^\ell \to \{0,1\}$, such that $g(x) = 1$, if $x \leq t$ and 0 otherwise, where t is any integer between 0 and $2^\ell - 1$. Thus, the Boolean function splits the integer interval $[0, 2^\ell)$ at each node into two distinct partitions, call it $\mathcal{X} = [0, t+1)$ and $\mathcal{X}' = [t+1, 2^\ell)$, where each interval contains ℓ-bit binary encoding of the integers. We further divide intervals $\mathcal{X}$ and $\mathcal{X}'$ into a sequence of disjoint sub-intervals of the form $[a, a+2^p)$, which is the primary building block of our construction. We present the formal description in Algorithm 5.1 and Algorithm 5.2.

Algorithm 5.1 $\mathsf{GenInt}_{\mathcal{X}}(t)$

Input: $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
Output: $\{[a_j, a_j + 2^{p_j})\}_{j \in [k]}$

1: Compute $\mathcal{X} = [0, t+1)$, where $\mathcal{X}$ contains ℓ-bit binary encoding of integers.
2: Compute $k = wt(|\mathcal{X}|)$, where $wt(n)$ calculates hamming weight of n.
3: Partition $[0, t+1)$ into k disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, such that $a_j = a_{j-1} + 2^{p_{j-1}}$ and $a_1 = 0$, where $p_1, \ldots, p_k$ ($p_1 > p_2 > \cdots > p_k$) denote the bit positions of the 1's in the binary encoding of $|\mathcal{X}|$.

Algorithm 5.2 $\mathsf{GenInt}_{\mathcal{X}'}(t)$

Input: $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
Output: $\{[a_j, a_j + 2^{p_j})\}_{j \in [k]}$

1: Compute $\mathcal{X}' = [t+1, 2^\ell)$, where $\mathcal{X}'$ contains ℓ-bit binary encoding of integers.
2: Compute $k' = wt(|\mathcal{X}'|)$.
3: Partition $[t+1, 2^\ell)$ into k' disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, such that $a_j = a_{j-1} + 2^{p_{j-1}}$ and $a_1 = t+1$, where $p_1, \ldots, p_{k'}$ ($p_1 < p_2 < \cdots < p_{k'}$) denote the bit positions of the 1's in the binary encoding of $|\mathcal{X}'|$.

Intersection of Intervals. Let $\mathcal{I}_{\mathrm{X}}$ be a set of sub-intervals, all of the form $[a, a+2^j)$ for some a and j. Let $\mathcal{I}_{\mathrm{X'}}$ be a set of sub-intervals, all of the form $[b, b+2^r)$ for some b and r. Define intersection of $\mathcal{I}_{\mathrm{X}}$ and $\mathcal{I}_{\mathrm{X'}}$ as $\mathcal{I} = \{I \cap J : I \in \mathcal{I}_{\mathrm{X}}, J \in \mathcal{I}_{\mathrm{X'}}\} \setminus \emptyset$.

Lemma 5.1. *Let $\ell \in \mathbb{N}$. Consider algorithms $\mathsf{GenInt}_{\mathcal{X}}$ (Algorithm 5.1) and $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 5.2). Let c, $c+w \in \mathbb{Z}$ be such that $0 \le c < c+w < 2^\ell$. Let $\mathcal{I}_X \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c+w)$, $\mathcal{I}_{X'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(c)$. Let $\mathcal{I} = \{I \cap J : I \in \mathcal{I}_X, J \in \mathcal{I}_{X'}\} \setminus \emptyset$. Then, for every integer c, $c+w$, $I \subseteq J$ or $J \subseteq I$ and $|\mathcal{I}| \le 2\ell - 2$.*

Proof. $\mathsf{GenInt}_{\mathcal{X}}(c+w)$ divides $[0, c+w+1)$ into k disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, where k is the hamming weight of ℓ-bit binary encoding of $c+w+1$. Since $p_j > p_{j-1}$, we can conclude that $2^{p_1} \le c+w+1 < 2^{p_1+1}$ and $\mathcal{I}_{\mathrm{X}} = \{[0, 2^{p_1}), \ldots, [2^{p_1} + \cdots + 2^{p_{k-1}}, 2^{p_1} + \cdots + 2^{p_k})\}$, where $\ell > p_1$. Since $c \in [0, c+w+1)$, we consider the following:

- If $c+1 = 2^q$, where $q \le p_1$, then $\mathcal{I}_{\mathrm{X'}} = \{[2^i, 2^{i+1})\}_{i \in \{q, q+1, \ldots, \ell-1\}}$, and it can be clearly seen that for every non-empty intersection $I \cap J$, either $I \subseteq J$ or $J \subseteq I$.

- If $2^{q-1} < c+1 < 2^q$, where $q \le p_1$. Let k' be the hamming weight of ℓ-bit binary encoding of $2^q - (c+1)$. Then, for $J = \{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{k'-1}}, c+1+\cdots+2^{p'_{k'}})\}$, where $2^q = c+1+\cdots+2^{p'_{k'}}$, $J \subseteq I$, where $I = [0, 2^{p_1})$. Also note, for $J = [2^{p_1}, 2^{p_1+1})$ and $I = \mathcal{I}_{\mathrm{X}} \setminus \{[0, 2^{p_1})\}$, $I \subseteq J$.

- If $2^{p_1} + \cdots + 2^{p_{m-1}} \le c+1 < 2^{p_1} + \cdots + 2^{p_m}$, where $m \le k$. Since $\mathsf{GenInt}_{\mathcal{X}'}(c)$ divides $[c+1, 2^\ell)$ into k' sub-intervals $\{[b_r, b_r + 2^{p'_r})\}_{r \in [k']}$, then let $p'_1 < \cdots < p'_s \le p_m < p'_{s+1} < \cdots < p'_{k'}$ for some $s < k'$. Let $I = [2^{p_1} + \cdots + 2^{p_{m-1}}, 2^{p_1} + \ldots, +2^{p_m})$. Then, for a non-empty intersection, $J = \{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{s-1}}, c+1+\cdots+2^{p'_s})\}$, $2^{p_1}+\cdots+2^{p_m} = c+1+\cdots+2^{p'_s}$ as 0 to $m-1$ bits of $2^\ell - (c+1)$ and $2^{p_1} + \cdots + 2^{p_m} - (c+1)$ are equal, and thus $J \subseteq I$. For all other non-empty intersections, $I \subseteq J$.

It is clear from algorithms 5.1 and 5.2 that $|\mathcal{I}_{\mathrm{X}}|$, $|\mathcal{I}_{\mathrm{X'}}| \le \ell$ (when the hamming weight of $|\mathcal{X}|$ and $|\mathcal{X}'|$ are ℓ). Let $|\mathcal{I}_{\mathrm{X}}| = |\mathcal{I}_{\mathrm{X'}}| = \ell$. Since $0 \notin (c, c+w]$, there exists an $[a_j, a_j + 2^{p_j}) \in \mathcal{I}_{\mathrm{X}}$, such that $[a_j, a_j + 2^{p_j}) \notin \mathcal{I}$. Also, since $2^\ell - 1 \notin (c, c+w]$ as $wt(c+w+1) = \ell$, there exists a $[b_r, b_r + 2^{p'_r}) \in \mathcal{I}_{\mathrm{X'}}$, such that $[b_r, b_r + 2^{p'_r}) \notin \mathcal{I}$ and thus $|\mathcal{I}| \le 2\ell - 2$.

Encodings of Interval. Let $\mathcal{I}$ be a set of sub-intervals, all of the form $[a, a+2^j)$. The encoder (Algorithm 5.3) receives $\mathcal{I}$ as input and outputs the set of encodings $\{h_1, \ldots, h_{|\mathcal{I}|}\}$. Define a family of functions $\mathcal{F}$ as follows: $\mathcal{F} = \{f_0, \ldots, f_{\ell-1}\}$ where $f_i(y) : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be a hash function with $\ell < \omega$ such that H is injective on the set of all strings of length less than or equal to ℓ.

Algorithm 5.3 IntEnc

Input: $\mathcal{I}$
Output: $\{h_1, \ldots, h_{|\mathcal{I}|}\}$

for $j = 1$ to $|\mathcal{I}|$ **do**
  Compute $\mu_j = f_{p_j}(a_j)$
  Compute $h_j = H(\mu_j)$.
end for

Decoding. The decoding algorithm 5.4 receives as input $\{h_1, \ldots, h_{k_{|\mathcal{I}|}}\} \leftarrow$ IntEnc $(\mathcal{I})$ and $x \in \{0,1\}^\ell$ and outputs 1 if x belongs to any of the sub-intervals in $\mathcal{I}$.

Algorithm 5.4 Dec (with embedded data $\{h_1, \ldots, h_{|\mathcal{I}|}\}$)

Input: $\ell \in \mathbb{N}$, $x \in \{0,1\}^\ell$
Output: 0 or 1.

for $i = 0$ to $\ell - 1$ **do**
  Compute $H(f_i(x))$
  if $H(f_i(x)) \in \{h_1, \ldots, h_{|\mathcal{I}|}\}$ **then**
    return 1
  end if
end for
return 0

Lemma 5.2 (Correctness). *Consider Algorithms* $\mathsf{GenInt}_{\mathcal{X}}$ *(Algorithm 5.1),* $\mathsf{GenInt}_{\mathcal{X}'}$ *(Algorithm 5.2),* IntEnc *(Algorithm 5.3),* Dec *(Algorithm 5.4) and input* $x \in \{0,1\}^\ell$*. Let* $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c+w)$ *and* $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c)$ *and* $\mathcal{I} \leftarrow \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'}$*. Let* $H : \{0,1\}^* \rightarrow \{0,1\}^\omega$ *be injective on the set of all strings of length less than or equal to* ℓ*, where* $\ell < \omega$*. For every integer* $c, c+w \in [0, 2^\ell)$*, for every* $\{h_1, \ldots, h_{|\mathcal{I}|}\} \leftarrow$ IntEnc $(\mathcal{I})$ *and for every* $x \in (c, c+w]$*,* Dec *outputs* 1*.*

Proof. Let $\mathcal{I} = \{[a_j, a_j + 2^{p_j})\}_{j \in [k]}$, then from Lemma 5.1, we can say that $\bigcup_{j=1}^{k} [a_j, a_j + 2^{p_j})$ contains all sub-intervals in $(c, c+w]$. Let x be an integer in $(c, c+w]$, then it must belong to at least one of the sub-intervals in $\mathcal{I}$. Algorithm 5.3 computes $f_{p_j}(y) = \mu_j$, for every $[a_j, a_j + 2^{p_j}) \in \mathcal{I}$. If $x \in [a_j, a_j + 2^{p_j})$, then there exists an $i \in \{0, \ldots, \ell - 1\}$ such that $f_i(x) = f_{p_j}(a_j) = \mu_j$. Hence $H(f_i(x)) \in \{h_1, \ldots, h_{|\mathcal{I}|}\} \leftarrow \mathsf{IntEnc}(\mathcal{I})$ and Dec outputs 1.

If $x \notin [a_j, a_j + 2^{p_j})$, $\nexists i = \{1, \ldots, \ell - 1\}$, such that $f_i(x) \in (\mu_j)_{j=1}^{|\mathcal{I}|}$ and therefore, $(h_1, \ldots, h_{|\mathcal{I}|})$ will not contain $H(f_i(x))$. Finally, Dec will correctly reject the input.

5.3 Obfuscator $\mathcal{O}_{\mathcal{D}}$

We now put forward the design of proposed decision tree obfuscator $\mathcal{O}_{\mathcal{D}}$ which takes $C \in C_\lambda$ as input and produces $C' \in \mathcal{C}'$, where $\mathcal{C}'$ denotes a separate family of polynomial-size programs. We first present a high-level survey, followed by a formal description of our construction.

Each $x_i \in (x_i)_{i\in[n]}$ is present at most twice along an accepting path, and as such the collection of inequalities $x_i \leq (c_i + w_i)$ and $x_i > c_i$ define $x_i \in (c_i, c_i + w_i]$, where c_i, $c_i + w_i \in (t_1, \ldots, t_{2^d})$. Ultimately, $C((x_i)_{i\in[n]})) = 1$, if $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. Our aim is to encode $(c_i, c_i + w_i]$ for every $i \in [n]$ along an accepting path. To do so, we calculate $\mathcal{I}^i_{\mathrm{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c_i + w_i)$ and $\mathcal{I}^i_{\mathrm{X'}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c_i)$ which gives the set of sub-intervals in $[0, c_i + w_i + 1)$ and $[c_i + 1, 2^\ell)$ respectively.

Next, we determine $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathrm{X}} \cap \mathcal{I}^i_{\mathrm{X'}}$ to generate sub-intervals, the union of which is equivalent to the interval $(c_i, c_i + w_i]$. To encode the interval, we calculate encodings $\mathcal{A}^i \leftarrow \mathsf{IntEnc}\,(\mathcal{I}^i)$. Let $H_c : \{0,1\}^* \rightarrow \{0,1\}^q$ be a hash function which maps arbitrary length bit strings to q-bit strings. Finally, for every element in $\mathcal{A}^i$, we concatenate n entries sorted in ascending order of i and hash them using H_c and publish the hashes. Formally, the obfuscated program is given in Algorithm 5.5.

Algorithm 5.5 Obfuscator $\mathcal{O}_{\mathcal{D}}$

Input: d, n, $\ell \in \mathbb{N}$, $str(C)$.
Output: α hash values, where $\alpha \leq |\mathcal{S}|\ell^{d-1}$.

Compute $P_{s_\tau} = (\llbracket t_j, i, b \rrbracket : v_j$ is an ancestor of $s_\tau \in \mathcal{S}$, $s_\tau = 1$ and $b = g_j(t_j))$.
for $i = 1$ to n **do**
  if $\llbracket t_{j_1}, i, 1 \rrbracket \in P_{s_\tau}$ **then**
    $\mathcal{I}^i_{\mathrm{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(t_{j_1})$
  end if
  if $\llbracket t_{j_2}, i, 0 \rrbracket \in P_{s_\tau}$ **then**
    $\mathcal{I}^i_{\mathrm{X'}} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(t_{j_2})$
  end if
  $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathrm{X}} \cap \mathcal{I}^i_{\mathrm{X'}}$
  if $\mathcal{I}^i = \phi$ **then**
    $\mathcal{A}^i = 1^\ell$
  else
    $\mathcal{A}^i \leftarrow \mathsf{IntEnc}\,(\mathcal{I}^i)$
  end if
end for
Let $\mathcal{A}^i = \{h^i_1, \ldots, h^i_\sigma\}$, where $\sigma \leq \ell$. For every $h^i_{q_\rho} \in \mathcal{A}^i$, publish $H_c(\|^n_{i=1} h^i_{q_\rho})$, where $q_\rho \in \{1, \ldots, \sigma\}$.

5.4 Obfuscated Decision Tree Evaluation

In the following, we discuss the process of evaluating the obfuscated program on input sequence $\mathcal{B} = (x_1, \ldots, x_n)$, where $x_i \in \{0,1\}^\ell$.

Algorithm 5.6 Evaluation (with embedded α hashes published by $\mathcal{O}_{\mathcal{D}}$)

Input: $\mathcal{B} = (x_1, \ldots, x_n)$, ℓ, d.
Output: 0 or 1.
1: **for** $i = 1$ to n **do**
2: $\quad \mathcal{E}^i \leftarrow 1^\ell \cup \{H(f_0(x_i)), \ldots, H(f_{\ell-1}(x_i))\}$
3: **end for**
4: Denote $\mathcal{E}_i = \{h_1^i, \ldots, h_\sigma^i\}$, $\sigma \leq \ell + 1$. For every $h_{q_\sigma}^i \in \mathcal{E}^i$, return 1, if $H_c(\|_{i\in[n]} h_{q_\sigma}^i)$ is contained in the set of α hashes published by $\mathcal{O}_{\mathcal{D}}$, else return 0.

5.5 Correctness and Efficiency

In this section we analyze the correctness of the our construction and evaluate the efficiency of the proposed obfuscator.

Lemma 5.3 (Correctness.). *Consider Algorithms 5.5 and 5.6 and an input $(x_i)_{i\in[n]}$, where $x_i \in \{0,1\}^\ell$. For every $[\![t_j, i, b]\!] \in P_{s_\tau}$, where integer $t_j \in [0, 2^\ell - 1]$, $i \in \{1, \ldots, n\}$, $b \in \{0, 1\}$, every $\mathcal{S} = (s_1, \ldots, s_{2^d+1})$, where $s_\tau \in \{0, 1\}$, every set of α hashes output by Algorithm 5.5 and every input $(x_i)_{i\in[n]}$, such that $C((x_i)_{i\in[n]}) = 1$, Algorithm 5.6 outputs* 1.

Proof. Algorithm 5.5 calculates set of encodings $\mathcal{A}^i \leftarrow \mathsf{IntEnc}\ (\mathcal{I}^i)$ for every $i \in [n]$. If $\exists\, i$ such that $[\![t_j, i, b]\!] \notin P_{s_\tau}$, then $\mathcal{A}^i \leftarrow 1^\ell$.

Let $(x_i)_{i\in[n]}$ be an accepting input. Then from Definition 4.3, it is evident that $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. From Lemma 5.2, we can conclude that if $x_i \in (c_i, c_i + w_i]$, then there exists a unique $h_k^i \in \mathcal{E}^i$, such that $h_k^i \in \mathcal{A}^i$. If $\exists i$, such that $[\![t_j, i, b]\!] \notin P_{s_\tau}$, then $h_k^i = 1^\ell = \mathcal{A}^i$. Thus, for an accepting input $(x_i)_{i\in[n]}$, there exists a unique $(h_{k_1}^1, \ldots, h_{k_n}^n)$, where $h_{k_i}^i \in \mathcal{E}^i$ and $H_c(h_{k_1}^1 \| \ldots \|_{k_n}^n)$ will be contained in the set of α hashes published by $\mathcal{O}_{\mathcal{D}}$ and Algorithm 5.6 will correctly output 1.

If $C((x_i)_{i\in[n]}) \neq 1$, then from Lemma 5.2 it is evident that $\nexists h_k^i$ such that $h_k^i \in \mathcal{E}^i$ and $h_k^i \in \mathcal{A}^i$. Thus $H_c(h_{k_1}^1 \| \ldots \| h_{k_n}^n)$ will not be contained in the set of α hashes published by the obfuscator $\mathcal{O}_{\mathcal{D}}$ with overwhelming probability and Algorithm 5.6 will correctly reject the input.

Efficiency. Let $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $\ell = \ell(\lambda)$, $m = m(\lambda)$ and $d = d(\lambda)$. Storing $[\![t_j, i, b]\!]$ at a decision node requires $(\ell + \log n + 1)$ bits. Since $|\mathcal{A}^i| < 2\ell$, where each hash value is of ω bits, overall storage (in bits) along an accepting path P_{s_τ} is given by $cost_{s_\tau} < 2(n\ell\omega + \ell + \log n + 1)$. Since $|\mathcal{S}| = m + 1$, # accepting paths $< m + 1$, the overall complexity for storing the obfuscated decision tree is given by $\mathrm{O}\big(m\ell(\frac{\log n}{\ell} + n\omega)\big)$. Evaluation requires $\ell + 1$ computations at each decision node, with the overall running time of the order $\mathrm{O}\big(\ell^d\big)$.

Lemma 5.4 (Polynomial Slowdown). *Let $\lambda \in \mathbb{N}$ be the security parameter and ℓ, n, d be polynomials in λ. Define $\mathcal{T}_\lambda$ to be a special family of evasive decision trees, where $d = 5$ and $\ell = \frac{\lambda}{4}$. For every $C \leftarrow \mathcal{T}_\lambda$, there exists a polynomial p such that the running time of $\mathcal{O}(C)$ is bounded by* $p(|C|, \lambda)$.

Proof. Let $C \leftarrow \mathcal{T}_\lambda$ computes whether an input $(x_i)_{i=1}^n$ is contained in the decision region defined by intervals $(c_i, c_i + w_i]$, where $w_i \in [0, w_{max})$. From Lemma 4.2, we get $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$, which specifies the maximum width of the intervals. For evasiveness, we require $\ell - \frac{\lambda}{n} \geq 0$, which gives $\ell n \geq \lambda$. Now, for $\ell = \frac{\lambda}{4}$ and $n = 4$, we get $\ell n = \lambda$, which is a feasible condition for evasiveness. Since $d = 5$, we equate $d - 1 = n$. The cost of evaluating $\mathcal{O}_\mathcal{D}$ is given by $\ell^n = (\frac{\lambda}{4})^4$.

Parameters for Secure Construction of $\mathcal{O}_\mathcal{D}$. The aim of this section is to explain how to choose the necessary parameters that adhere to the restrictions of a secure and efficient obfuscator.

Our a priori knowledge on the parameters are as follows: $\ell = \ell(\lambda)$, $d = d(\lambda)$, $n = n(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter. For evasiveness, the maximum width of the intervals representing the *decision regions* should be $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$ keeping to Lemma 4.2. Adding to that, we require $d \leq n + 1$. We give some example parameters along with their bit-security in Table 1.

d	ℓ	n	λ	α	β	Total Cost (in bits)
5	64	4	128	127^4	65^4	80×65^4
3	64	2	64	127^2	65^2	80×65^2

Table 1: Example parameter sets for an obfuscated decision tree with $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$, $\omega = 80$ bits and one accepting path, where ω is the output size of hash function H_c. For each parameter set, we calculate the number of hashes α (maximum value) and β computed by algorithms 5.5 and 5.6 respectively, along with the overall complexity.

6 Proof of VBB Security

Our VBB security is based upon the *random oracle* paradigm. The security of our construction relies upon the existence of two *preimage resistant hash functions* $H : \{0,1\}^* \rightarrow \{0,1\}^\omega$ and $H_c : \{0,1\}^* \rightarrow \{0,1\}^q$, which we model as random oracles. Our objective is to show that a PPT adversary having access to the obfuscated function has no advantage over a simulator having oracle access to the function. This is achieved by a simulating the execution of the adversary and outputting what the adversary does, such that the adversary cannot distinguish between the simulation and the real environment, a notion called simulation-based obfuscation.

Theorem 6.1. *Let $\lambda \in \mathbb{N}$ be the security parameter and $\ell = \ell(\lambda)$ and $n = n(\lambda)$. Let $\mathcal{C} = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ be an ensemble of decision tree evasive distributions (as in Definition 4.4). For random oracles $H : \{0,1\}^* \rightarrow \{0,1\}^\omega$ and $H_c : \{0,1\}^* \rightarrow \{0,1\}^q$, the decision tree obfuscator $\mathcal{O}_\mathcal{D}$ is a VBB obfuscator.*

Proof. As evident from Lemma 5.2, $\mathcal{O}_{\mathcal{D}}$ satisfies functionality preservation. Lemma 5.4 shows that the obfuscator causes polynomial slowdown. Thus it suffices to show that there exists a (non-uniform) PPT simulator $\mathcal{S}$ for every (non-uniform) PPT adversary $\mathcal{A}$, such that for an ensemble of decision tree evasive distributions C_λ, the following holds:

$$\underset{C \leftarrow C_\lambda}{Pr}[\, C(\mathcal{A}\, \mathcal{O}(1^\lambda, C))) = \varphi(C)] - \underset{C \leftarrow C_\lambda}{Pr}[\, \mathcal{S}^C(1^\lambda) = \varphi(C)] \Big| \leq \mu(\lambda)$$

Every $C \leftarrow C_\lambda$ identifies unique $(c_1, \ldots, c_n) \leftarrow X_n$ and$(w_1, \ldots, w_n) \leftarrow [0, w_{max})^n$ and on input $(x_1, \ldots, x_n)$ checks if $x_i \in (c_i, c_i{+}w_i]$, for all $i \in [n]$. Let $\mathcal{O}(1^\lambda, C) = \{h_1, \ldots, h_\alpha\}$ denote the obfuscation of C. Let $\mathcal{A}$ be a PPT adversary that takes as input $\mathcal{O}(1^\lambda, C)$ and outputs a predicate of C, denoted by $\varphi(C)$. We use this adversary to design a PPT simulator $\mathcal{S}$ that simulates an execution of $\mathcal{A}$.

Since $\mathcal{A}$ expects the oracles H and H_c, $\mathcal{S}$ provides a simulation of both the oracles. In order to record the choices of the random oracles, $\mathcal{S}$ maintains two tables : $\mathcal{T}_1$ to record responses for queries to H and $\mathcal{T}_2$ to record responses for queries to H_c. Since $\mathcal{S}$ does not have access to $\mathcal{O}(1^\lambda, C)$, it prepares a purported obfuscation of C as follows: It takes as input $\pi = (\alpha,\ \ell,\ n)$ and samples values uniformly at random from the co-domain of H_c to compute the purported obfuscation of C, given by $\{h'_1, \ldots, h'_\alpha\}$.

We assume that $\mathcal{A}$ makes polynomial queries to both the random oracles. We use the notation $\mathcal{A} \rightarrow u$ to indicate that adversary $\mathcal{A}$ is making a random oracle query u and $\mathcal{A} \leftarrow v$ to indicate that v is returned to $\mathcal{A}$ as a response to this query. When $\mathcal{A}$ queries random oracle H with $\mathbf{u}^*$, the simulator looks up v such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$ and returns it to the adversary. If no such v exists, then the simulator assigns v with a value chosen uniformly at random from the co-domain of H, registers the value in $\mathcal{T}_1$ and returns it to $\mathcal{A}$.

When $\mathcal{A}$ makes a query $\mathbf{h}^*$ to the random oracle H_c, the simulator checks for a val such that $(\mathbf{h}^*, val) \in \mathcal{T}_2$ and returns it to $\mathcal{A}$. If there are no entries corresponding to $\mathbf{h}^*$, the simulator parses $\mathbf{h}^*$ as a sequence of n strings $(h_i^*)_{i \in [n]}$ and looks up table $\mathcal{T}_1$ to find an entry corresponding to each string.

If there does not exist any entry in $\mathcal{T}_1$ corresponding to the parsed strings, then $val \leftarrow \{0,1\}^q$, entry $(\mathbf{h}^*, val)$ is recorded in $\mathcal{T}_2$ and val is returned it $\mathcal{A}$. If there exists a unique u such that $(u, h_i^*) \in \mathcal{T}_1$, then the simulator calculates $j \leftarrow \ell - |u|$, where $|u|$ denotes the bit length of u, and $x_i \leftarrow u \times 2^j$. Since u corresponds to the representative value μ_j for a correct input, adding j 0's yields an accepting input for C. Eventually, the simulator queries the oracle C with the (x_i). If C returns 1, $\mathcal{S}$ determines the c_i's and w_i's, calculates pairs (u, v) and registers the entries in $\mathcal{T}_1$. Thereafter the simulator calculates the α input entries of $\mathcal{T}_2$,maps them to the α entries from the purported set $\{h'_1, \ldots, h'_\alpha\}$, registers the pairs in $\mathcal{T}_2$ and returns val to the adversary. If there are multiple entries in $\mathcal{T}_1$, the simulation halts. The simulation in form of pseudo code is presented as follows:

Algorithm 6.1 Simulator $\mathcal{S}^C(1^\lambda, \pi)$

Initialize:

for $i = 1$ to α **do**
  $h'_i \leftarrow \$ \{0,1\}^q$ ▷ Set denoting purported obfuscation
end for
$\mathcal{T}_1 \leftarrow (\,); \mathcal{T}_2 \leftarrow (\,)$ ▷ Initialize tables to record choices of the random oracles
$counter \leftarrow 0$

/* Begin simulation for adversary $\mathcal{A}$*/

Hash Query:

$\mathcal{A} \rightarrow \mathbf{u}^*$ ▷ $\mathcal{A}$ submits hash query to H
if $(\mathbf{u}^*, v) \notin \mathcal{T}_1$ **then**
  $v \leftarrow \$ \{0,1\}^\omega$
  $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup (\mathbf{u}^*, v)$
end if
if $\exists\, w\ (\mathbf{u}^* \neq w)$, such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$ and $(w, v) \in \mathcal{T}_1$ **then**
  HALT ▷ Simulation aborts
end if
$\mathcal{A} \leftarrow v$ ▷ Return v as response to $\mathcal{A}$

$\mathcal{A} \rightarrow \mathbf{h}^*$ ▷ $\mathcal{A}$ submits hash query to H_c
if $(\mathbf{h}^*, val) \notin \mathcal{T}_2$ **then**
  Parse $\mathbf{h}^*$ as sequence $(h^*_i)_{i \in [n]}$
  for $i = 1$ to n **do**
    if $\exists!\, u$, such that $(u, h^*_i) \in \mathcal{T}_1$ **then**
      $counter \leftarrow counter{+}{+}$
      $j \leftarrow \ell - |u|$
      $x_i \leftarrow u \times 2^j$
    end if
  end for
  if $(counter < n)$ **then**
    $val \leftarrow \$ \{0,1\}^q$
    $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (\mathbf{h}^*, val)$
  else
    $\mathcal{S} \rightarrow (x_1, \ldots, x_n)$ ▷ $\mathcal{S}$ submits $(x_1, \ldots, x_n)$ to the oracle C
    if $\mathcal{S} \leftarrow 1$ **then**
      Determine $(c_1, \ldots, c_n)$ and $(w_1, \ldots, w_n)$ using Binary Search
      Calculate pairs (u, v)
      $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup (u, v)$
      if $\exists\, \mathrm{u}_1, u_2\ (u_1 \neq u_2)$, such that $(u_1, v) \in \mathcal{T}_1$ and $(u_2, v) \in \mathcal{T}_1$ **then**
        HALT ▷ Simulation aborts
      else
        for $i = 1$ to α **do**
          Calculate h_i
          $val_i \leftarrow h'_i$
          $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (h_i, val_i)$
        end for
      end if
    else
      $val \leftarrow \$ \{0,1\}^q$
      $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (\mathbf{h}^*, val)$
    end if
  end if
end if
$\mathcal{A} \leftarrow val$ ▷ Return val as a response to $\mathcal{A}$

At any point, the simulated view is identical to the real view such that the adversary cannot distinguish between the real and purported obfuscation.

We now analyze the scenario where the simulator fails due to conflicts in table $\mathcal{T}_1$ and show that the probability of such conflicts is negligible in λ.

Lemma 6.1. *Let $\lambda \in \mathbb{N}$ be the security parameter and let ℓ, n, α are polynomials in λ. Let C_λ be an ensemble of decision tree evasive distributions and let $\mathcal{O}(1^\lambda, C)$ denote the obfuscation of $C \leftarrow C_\lambda$. Consider Algorithm 6.1 and random oracles $H : \{0,1\}^* \to \{0,1\}^{\omega(\lambda)}$ and $H_c : \{0,1\}^* \to \{0,1\}^{q(\lambda)}$. Let $\eta = \eta(\lambda)$ be the number of entries in $\mathcal{T}_1$, then there exists a negligible function $\mu(\lambda)$ such that:*

$$\Pr_{C \leftarrow C_\lambda}[\, \mathcal{S}^C(1^\lambda, \pi) = \perp] \;\leq\; \mu(\lambda)$$

where $\mathcal{S}$ is a (non-uniform) PPT algorithm having oracle access to the function C.

Proof. The simulation fails when there is a conflict in table $\mathcal{T}_1$ and $\mathcal{S}$ halts. Conflicts may arise when $\mathcal{S}$ has responded to a random oracle query $\mathbf{u}^*$ to H with $v \leftarrow\!\$\, \{0,1\}^\omega$ and later, on the hash query $\mathbf{h}^*$ to H_c, it makes a call to oracle C and populates $\mathcal{T}_1$ with a pair (w, v) such that $w \neq \mathbf{u}^*$. The probability that a conflict occurs in $\mathcal{T}_1$ is equal to the probability that a hash value is same as at least one of the η values in table $\mathcal{T}_1$. Since $H : \{0,1\}^* \to \{0,1\}^{\omega(\lambda)}$, there are $2^{\omega(\lambda)}$ choices for a hash value. When there are no entries in $\mathcal{T}_1$, the collision probability is 0, when there is one entry in $\mathcal{T}_1$, the collision probability is $\frac{1}{2^{\omega(\lambda)}}$ and continuing the same way, when there are $\eta - 1$ entries in $\mathcal{T}_1$, the probability of collision is $\frac{(\eta-1)}{2^{\omega(\lambda)}}$. Assuming all the samples are independent, the probability with which $\mathcal{S}^C(1^\lambda, \pi)$ fails is given by:

$$\begin{aligned}\Pr_{C \leftarrow C_\lambda}[\, \mathcal{S}^C(1^\lambda, \pi) = \perp\,] &= \frac{1 + \cdots + (\eta - 1)}{2^{\omega(\lambda)}} \\ &= \frac{\eta^2 - \eta}{2^{\omega(\lambda)+1}} \\ &\leq \mu(\lambda)\end{aligned}$$

7 Conclusion

In this paper, we have introduced a new special-purpose VBB obfuscator for binary evasive decision trees. While doing so, we have presented an encoder for hiding parameters in an interval-membership function. Our security analysis follows the Random Oracle paradigm [7] [28] [22] [25]. To the best of our knowledge, our construction provides the first non-interactive solution for privacy-preserving classification with decision trees. Furthermore, our methods rely upon hash functions as opposed to computationally expensive cryptographic primitives used by the state-of-art protocols. Finally our construct provides computational security against model-extraction attacks for decision trees that are evasive in nature.

References

1. Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 145–161. Springer, 2020.
2. Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3):137–150, 2015.
3. Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *Theory of Cryptography Conference*, pages 26–51. Springer, 2014.
4. Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual international cryptology conference*, pages 1–18. Springer, 2001.
5. Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *European symposium on research in computer security*, pages 424–439. Springer, 2009.
6. Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Annika Paus, Ahmad-Reza Sadeghi, Thomas Schneider, and Vladimir Kolesnikov. Efficient privacy-preserving classification of ecg signals. In *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 91–95. IEEE, 2009.
7. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
8. Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. A simple obfuscation scheme for pattern-matching with wildcards. In *Annual International Cryptology Conference*, pages 731–752. Springer, 2018.
9. Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. A simple obfuscation scheme for pattern-matching with wildcards. In *Annual International Cryptology Conference*, pages 731–752. Springer, 2018.
10. Edward S Blurock. Automatic learning of chemical concepts: Research octane number and molecular substructures. *Computers & chemistry*, 19(2):91–99, 1995.
11. Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.
12. Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. *Cryptology ePrint Archive*, 2014.
13. Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 498–507, 2007.
14. Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Annual International Cryptology Conference*, pages 455–469. Springer, 1997.
15. Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Annual International Cryptology Conference*, pages 455–469. Springer, 1997.
16. Ran Canetti, Guy N Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *Theory of Cryptography Conference*, pages 72–89. Springer, 2010.

17. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder VL Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. *Cryptology ePrint Archive*, 2022.
18. Giovanni Di Crescenzo. Cryptographic formula obfuscation. In *International Symposium on Foundations and Practice of Security*, pages 208–224. Springer, 2018.
19. Christine Decaestecker, Myriam Remmelink, Isabelle Salmon, Isabelle Camby, Denis Goldschmidt, Michel Petein, Philippe Van Ham, Jean-Lambert Pasteels, and Robert Kiss. Methodological aspects of using decision trees to characterise leiomyomatous tumors. *Cytometry: The Journal of the International Society for Analytical Cytology*, 24(1):83–92, 1996.
20. Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.
21. Steven D Galbraith and Lukas Zobernig. Obfuscated fuzzy hamming distance and conjunctions from subset product problems. In *Theory of Cryptography Conference*, pages 81–110. Springer, 2019.
22. Shafi Goldwasser and Guy N Rothblum. On best-possible obfuscation. In *Theory of Cryptography Conference*, pages 194–213. Springer, 2007.
23. Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–621. IEEE, 2017.
24. Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 371–380, 2018.
25. Ilan Komargodski and Eylon Yogev. Another step towards realizing random oracles: non-malleable point obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 259–279. Springer, 2018.
26. Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against model stealing attacks using deceptive perturbations. *arXiv preprint arXiv:1806.00054*, 2018.
27. Lin Liu, Rongmao Chen, Ximeng Liu, Jinshu Su, and Linbo Qiao. Towards practical privacy-preserving decision tree training and evaluation in the cloud. *IEEE Transactions on Information Forensics and Security*, 15:2914–2929, 2020.
28. Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *International conference on the theory and applications of cryptographic techniques*, pages 20–39. Springer, 2004.
29. Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. A framework for the extraction of deep neural networks by leveraging public data. *arXiv preprint arXiv:1905.09165*, 2019.
30. Erwin Quiring, Daniel Arp, and Konrad Rieck. Forgotten siblings: Unifying attacks on machine learning and digital watermarking. In *2018 IEEE European symposium on security and privacy (EuroS&P)*, pages 488–502. IEEE, 2018.
31. Steven Salzberg, Rupali Chandar, Holland Ford, Sreerama K Murthy, and Richard White. Decision trees for automated identification of cosmic-ray hits in hubble space telescope images. *Publications of the Astronomical Society of the Pacific*, 107(709):279, 1995.
32. Craig Silverstein and Stuart M Shieber. Predicting individual book use for off-site storage using decision trees. *The Library Quarterly*, 66(3):266–293, 1996.

33. Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.
34. Raymond KH Tai, Jack PK Ma, Yongjun Zhao, and Sherman SM Chow. Privacy-preserving decision trees evaluation via linear functions. In *European Symposium on Research in Computer Security*, pages 494–512. Springer, 2017.
35. Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.
36. Anselme Tueno, Yordan Boev, and Florian Kerschbaum. Non-interactive private decision tree evaluation. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 174–194. Springer, 2020.
37. Hoeteck Wee. On obfuscating point functions. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 523–532, 2005.
38. Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–611. IEEE, 2017.
39. David J Wu, Tony Feng, Michael Naehrig, and Kristin E Lauter. Privately evaluating decision trees and random forests. *Proc. Priv. Enhancing Technol.*, 2016(4):335–355, 2016.
40. David J Wu, Tony Feng, Michael Naehrig, and Kristin E Lauter. Privately evaluating decision trees and random forests. *Proc. Priv. Enhancing Technol.*, 2016(4):335–355, 2016.
41. Huadi Zheng, Qingqing Ye, Haibo Hu, Chengfang Fang, and Jie Shi. Bdpl: A boundary differentially private layer against machine learning model extraction attacks. In *European Symposium on Research in Computer Security*, pages 66–83. Springer, 2019.
42. Yifeng Zheng, Huayi Duan, and Cong Wang. Towards secure and efficient outsourcing of machine learning classification. In *European Symposium on Research in Computer Security*, pages 22–40. Springer, 2019.