# Obfuscating Evasive Decision Trees

Shalini Banerjee*[0000−0002−8844−0427], Steven D. Galbraith[0000−0001−7114−8377], and Giovanni Russello[0000−0001−6987−0803]

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

**Abstract.** We present a new encoder for hiding parameters in an interval membership function. As an application, we design a simple and efficient *virtual black-box obfuscator* for *evasive* decision trees. The security of our construction is proved in the random oracle model. Our goal is to increase the class of programs that have practical and cryptographically secure obfuscators.

**Keywords:** program obfuscation · cryptographic hash functions · decision trees

## 1 Introduction

Program obfuscation has received considerable attention by the cryptographic community in recent years. An obfuscator $\mathcal{O}$ is a probabilistic polynomial-time algorithm that transforms a program $C$ to a semantically equivalent counterpart $\tilde{C}$, such that a secret that is efficiently computable from $C$, is hard to extract given $\tilde{C}$.

The definitional framework of program obfuscation was given by Barak *et al.* in their seminal work [4] using a simulation-based security paradigm. They established the notion of *virtual black-box* (VBB) obfuscation, where a polynomial-time adversary $\mathcal{A}$ that takes input $\tilde{C}$ has a negligible advantage over a polynomial-time simulator $\mathcal{S}$ who only has oracle access to $C$; in short, anything that is efficiently computable from $\tilde{C}$, can also be computed efficiently from the input-output access to the program. Their main results rule out the possibility of designing efficient obfuscators for a generic class of programs. However, obfuscators for *specific* families of programs may be achievable. Canetti [14] shows the construction of an efficient obfuscator for point functions, that achieves a relaxed notion of virtual black-box security using a probabilistic hashing algorithm $\mathcal{R}$, which imitates the 'useful' properties of a random oracle. The obfuscated program stores $\mathcal{R}(x)$, where $x$ is sampled uniformly from a superlogarithmic min-entropy distribution, such that on input $y$, outputs 1 if $\mathcal{R}(x) = \mathcal{R}(y)$. Such favoring assertions were followed by designing efficient VBB obfuscators for a special family of functions (*evasive functions* [3]) which achieve the goal that a PPT adversary cannot distinguish between the obfuscation of $C$ drawn randomly from the function family and obfuscation of a function that always outputs

* corresponding author

zero. Notable works in this direction include obfuscation of point-functions [30], pattern-matching with wildcards [7,8], compute-and-compare programs [21,31], fuzzy-matching for Hamming distance [19], hyperplane membership [15], etc.

In this paper, we focus on a new technique for *encoding interval membership functions*. This is motivated by designing an efficient *virtual black-box obfuscator* for evasive decision trees (see Definition 3).

### 1.1 Privacy-Preserving Classification using Decision Trees

In the interest of establishing the usefulness and significance of obfuscating decision trees, we provide a brief overview on privacy-preserving classification using decision tree classifiers.

Decision tree classifiers are extensively used for prediction and analysis in sensitive applications such as spam detection, medical or genomics, stock investment, etc. [9,17,28].

Consider an example of a medical facility (model-provider) who designs a model from sensitive profiles of patients to diagnose certain disease. The model is then outsourced to a cloud server to provide classification to a user who wants to make a prediction about her health. If the model is leaked, the sensitive training data will be disclosed [1,18], breaching the HIPAA[1] compliance. What's more, the user does not want to reveal her queries and classification results to the cloud server. This calls for *privacy-preserving classification techniques*, where the model should be hidden from anyone but the model-provider, and prediction queries/classification must remain private to the user, such that no leakage of useful information happen during the classification phase.

The state-of-art privacy-preserving classification solutions employ an *interactive* approach: encrypt and outsource the model to cloud server, where it processes encrypted queries and forwards encrypted classification to the users. The solutions involve multiple rounds of communication and rely upon expensive cryptographic computations using fully-homomorphic encryption (FHE) [10,11], garbled circuits [5,6], etc. 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.* [11] present a comparison protocol between model-provider and the user for each node in the decision tree using FHE methods. Tai *et al.* [25] make use of multiple communication rounds to transfer the path costs and encrypted labels to the client. The authors of [16] design an FHE-based solution (SortingHat) to secure prediction queries and classification results with reduced communication costs, but do not guarantee the privacy of the model.

Our motivation for obfuscating decision trees is to eliminate interaction between user and the model-provider/cloud server. In particular, we aim to construct an efficient non-interactive solution to privacy-preserving classification

[1] Health Insurance Portability and Accountability Act of 1996

with evasive decision trees. We now explain why we do not consider obfuscating arbitrary decision trees. If a decision tree can be learned from the input-output behaviour of the model, then protecting the privacy of the model is impossible. Note that, learning a decision tree means identifying the decision nodes and input attributes associated with them, and identifying the accepting nodes. Tramer *et al.* [29] show that a decision tree can be learned through $m \cdot \log_2(b/\epsilon)$ oracle queries, where $m \cdot \log_2(b/\epsilon)$ oracle queries, where $m$ is the number of internal nodes, $b$ is the minimum width of an interval in a node, and $\epsilon$ is the specified precision value; they call it *model extraction attack*. To prevent such attacks, the existing literature observes API calls to issue warnings [23,27] or adds perturbations [24,32]. However, since there are no theoretical restrictions on the number of prediction queries made by a user [26], limiting them is not reasonable approach towards thwarting such attacks. We define a special class of decision trees, for which it is hard to find an accepting input, such that an efficient algorithm cannot extract the model except with negligible probability; we call such decision trees *evasive*, and claim that if a decision tree is not evasive, then it is impossible to protect the privacy of the model, and hence there is no choice but to restrict to evasive decision trees.

Our restriction to evasive decision trees means that our methods are not generally applicable to classifiers produced by machine learning algorithms. The whole point of machine learning is to learn a model based on training data. Unless the training data is very specifically chosen, then it follows that a classifier produced by a machine learning process will be learnable from black box access, because we can use the black box classifier to generate new training data and then run a learning algorithm on the new training data to produce a new classifier for the same model. This means that classifiers produced by machine learning in the real world are generally not evasive and hence preserving privacy of the model is impossible. Similarly, if one starts with an evasive decision tree and evaluates it on random inputs then all of them will be rejected. Hence such data is not useful training data for a machine learning tool. In conclusion, this paper is about human-made evasive decision trees, not decision trees produced by machine learning tools.

Lockable obfuscation (also called compute-and-compare obfuscation) [21,31] is a very general tool which encodes a class of branching programs under learning-with-errors (LWE) assumption. It could be employed to build a decision tree obfuscator, by writing the decision tree as a circuit. Nevertheless, we focus on solutions that are simpler and potentially more practical. In [12], the authors initiate a theoretical investigation on decision tree obfuscation based on indistinguishability obfuscation [20] which is the 'best-possible' obfuscation from the point of view of VBB, but does not guarantee the privacy of the decision tree model. We aim to achieve stronger notions of security that allow us to protect the privacy of the model.

Our work has been used in a recent paper by Banerjee *et al.* [2]. The authors aim to provide security against reverse-engineering of PLC programs concerning critical industrial facilities (nuclear-enrichment etc.) in order to prevent infamous

attacks such as Stuxnet [22]. They extend our scheme to design a platform ObfCP, with empirical results that reflect its efficiency in real-time applications.

### 1.2 Our Contributions

We present a new technique for encoding interval membership functions. As an application, we construct an efficient VBB obfuscator for *evasive decision trees* (see Definition 6). We focus on trees of bounded number of inputs and depth.

Note that, we do not consider privacy-preserving methods to construct the model. How the decision tree is constructed is out of the scope of this study. A technical briefing of our construction follows.

**Technical Overview.** We consider decision trees that perform binary classification based on the values of $n$ attributes. Attributes are represented as $\ell$-bit strings $x_i$, and are interpreted as integers in $[0, 2^\ell)$. A decision tree is a full binary tree of depth $d$. Internal node $v_j$ (also called *decision* node) associates threshold $t_j$, where $t_j$ is an integer between 0 and $2^\ell - 1$. Each decision node tests $x_i \leq t_j$ for some $i$. The leaf nodes $(s_1, \ldots, s_{2^d})$ are labelled 0 (reject) or 1 (accept). Hence the decision tree is represented by the pairs $[\![t_j, i]\!]$, and the labels on the leaf nodes.

Without loss of generality we may assume that, for any specific path from the root to an accepting leaf, $x_i$ is compared at most twice. Hence each accepting path corresponds to a sequence of interval membership predicates $x_i \in (c_i, c_i + w_i]$. The key observation is that membership $x_i \in (c_i, c_i + w_i]$ can be expressed as a union of distinct predicates $x_i \in [a, a+2^p)$ for certain pairs $(a, p)$. Each such predicate can be turned into a point function predicate and hence be obfuscated using hashing. We explain the details in the next paragraph.

Let $f_i : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$ for $i \in \{0, 1, \ldots, \ell - 1\}$. Calculate intersection of sub-intervals $\mathcal{I}^i$ corresponding to $(c_i, c_i+w_i]$ (of the form $[a, a+2^p)$). Encode each entry in $\mathcal{I}^i$ using $H(f_p(a))$, where $H : \{0,1\}^* \to \{0,1\}^\omega$ is a cryptographic hash function; call the set $\mathcal{B}^i$. Note that, this method converts interval membership predicate $x_i \in [a, a+2^p)$ into a point function predicate that determines whether $H(f_p(a))$ is equal to $H(f_q(x_i))$ for some $q \in \{0, \ldots, \ell - 1\}$. Finally, for each encoding in $\mathcal{B}^i$, concatenate $n$ entries sorted in the order of $i$, apply cryptographic hash function $H_c : \{0,1\}^* \to \{0,1\}^q$, and publish the set of hashes. Reordering the nodes in the 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 $(x_i)_{i \in [n]}$, compute the set of encodings $\mathcal{E}^i$ by calculating $H(f_q(x_i))$, where $q \in \{0, \ldots, \ell - 1\}$ and for each encoding, concatenate $n$ entries sorted in order of $i$, and apply $H_c$. For an accepting input, one of the hashes computed by the evaluation procedure will be contained in the set of hashes published by the obfuscator.

## 2 Preliminaries

We denote by $|S|$, size of a 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 by $\lfloor x \rfloor$ the integral part of $x$, where $x \in \mathbb{R}$. We use $\log_2(n)$ to denote the power to which 2 should be raised to obtain the value $n \in \mathbb{N}$. We denote the binary encoding of $n$ as $r_{\ell-1} \cdot 2^{\ell-1} + \cdots + r_0 \cdot 2^0$ 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 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 $a \cdot \lambda^c$, for some constants $a, c$. A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is called negligible in $n$, if it grows slower than $n^{-c}$, for every constant $c$. We measure negligibility with respect to the security parameter $\lambda$. We use $x \leftarrow\$ X$ to denote $x$ is drawn uniformly at random from the space $X$. Finally, we let $\|_{i=1}^{n} (a_i)$ denote concatenation $a_1 \| a_2 \| \cdots \| a_n$ of the sequence $(a_i)_{i \in [n]}$.

## 3 Obfuscation Definitions

In this section, we present the standard definition of obfuscation and *distributional virtual black box* (DVBB) security that our obfuscator satisfies.

**Definition 1 (Distributional Virtual Black-Box Obfuscation [3,4]).** *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be a family of polynomial-size programs parameterized by inputs of length $n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be a class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{C}_\lambda$. A PPT algorithm $\mathcal{O}$ is an obfuscator for the distribution $\mathcal{D}$, if it satisfies the following conditions:*

- *Correctness: For every $\lambda \in \mathbb{N}$ and for every $x \in \{0,1\}^{n(\lambda)}$, there exists a negligible function $\mu(\lambda)$, such that:*

$$\Pr_{\mathcal{O}, C \leftarrow \mathcal{D}_\lambda}[\mathcal{O}(C)(x) = C(x)] > 1 - \mu(\lambda)$$

  *where the probability is over the sampling of the program and coin tosses of $\mathcal{O}$.*
- *Polynomial Slowdown: There exists a polynomial $q$ such that for every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{D}_\lambda$, the running time of $\mathcal{O}(C)$ is bounded by $q(|C|)$, where $|C|$ denotes the size of the program.*
- *Virtual Black-Box: For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to $C$, such that for every $\lambda$:*

$$\left| \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{O}, \mathcal{A}}[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{S}}[\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$

  *where $\mu(\lambda)$ is a negligible function.*

**Definition 2 (Evasive Program Collection [3]).** *A distribution of programs* $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ *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 for every input* $x \in \{0,1\}^{n(\lambda)}$

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

*where the probability is taken over the random sampling of* $C$ *from* $\mathcal{D}_\lambda$*.*

## 4 Decision Trees

In this section, we formalize binary decision trees. Without loss of generality, we consider decision trees to be full binary trees and restrict to binary classification. Following this, we introduce *evasive decision trees* and what it means to *learn* a decision tree.

**Definition 3 (Decision Trees).** *Let* $n, d, \ell \in \mathbb{N}$ *and* $(x_i)_{i=1}^n = (x_1, \ldots x_n) \in \mathbb{N}^n$ *be a finite sequence of input elements, where* $x_i$ *is an integer between* 0 *and* $2^\ell - 1$ *that represents the value of some attribute.*

*A decision tree is a representation of a function* $C : [0, 2^\ell)^n \to \{0,1\}$*. It is a full binary tree of depth* $d$ *with internal nodes* $(v_1, \ldots, v_{2^d-1})$ *(where* $v_1$ *is the root,* $v_2$*,* $v_3$ *are nodes at the second level, and so on) and leaf nodes* $(s_1, \ldots, s_{2^d})$*. Leaf node* $s_k \in \{0,1\}$ *gives the value of the function* $C$*. Internal nodes* $v_j$ *are labelled by a pair* $[\![t_j, i]\!]$ *that defines a predicate* $g_j$ *as* $g_j(x_i) = 1$ *if and only if* $x_i \leq t_j$*. To evaluate the tree on an input* $(x_1, \ldots, x_n)$*, one follows a path from the root to a leaf, by taking the left child if* $g_j = 0$ *and the right child if* $g_j = 1$*. The output is the value of the leaf* $s_k$ *at the end point of this walk in the tree.*

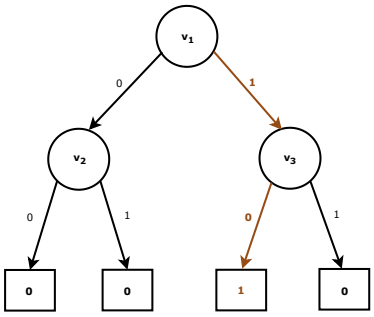


**Fig. 1.** Binary classification with a decision tree: the circular nodes represent decision nodes, and the square nodes represent leaf nodes. Decision nodes are numbered in level-order sequence. The path in brown represents the accepting path with leaf node labeled 1.

To define model extraction resistance we define the *assets* of a decision tree. Note that a decision tree does not necessarily have a unique representation.

**Definition 4 (Asset of Decision Tree).** *Let $C$ be the function represented by a decision tree. We define* $\mathsf{asset}(C)$ *to be a sequence of pairs $[\![t_j, i]\!]$ and a sequence of leaf nodes $(s_1, \ldots, s_{2^d})$ such that the corresponding decision tree from Definition 3 defines the same function $C$.*

Without loss of generality we may assume $x_i$ to be compared at most twice along an accepting path. We stress that different accepting paths may arise from different comparisons of $x_i$. It follows that each accepting path is checking $x_i \in (c_i, c_i + w_i]$ for some $c_i$ and $w_i$. As we now explain, for evasiveness we need the $w_i$ to be not too large, so we introduce an upper bound $w_{max} \in \mathbb{N}$.

**Definition 5 (Decision Region).** *Let $n$, $\ell$, $w_{max} \in \mathbb{N}$. Let $c_i$, $c_i + w_i$ be integers between 0 and $2^\ell - 1$ and $w_i \in (0, w_{max}]$. Define a decision region as the hyper-rectangle formed by $n$ intervals $(c_i, c_i + w_i]$.*

**Evasive Function Family.** As already explained, it is impossible to hide the assets in a learnable decision tree. Hence we study *evasive decision trees*. Throughout the paper, we assume that an adversary knows the domain of inputs, but not the accepting inputs.

**Definition 6 (Evasive Decision Tree Distribution).** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ be a distribution of polynomial-size classification functions represented as decision trees of depth $d(\lambda)$ on $n(\lambda)$ variables. We say $\mathcal{D}$ is evasive, if there exists a negligible function $\mu$ such that for every $\lambda \in \mathbb{N}$, for every input $(x_i)_{i \in [n(\lambda)]}$*

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

In short, Definition 6 requires that for every $(x_i)_{i \in [n]}$, a program $C$ chosen randomly from the distribution evaluates to 1 with negligible probability.

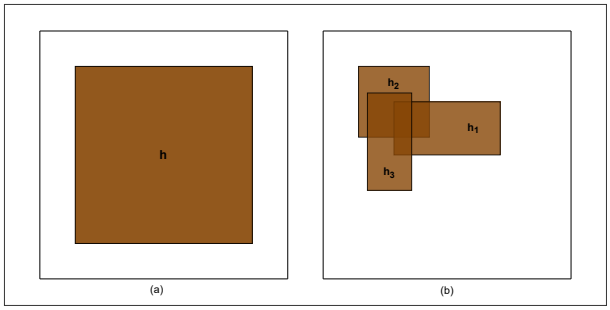


**Fig. 2.** Two example cases of distributions which lead towards non-evasiveness: (a) Decision region is very big. (b) Overlapping decision regions $h_1$, $h_2$ and $h_3$ always accept a point $x$ in their common intersection.

A distribution $X_n \in [0, 2^\ell)^n$ defines a distribution $\mathcal{D}_\lambda$, such that $C \leftarrow \mathcal{D}_\lambda$ computes whether an input $(x_i)_{i\in[n]}$ is accepted or not as follows: sample $(c_1, \ldots, c_n) \leftarrow X_n$ and $(w_1, \ldots, w_n) \leftarrow (0, w_{max}]^n$. The accepted inputs satisfy $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$. For the program collection to be evasive, it is necessary that, for fixed $(x_1, \ldots, x_n)$, the probability is negligible that $x_i \in (c_i, c_i + w_i]$ for every $i$. Thus we require $X_n$ to have large entropy. As uniform distributions provide the highest entropy, this is the best case. However, real-world applications may have accepting regions that are less uniform. The scenarios that lead to non-evasiveness are: (1) the decision regions are too big (so a random $x$ is likely to be in the set); (2) the number of points in the space $[0, 2^l)^n$ representing $(c_1, \ldots, c_n)$ is too few (not enough entropy); (3) the decision regions overlap each other (so one can choose $x$ from the intersection of the regions). Figure 2 shows two example distributions that give non-evasive decision trees. Hence for an evasive collection, the distribution $X_n$ needs to have high entropy and it needs to be "well-spread" (meaning that a set of randomly chosen accepting regions should have empty intersection).

We now calculate some parameters that suffice for evasiveness. We start with uniform distribution on $[0, 2^l)^n$ where $n, \ell$ are polynomials in $\lambda$.

**Lemma 1.** *Let $n, \ell \in \mathbb{N}$. Let $c_i$ be an integer between 0 and $2^\ell - 1$ and $w_i$ be an integer between 1 and $w_{max}$. The number of elements in the decision region $(c_1, c_1 + w_1] \times \cdots \times (c_n, c_n + w_n]$ is at most $(w_{max})^n$.*

*Proof.* Each interval has length at most $w_{max}$. □

**Lemma 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 $(x_i)_{i\in[n]}$. Choose uniformly $c_i \in [0, 2^\ell)$ and $w_i \in (0, w_{max}]$. Then the probability that $(x_i)_{i\in[n]}$ belongs to the decision region defined by $(c_i, c_i + w_i]$, for $i \in [n]$ is not more than $2^{-\lambda}$.*

*Proof.* The total number of points $(x_i)_{i\in[n]}$ in the space $[0, 2^\ell)^n$ is $2^{\ell n}$. By Lemma 1, the decision region defined by $(c_i, c_i + w_i]$, $i \in [n]$ has size $(w_{max})^n$. Hence uniformly sampled input $(x_i)_{i\in[n]}$ is contained in the decision region with probability $\frac{(w_{max})^n}{2^{\ell n}}$. For $w_{max} \leq 2^{(\ell - \frac{\lambda}{n})}$, the above probability is at most $2^{-\lambda}$. □

The result shows that if the intervals $(c_i, c_i + w_i]$'s are uniformly chosen with $w_i \leq 2^{(\ell - \frac{\lambda}{n})}$, then the probability that an input (fixed a priori) belongs to the decision region is negligible in $\lambda$. We now prove that the class of decision tree functions defined by uniform distributions that follow the above mentioned parameter restrictions, forms an evasive program collection.

**Lemma 3.** *Let $\lambda \in \mathbb{N}$ be the security parameter, and let $\ell, n$ be polynomials in $\lambda$. Let $w_{max} = w_{max}(\lambda)$ be a function such that $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$. Let $2^{-\lambda}$ be a negligible function. Let $X_n$ be the uniform distribution in $[0, 2^\ell)^n$, and let $\mathcal{D}_\lambda$ be the corresponding distribution on decision trees that determines if $(x_i)_{i\in[n]}$ belongs to the decision region defined by the $c_i$'s and $w_i$'s. Then $\mathcal{D}_\lambda$ is an evasive program distribution.*

*Proof.* The uniform distribution on $[0, 2^\ell)^n$ defines $\mathcal{D}_\lambda$. We need to show that for every $\lambda \in \mathbb{N}$ and every $(x_i)_{i\in[n]}$, $\Pr_{C\leftarrow\mathcal{D}_\lambda}[C((x_i)_{i\in[n]}) = 1] \leq \mu(\lambda)$. For $C \leftarrow \mathcal{D}_\lambda$, the probability that $(x_i)_{i\in[n]}$ is accepted by $C$ is equal to the probability that $(x_i)$ lies in the product of uniformly chosen intervals $(c_i, c_i+w_i]$ as above. Lemma 2 shows the probability is at most $2^{-\lambda}$, which is a negligible function in $\lambda$. □

We next discuss what it means to *learn* an evasive decision tree. If a decision tree is unlearnable, then there is no model extraction attack.

**Definition 7 (Unlearnable Decision Trees).** *A collection of classification functions $\mathcal{C}$ is unlearnable, if for every polynomial time algorithm $\mathcal{A}$ with oracle access to $C$, there exists a negligible function $\mu$, such that for every $\lambda \in \mathbb{N}$:*

$$\Pr_{C\leftarrow\mathcal{D}_\lambda}[\mathcal{A}^C(1^\lambda) = \mathsf{asset}(C)] \leq \mu(\lambda)$$

*Remark 1.* Note that evasiveness implies unlearnability, because evaluating an evasive function always returns 0 with overwhelming probability and so no information about the function is provided by these queries.

We now explore general distributions. As already discussed, we do not want the support of the distribution to have 'few' points and the decision regions to be clustered along the distribution (clumped distribution) so that we can eliminate the possibilities of non-evasiveness. An important metric for quantifying the randomness of a distribution is *min-entropy*, which measures the difficulty of correctly guessing a sample from a given distribution.

**Definition 8 (Min-entropy).** *A random variable $X$ has a min-entropy, defined by $\mathcal{H}_\infty(X) = -\log[\max_x Pr[X = x]]$ and an average (conditional) min-entropy on a (possibly) correlated random variable $Y$ defined by $\mathcal{H}_\infty(X|Y) = -\log(\mathbb{E}_{y\leftarrow Y}[\max_x Pr[X = x|Y = y]]$.*

**Definition 9 (Decision Tree Min-entropy).** *Let $\ell$, $n$ be polynomials in the security parameter $\lambda \in \mathbb{N}$. Let $w_{max} \leq 2^{(\ell-\frac{\lambda}{n})}$. Let $X$ be a random variable on $[0, 2^\ell)^n$. Then the decision tree min-entropy of $X$ is defined as:*

$$\mathcal{H}_{D,\infty}(X) = -\log\left[\max_{(x_i)_{i=1}^n\in\mathbb{N}^n} Pr\left[X \in \prod_{i=1}^{n}(x_i - w_{max}, x_i]\right]\right.$$

**Lemma 4.** *Let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathcal{D} = D_\lambda$ be an ensemble of distributions over $[0, 2^{\ell(\lambda)})^{n(\lambda)}$. For $w_{max}(\lambda) \leq 2^{\ell(\lambda)-\frac{\lambda}{n(\lambda)}}$, $\mathcal{D}_\lambda$ is decision tree evasive distribution, if min-entropy of $\mathcal{D}_\lambda$ is at least $\lambda$.*

## 5 Obfuscating Evasive Decision Trees

In this section, we introduce a *new technique for encoding interval membership functions*. We follow this with a description of our decision tree obfuscator.

### 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 function $C \in \mathcal{D}_\lambda$ with a depth $d$. We denote decision nodes by $(v_1, \ldots, v_{2^d-1})$ and leaf nodes by $\mathcal{S} = (s_1, \ldots, s_{2^d})$. An accepting path $\mathsf{path}_{s_\tau}$ is defined 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 the predicate $g_j(x_i)$. We assume each element in the input sequence to be compared at most twice along an accepting path. This assumption in reasonable, since any collection of inequalities in the form $x_i \leq t_j$ and $x_i > t_j$ defines an interval, and so is defined by a pair of comparisons. We assume that the evaluation procedure (Algorithm 6) is oblivious to the tuples in $\mathsf{path}_{s_\tau}$. This implies the depth is at most two times the number of input elements.

### 5.2 Encoding Intervals

We now describe our technique to encode integer intervals. We then extend this to construct our decision tree obfuscator.

**Converting an Inequality into Intervals.** A decision node tests $x \leq t$ for some fixed $t \in [0, 2^\ell)$. In other words, the node partitions $[0, 2^\ell)$ into $\mathcal{X} = [0, t+1)$ and $\mathcal{X}' = [t+1, 2^\ell)$. We further divide $\mathcal{X}$ and $\mathcal{X}'$ into disjoint sub-intervals of the form $[a, a + 2^p)$. This is the primary building block of our construction. The formal procedure is given in Algorithms 1 and 2.

**Algorithm 1** $\mathsf{GenInt}_{\mathcal{X}}(t)$

**Input:** $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
**Output:** $\mathcal{I}_{\mathcal{X}} = \{[a_j, a_j + 2^{p_j})\}_{j\in[k]}$
1: $\mathcal{I}_{\mathcal{X}} = \emptyset$ ; $\mathsf{temp} = 0$
2: Compute $k = wt(t+1)$
3: Compute $p_1, \ldots, p_k$, such that $t + 1 = \sum_{j=1}^{k} 2^{p_j}$ and $p_j < p_{j-1}$
4: $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1})\}$
5: **for** $j = 2$ to $k$ **do**
6: $\quad a_j = \mathsf{temp} + 2^{p_{j-1}}$
7: $\quad \mathcal{I}_{\mathcal{X}} = \mathcal{I}_{\mathcal{X}} \cup \{[a_j, a_j + 2^{p_j})\}$
8: $\quad \mathsf{temp} = a_j$
9: **end for**
10: **return** $\mathcal{I}_{\mathcal{X}}$

**Lemma 5.** *Let $\ell \in \mathbb{N}$ and $t \in [0, 2^\ell)$. Consider algorithms $\mathsf{GenInt}_{\mathcal{X}}$ (Algorithm 1) and $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 2). Let $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(t)$ and $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}$. Then $\mathcal{I}_{\mathcal{X}}$ defines sub-intervals of the form $[a, a + 2^p)$ for some $a$ and $p$, whose union is $[0, t+1)$, and $\mathcal{I}_{\mathcal{X}'}$ defines sub-intervals of the form $[b, b + 2^{p'})$ for some $b$ and $p'$, whose union is $[t+1, 2^\ell)$.*

**Algorithm 2** $\mathsf{GenInt}_{\mathcal{X}'}(t)$

**Input:** $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
**Output:** $\mathcal{I}_{\mathcal{X}'} = \{[b_j, b_j + 2^{p'_j})\}_{j \in [k']}$
1: $\mathcal{I}_{\mathcal{X}'} = \emptyset$ ; $\mathsf{temp} = t + 1$
2: Compute $k' = wt(2^\ell - t - 1)$
3: Compute $p'_1, \ldots, p'_{k'}$, such that $2^\ell - t - 1 = \sum_{j=1}^{k'} 2^{p'_j}$ and $p'_j > p'_{j-1}$
4: $\mathcal{I}_{\mathcal{X}'} = [\mathsf{temp}, 2^{p'_1})$
5: **for** $j = 2$ to $k'$ **do**
6: $\quad b_j = \mathsf{temp} + 2^{p'_{j-1}}$
7: $\quad \mathcal{I}_{\mathcal{X}'} = \mathcal{I}_{\mathcal{X}'} \cup \{[b_j, b_j + 2^{p'_j})\}$
8: $\quad \mathsf{temp} = b_j$
9: **end for**
10: **return** $\mathcal{I}_{\mathcal{X}'}$

*Proof.* $\mathsf{GenInt}_{\mathcal{X}}(t)$ divides $[0, t+1)$ into $k$ disjoint sub-intervals $\{[a_j, a_j + 2^{p_j})\}_{j \in [k]}$ such that $\sum_{j=1}^{k} 2^{p_j} = t + 1$ and $p_j < p_{j-1}$. Since $a_1 = 0$, $a_j = a_{j-1} + 2^{p_{j-1}}$, we can write $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1}), [2^{p_1}, 2^{p_1} + 2^{p_2}), \ldots, [\sum_{j=1}^{k-1} 2^{p_j}, \sum_{j=1}^{k} 2^{p_j})\}$, and the union of these intervals is $[0, t + 1)$.

$\mathsf{GenInt}_{\mathcal{X}'}(t)$ divides $[t+1, 2^\ell)$ into $k'$ disjoint sub-intervals $\{[b_j, b_j + 2^{p'_j})\}_{j \in [k']}$ such that $\sum_{j=1}^{k'} 2^{p'_j} = 2^\ell - t - 1$ and $p'_j > p'_{j-1}$. Since $b_1 = t+1$, $b_j = b_{j-1} + 2^{p'_{j-1}}$, we can write $\mathcal{I}_{\mathcal{X}'} = \{[t + 1, t + 1 + 2^{p'_1}), [t + 1 + 2^{p'_1}, t + 1 + 2^{p'_1} + 2^{p'_2}), \ldots, [t + 1 + \sum_{j=1}^{k'-1} 2^{p'_j}, t + 1 + \sum_{j=1}^{k'} 2^{p'_j})\}$, and the union of these intervals is $[t + 1, 2^\ell)$. □

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

**Lemma 6.** *Let $\ell \in \mathbb{N}$. Consider algorithms $\mathsf{GenInt}_{\mathcal{X}}$ (Algorithm 1) and $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 2). Let $c$, $c + w \in \mathbb{Z}$ be such that $0 \leq c < c + w < 2^\ell$. Let $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c + w)$, $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(c)$. Let $\mathcal{I} = \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'} = \{I \cap J : I \in \mathcal{I}_{\mathcal{X}}, J \in \mathcal{I}_{\mathcal{X}'}\} \setminus \emptyset$. Then every interval in $\mathcal{I}$ is of the form $[a, a + 2^i)$, for some $i$, and $|\mathcal{I}| \leq 2\ell - 2$.*

*Proof.* We will prove that if $I \in \mathcal{I}_{\mathcal{X}}$ and $J \in \mathcal{I}_{\mathcal{X}'}$ are such that $I \cap J \neq \emptyset$, then $I \subseteq J$ or $J \subseteq I$. $\mathsf{GenInt}_{\mathcal{X}}(c + w)$ divides $[0, c + w + 1)$ into $k$ disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, where $k$ is the Hamming weight of $\ell$-bit binary encoding of $c + w + 1$. Since $p_j < p_{j-1}$, we can conclude that $2^{p_1} \leq c + w + 1 < 2^{p_1+1}$ and $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1}), \ldots, [2^{p_1} + \cdots + 2^{p_{k-1}}, 2^{p_1} + \cdots + 2^{p_k})\}$, where $\ell > p_1$. Since $c \in [0, c + w + 1)$, we consider the following:

- If $c+1 = 2^q$, where $q \leq p_1$, then $\mathcal{I}_{\mathcal{X}'} = \{[2^i, 2^{i+1})\}_{i \in \{q, q+1, \ldots, \ell-1\}}$, 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 $q \leq p_1$. Let $k'$ be the Hamming weight of $\ell$-bit binary encoding of $2^q-(c+1)$. Then, for $J \in \{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{k'-1}}, c+1+\cdots+2^{p'_{k'}})\}$, 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 $I \in \mathcal{I}_{\mathcal{X}} \setminus \{[0, 2^{p_1})\}$, $I \subseteq J$.
- If $2^{p_1}+\cdots+2^{p_{m-1}} \leq c+1 < 2^{p_1}+\cdots+2^{p_m}$, where $m \leq k$. 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 $p'_1 < \cdots < p'_s \leq p_m < p'_{s+1} < \cdots < p'_{k'}$ for some $s < k'$. Let $I = [2^{p_1}+\cdots+2^{p_{m-1}}, 2^{p_1}+\ldots,+2^{p_m}) \in \mathcal{I}_{\mathcal{X}}$. Then, for a non-empty intersection $J$, let $J \in \mathcal{I}_{\mathcal{X}'}$ be such that $I \cap J \neq \emptyset$. Then $J$ is an element of the set $\{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{s-1}}, c+1+\cdots+2^{p'_s})\}$. Note that, $2^{p_1}+\cdots+2^{p_m} = c+1+\cdots+2^{p'_s}$ 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 Algorithm 1 and 2 that $|\mathcal{I}_{\mathcal{X}}|, |\mathcal{I}_{\mathcal{X}'}| \leq \ell$ (when the Hamming weight of $|\mathcal{X}|$ and $|\mathcal{X}'|$ are $\ell$). Let $\mathcal{I} = \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'}$ and consider any $H \in \mathcal{I}$. By the above, either $H = I$ for some $I \in \mathcal{I}_{\mathcal{X}}$ or $H = J$ for some $J \in \mathcal{I}_{\mathcal{X}'}$. It follows that $|\mathcal{I}| \leq |\mathcal{I}_{\mathcal{X}}| + |\mathcal{I}_{\mathcal{X}'}|$. In the case $H = I$, there is some $J \in \mathcal{I}_{\mathcal{X}'}$ such that $I \subseteq J$, and so $J$ itself is not counted in $|\mathcal{I}|$. Similarly in the case $H = J$ there is some $I \in \mathcal{I}_{\mathcal{X}}$ such that $J \subseteq I$, and so $I$ is not counted in $|\mathcal{I}|$. Hence $|\mathcal{I}| \leq |\mathcal{I}_{\mathcal{X}}| + |\mathcal{I}_{\mathcal{X}'}| - 2 \leq 2\ell - 2$. □

**Calculating the Encodings.** Let $\mathcal{I}$ be a set of sub-intervals, all of the form $[a, a+2^j)$. The encoder (Algorithm 3) receives $\mathcal{I}$ as input, and outputs the set of encodings $E = \{h_1, \ldots, h_{|\mathcal{I}|}\}$. For $i \in \{0, \ldots, \ell-1\}$ define $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$, which defines a function $f_i : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be a cryptographic hash function with $\omega > 2\ell$ so that $H$ is injective when restricted to inputs of length at most $\ell$. A random oracle is not generally injective, but when the output length is large enough compared to the input then it will be. Our argument behind imposing this additional constraint on $H$ is that every $|\lfloor \frac{y}{2^i} \rfloor| \leq \ell$ maps to a unique encoding in $E$. Algorithm 4 receives as input $E \leftarrow \mathsf{IntEnc}(\mathcal{I})$ and $x \in \{0,1\}^\ell$ and outputs 1, if $x$ belongs to any of the sub-intervals in $\mathcal{I}$.

**Algorithm 3** $\mathsf{IntEnc}(\{[a_j, a_j+2^{p_j})\}_{j\in[k]})$

1: $E = \emptyset$
2: **for** $j = 1$ to $k$ **do**
3: Compute $\mu_j = f_{p_j}(a_j)$
4: Compute $h_j = H(\mu_j)$.
5: $E = E \cup \{h_j\}$
6: **end for**
7: **return** $E$

**Lemma 7 (Correctness).** *Let $c, c+w \in [0, 2^\ell)$. Consider algorithms* $\mathsf{GenInt}_{\mathcal{X}}$ *(Algorithm 1),* $\mathsf{GenInt}_{\mathcal{X}'}$ *(Algorithm 2),* $\mathsf{IntEnc}$ *(Algorithm 3),* $\mathsf{Dec}$ *(Algorithm*

**Algorithm 4** Dec (with embedded data $E$)

**Input:** $\ell \in \mathbb{N}$, $x \in \{0,1\}^\ell$
**Output:** 0 or 1.
1: **for** $i = 0$ to $\ell - 1$ **do**
2: Compute $H(f_i(x))$
3: **if** $H(f_i(x)) \in E$ **then**
4: **return** 1
5: **end if**
6: **end for**
7: **return** 0

*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* $H : \{0,1\}^* \to \{0,1\}^\omega$ *be injective when restricted to inputs of length at most* $\ell$ *(e.g.,* $\omega > 2\ell$*). Then* $x \in (c, c+w]$ *if and only if* Dec *outputs* 1.

*Proof.* Let $\mathcal{I} = \{[a_j, a_j + 2^{p_j})\}_{j\in[k]}$, then from Lemma 6 we have $(c, c+w] = \bigcup_{j=1}^{k}[a_j, a_j + 2^{p_j})$. Let $x$ be an integer in $(c, c+w]$, then it must belong to one of the sub-intervals in $\mathcal{I}$. Algorithm 3 computes $f_{p_j}(a_j) = \mu_j$, 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 $H(f_i(x)) \in E \leftarrow \mathsf{IntEnc}(\mathcal{I})$, and Dec outputs 1. If $x \notin (c, c+w]$ then $x$ does not lie in any of the intervals $[a_j, a_j + 2^{p_j})$. It follows that $f_i(x) \notin \{\mu_j : 1 \le j \le |\mathcal{I}|\}$ and therefore, since $H$ is injective, $E$ will not contain $H(f_i(x))$. Hence, Dec will correctly reject the input. □

We illustrate the interval encoding technique with the following concrete setting: consider the interval membership predicate $x \in (10, 14]$. To encode the interval, calculate $\mathcal{I}_\mathcal{X} \leftarrow \mathsf{GenInt}_\mathcal{X}(14)$ and $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(10)$ which gives the set of sub-intervals in $[0, 15)$ and $[15, 256)$ for $\ell = 8$. Finally, calculate $\mathsf{IntEnc}(\mathcal{I})$, where $\mathcal{I} \leftarrow \mathcal{I}_\mathcal{X} \cap \mathcal{I}_{\mathcal{X}'}$. The sets are indicated as follows:

$\mathcal{I}_\mathcal{X} = \{[0,8), [8,12), [12,14), [14,15)\}$

$\mathcal{I}_{\mathcal{X}'} = \{[11,12), [12,16), [16,32), [32,64), [64,128), [128,256)\}$

$\mathcal{I} = \{[11,12), [12,14), [14,15)\}$

$\mathsf{IntEnc}(\mathcal{I}) = \{H(f_0(00001011), H(f_1(00001100), H(f_0(00001110))\} = \{H(0000 1011), H(0000110), H(00001110)\}$

### 5.3 Obfuscator $\mathcal{O}$

We now present our decision tree obfuscator $\mathcal{O}$. Let $\mathcal{S} = (s_1, \ldots, s_{2^d})$ be the list of leaf nodes and $\mathsf{path}_{s_\tau}$ denotes an accepting path through the tree to leaf $s_\tau = 1$. Each accepting path is a conjunction of inequalities, and our objective is to obfuscate the conjunctions using our encoding technique for interval membership functions (see Section 5.2). Our construction relies on the fact that the terms in

a conjunction can be reordered. Recall that $x_i$ is compared at most twice along an accepting path, and hence the accepting path corresponds to $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$.

Precisely, the obfuscator works as follows: to encode $(c_i, c_i + w_i]$, calculate $\mathcal{I}^i_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c_i + w_i)$ and $\mathcal{I}^i_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(c_i)$, which gives the set of sub-intervals in $[0, c_i + w_i + 1)$ and $[c_i + 1, 2^\ell)$. Next, determine $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathcal{X}} \cap \mathcal{I}^i_{\mathcal{X}'}$ which is a set of sub-intervals whose union is $(c_i, c_i + w_i]$. To encode each sub-interval in $\mathcal{I}^i$, calculate encodings $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$. Let $H_c : \{0,1\}^* \to \{0,1\}^q$ be a cryptographic hash function with $q \geq 2\omega n$ (and hence injective on restricted inputs). The main idea is to concatenate each combination of $n$ hashes sorted in ascending order of $i$ for each entry in $\mathcal{B}^i$, and apply $H_c$ ; call the set of hashes $\mathcal{B}$. If $|\mathcal{B}| < 2^d(2\ell - 2)^n$, add dummy entries drawn uniformly at random from $\{0,1\}^q$. Finally, output $\mathcal{B}$. We give the formal details in Algorithm 5.

On input $(x_i)_{i\in[n]}$, the evaluation procedure calculates all possible encodings by evaluating $H(f_p(x_i))$ for every $p \in \{0, \ldots, \ell - 1\}$; call it $\mathcal{E}^i = \{h^i_1, \ldots, h^i_\sigma\}$. Finally compute all possible $H_c(\|_{i\in[n]} h^i_{q_i})$, where encodings are listed in ascending order of $i$. For an accepting input, one of the computed hash values belongs to the set $\mathcal{B}$ published by $\mathcal{O}$. Formally, the evaluation procedure is specified in Algorithm 6.

### 5.4 Correctness and Efficiency

We now analyze the correctness and efficiency of the obfuscator.

**Lemma 8 (Correctness.).** *Let $\lambda \in \mathbb{N}$ be the security parameter, and let $n$, $\ell$, $\omega$ and $q$ be polynomials in $\lambda$. Consider algorithms $\mathcal{O}$ (Algorithm 5) and* Eval *(Algorithm 6), and an input $(x_i)_{i\in[n]}$ with $x_i \in \{0,1\}^\ell$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ with $\omega > 2\ell$, and let $H_c : \{0,1\}^* \to \{0,1\}^q$ with $q > 2\omega n$. Then for every $[\![t_j, i, b]\!] \in \mathsf{path}_{s_\tau}$, where $t_j \in [0, 2^\ell)$, $i \in \{1, \ldots, n\}$, $b \in \{0,1\}$, for every $\mathcal{S} = (s_1, \ldots, s_{2^d})$ with $s_\tau \in \{0,1\}$, for every $\mathcal{B} \leftarrow \mathcal{O}$, and for every input $(x_i)_{i\in[n]}$, if $C((x_i)_{i\in[n]}) = 1$, then* Eval *outputs 1, else it outputs 0 with overwhelming probability in $\lambda$.*

*Proof.* Algorithm 5 calculates set of encodings $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$ for every $i \in [n]$. If there is some $i$ such that $[\![t_j, i, b]\!] \notin \mathsf{path}_{s_\tau}$ (which means $x_i$ is not compared in the path) then $\mathcal{B}^i = \{1^\ell\}$. Let $(x_i)_{i\in[n]}$ be an accepting input. From Definition 5, $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. From Lemma 7, if $x_i \in (c_i, c_i + w_i]$, then there exists a unique $h^i_k \in \mathcal{E}^i$, such that $h^i_k \in \mathcal{B}^i$. If $\exists\, i$, such that $[\![t_j, i, b]\!] \notin \mathsf{path}_{s_\tau}$, then $h^i_k = 1^\ell \in \mathcal{B}^i$. Thus, for an accepting input $(x_i)_{i\in[n]}$, there exists a unique $(h^1_{k_1}, \ldots, h^n_{k_n})$, where $h^i_{k_i} \in \mathcal{E}^i$ and $H_c(h^1_{k_1} \| \ldots \|^n_{k_n})$ will be contained in $\mathcal{B}$, and Algorithm 6 will correctly output 1.

If $C((x_i)_{i\in[n]}) \neq 1$, then by Lemma 7, hash values computed from $(x_i)$ will not match the hash values input to $H_c$. As we choose parameters such that $H_c$ is injective, $H_c(h^1_{k_1} \| \ldots \| h^n_{k_n})$ will not be contained in $\mathcal{B}$, except if it equals to one of its dummy entries. Since the number of possible encodings (in Eval) input to $H_c$ is $2^{(\ell+1)n}$, and $w > 2\ell$, $q > 2\omega n$, the probability that Eval incorrectly

**Algorithm 5** $\mathcal{O}\,(d, n, \ell \in \mathbb{N}, \mathsf{asset}(C))$

$\mathcal{B} = \emptyset$
$\alpha = 2^d(2\ell-2)^n$
**for all** $\tau$ such that $s_\tau = 1$ **do**
  Compute $\mathsf{path}_{s_\tau} = (\llbracket t_j, i, b \rrbracket : v_j$ is an ancestor of $s_\tau$, and $b = g_j(t_j))$
  **for** $i = 1$ to $n$ **do**
    $\mathcal{I}^i_{\mathcal{X}} = \mathcal{I}^i_{\mathcal{X}'} = [0, 2^\ell)$
    **if** $\llbracket t_{j_1}, i, 1 \rrbracket \in \mathsf{path}_{s_\tau}$ **then**
      $\mathcal{I}^i_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(t_{j_1})$
    **end if**
    **if** $\llbracket t_{j_2}, i, 0 \rrbracket \in \mathsf{path}_{s_\tau}$ **then**
      $\mathcal{I}^i_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(t_{j_2})$
    **end if**
    $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathcal{X}} \cap \mathcal{I}^i_{\mathcal{X}'}$
    **if** $(\mathcal{I}^i == [0, 2^\ell))$ **then**
      $\mathcal{B}^i = \{1^\ell\}$
    **else**
      $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$
    **end if**
  **end for**
  Denote $\mathcal{B}^i = (h^i_1, \ldots, h^i_{\sigma_i})$, where $\sigma_i \leq 2\ell - 2$ for each $i$.
  **for all** $((q_1, \ldots, q_n) \in [\sigma_1] \times \cdots \times [\sigma_n])$ **do**
    $\mathcal{B} = \mathcal{B} \cup \{H_c(\|_{i=1}^n h^i_{q_i})\}$
  **end for**
**end for**
**while** $(|\mathcal{B}| < \alpha)$ **do**
  $r \leftarrow\$ \{0,1\}^q$
  $\mathcal{B} = \mathcal{B} \cup \{r\}$
**end while**
**return** $\mathcal{B}$

**Algorithm 6** $\mathsf{Eval}$ ($\mathcal{B}$ with $|\mathcal{B}| = 2^d(2\ell-2)^n$)

**Input:** $(x_1, \ldots, x_n)$, $\ell$, $d$
**Output:** 0 or 1

**for** $i = 1$ to $n$ **do**
  $\mathcal{E}^i \leftarrow \{1^\ell\} \cup \{H(f_0(x_i)), \ldots, H(f_{\ell-1}(x_i))\}$
**end for**
$\mathcal{E}^i = \{h^i_1, \ldots, h^i_\sigma\}$, $\sigma \leq \ell + 1$
**for all** $(q_1, ..., q_i) \in [\sigma_1] \times \cdots \times [\sigma_n]$ **do**
  **if** $H_c(\|_{i\in[n]} h^i_{q_i}) \in \mathcal{B}$ **then**
    **return** 1
  **end if**
**end for**
**return** 0

accepts the input is given by $\frac{2^{(\ell+1)n}}{2^q} = \frac{1}{2^{3\ell n}}$. Finally, the probability that the $\alpha$ hash values output by $\mathcal{O}$ are good is given by $(1 - \frac{1}{2^{3\ell n}})^\alpha \approx 1 - negl(\lambda)$. □

**Complexity Analysis** We now discuss the size complexity of the obfuscated decision tree. Let $\lambda \in \mathbb{N}$, and $d, \ell, n, q$ be polynomials in $\lambda$.

Along an accepting path, the upper bound on the size of $\mathcal{B}^i$ is $2\ell - 2$ (see Lemma 6). There are $n$ such set of encodings, hence the total number of possible encodings input to $H_c$ is $(2\ell-2)^n$. Since, the number of accepting paths is $\mathrm{O}(2^d)$, the overall complexity of storing the obfuscated tree is $\mathrm{O}(2^{d+n} \cdot \ell^n \cdot q)$, where $q$ is the output size of $H_c$. Note that the upper bound $\alpha = 2^d(2\ell - 2)^n$ on the number of hashes is much larger than will be needed for most evasive decision trees, so in practice this parameter could be chosen a lot smaller to get a more compact obfuscated program.

Next, we analyze the time complexity of the evaluation procedure (Algorithm 6). Each node corresponds to set of encodings $\mathcal{E}^i$, where $|\mathcal{E}^i| = \ell+1$. For $n$ input attributes, the overall running time of the evaluation algorithm is of the order $\mathrm{O}(\ell^n \log(\alpha))$ operations. Since the query response time is exponential in $n$, we restrict to decision trees of constant number of input elements.

We now prove polynomial slowdown only for some special cases.

**Lemma 9 (Polynomial Slowdown).** *Let $\lambda \in \mathbb{N}$ be the security parameter and $\ell$, $n$, $d$ be polynomials in $\lambda$. Define $\mathcal{T}_\lambda$ as a special family of evasive decision trees, where $d = 5$, $n \leq 4$ and $\ell = \frac{\lambda}{4}$. Then for every $C \leftarrow \mathcal{T}_\lambda$, there exists a polynomial $p$ such that the running time of $\mathcal{O}(C)$ is bounded by $\mathrm{O}(p(\lambda))$.*

*Proof.* Let $C \leftarrow \mathcal{T}_\lambda$ determines whether an input $(x_i)_{i=1}^n$ is contained in the decision region defined by $(c_i, c_i + w_i]$ with $w_i \in (0, w_{max}]$. From Lemma 2, $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$, which specifies the maximum width of the intervals. For evasiveness, we require $\ell - \frac{\lambda}{n} \geq 0$, which gives $\ell n \geq \lambda$. Taking $\ell = \frac{\lambda}{4}$ 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 $\mathcal{O}$ is given by $\ell^n = (\frac{\lambda}{4})^4$. □

**Parameters for Secure Construction.** In the previous sections, we have discussed the choice of parameters that provide security to our decision tree obfuscator, i.e. for $\ell = \ell(\lambda)$, $d = d(\lambda)$, $n = n(\lambda)$, the conditions for evasiveness are given in Lemma 2. For the hardness of finding $x_i$ that belongs to the decision region $(c_i, c_i + w_i]$, we require $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$ for every $i \in [n]$. In addition to that, we impose $d \leq n + 1$ (by construction). We now present some example parameters along with their bit-security in Table 1.

## 6 Proof of VBB Security

We prove VBB security in the random oracle model. For simplicity we restrict to decision trees with one accepting path, and let $\alpha = (2\ell - 2)^n$ be an upper bound on the number of hash values required for the obfuscated program.

| $d$ | $\ell$ | $n$ | $\lambda$ | Program size | Evaluation cost |
|---|---|---|---|---|---|
| 5 | 64 | 4 | 128 | $126^4 \times 512$ | $65^4 \times 512$ |
| 3 | 64 | 2 | 64 | $126^2 \times 512$ | $65^2 \times 512$ |

**Table 1.** Example parameter sets for an obfuscated decision tree with $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$. For $q = 512$ bits and *one* accepting path ($q$ is the output size of hash function $H_c$), we calculate the size of the obfuscated program and the cost of the evaluation (algorithms 5 and 6).

We use cryptographic hash functions $H : \{0,1\}^* \to \{0,1\}^\omega$ and $H_c : \{0,1\}^* \to \{0,1\}^q$ in our construction, which we model as random oracles in our security proof. 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 executing the adversary in a simulation, such that the adversary cannot distinguish between the simulated and real environment. We first give a brief intuition to our security proof.

Consider a simulator $\mathcal{S}$ who samples parameters $\ell, n$ following the conditions in Lemma 9, and sends them to adversary $\mathcal{A}$. $\mathcal{A}$ now samples $C$ and provides $\mathcal{S}$ oracle access to $C$. Since $\mathcal{S}$ does not know the program $C$, it simulates the obfuscated program and random oracles and provides answers to $\mathcal{A}$'s queries.

If the adversary never queries the circuit with an accepting input, everything is a correct simulation. However, if the adversary does query the circuit with an accepting input, then the security reduction immediately uses this clue to mount a model extraction attack, and hence learn the corresponding accepting path in the decision tree. The security reduction can then run the obfuscator correctly for that single accepting path, and program the random oracles to be consistent with the simulated $\mathcal{O}$ (the reduction does not learn anything about the other possible accepting paths). Hence, again everything is a correct simulation.

**Theorem 1.** *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\ell = \ell(\lambda)$ and $\alpha = \alpha(\lambda)$ satisfy the conditions required for Lemma 9. Let $\mathcal{D}_\lambda$ be a distribution of evasive decision trees. Then for random oracles $H : \{0,1\}^* \to \{0,1\}^\omega$ and $H_c : \{0,1\}^* \to \{0,1\}^q$, the decision tree obfuscator $\mathcal{O}$ is a VBB obfuscator.*

*Proof.* As evident from Lemma 8, $\mathcal{O}$ satisfies functionality preservation. Lemma 9 shows that $\mathcal{O}$ 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 $\mathcal{D}_\lambda$ (from Lemma 9), the following holds:

$$\left| \Pr_{C \leftarrow D_\lambda} [\mathcal{A}(\mathcal{O}(1^\lambda, C)) = 1] - \Pr_{C \leftarrow D_\lambda} [\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$

Every $C \leftarrow \mathcal{D}_\lambda$ identifies unique $(c_1, \ldots, c_n) \leftarrow X_n$ and $(w_1, \ldots, w_n) \leftarrow (0, w_{max}]^n$, and on input $(x_1, \ldots, x_n)$ determines 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 correct obfuscation of $C$. Let

$\mathcal{A}$ be a PPT adversary that takes as input $\mathcal{O}(1^\lambda, C)$. 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: $\mathcal{S}$ samples $\alpha$ values uniformly at random from the co-domain of $H_c$, and sends $\{h'_1, \ldots, h'_\alpha\}$ to $\mathcal{A}$.

We assume that $\mathcal{A}$ makes polynomially many queries to both the random oracles. When $\mathcal{A}$ queries oracle $H$ with $\mathbf{u}^*$, $\mathcal{S}$ looks for $v$ such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$ and returns it to the adversary. If no such $v$ exists, then the simulation samples a distinct $v \in \{0,1\}^\omega$ uniformly at random, adds $(\mathbf{u}^*, v)$ to $\mathcal{T}_1$, and returns $v$ to $\mathcal{A}$.

When $\mathcal{A}$ makes a query $\mathbf{h}^*$ to the random oracle $H_c$, the simulator looks 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 $(h_1^*, \ldots, h_n^*)$ and looks up table $\mathcal{T}_1$ to find an entry corresponding to each parsed string. If there does not exist such an entry in $\mathcal{T}_1$, then a distinct $val \in \{0,1\}^q$ is chosen uniformly at random, $(\mathbf{h}^*, val)$ is added to $\mathcal{T}_2$, 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 $\mu_j$ (for a correct input), adding $j$ zeroes yields an accepting input for $C$. Eventually, the simulator queries the oracle $C$ with the $(x_i)_{i\in[n]}$. If $C$ returns 1, $\mathcal{S}$ determines the $c_i$'s and $w_i$'s by doing standard model extraction attack for that *single* accepting path, calculates pairs $(u, v)$ and registers the entries in $\mathcal{T}_1$.

Note that we do not learn anything about other possible accepting paths. Thereafter the simulator calculates the $\alpha$ input entries of $\mathcal{T}_2$, defines them to be $\alpha$ entries from the already published 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.

We describe the simulation in form of pseudo code in Algorithm 7 and 8.

**Algorithm 7** Oracle $H(\mathbf{u}^*)$

```
Find all v such that (u*, v) ∈ T1
if no such v exists then
    v ←$ {0,1}^ω
    T1 ← T1 ∪ (u*, v)
else
    if ∃ w (u* ≠ w), such that (u*, v) ∈ T1 and (w, v) ∈ T1 then
        return ⊥
    end if
end if
return v
```

**Algorithm 8** Oracle $H_c(\mathbf{h}^*)$

Find all $val$ such that $(\mathbf{h}^*, val) \in \mathcal{T}_2$
**if** no such $val$ exists **then**
  Parse $\mathbf{h}^* = (h_i^*)_{i\in[n]}$
  $\mathsf{counter} = 0$
  **for** $i = 1$ to $n$ **do**
    **if** $(u, h_i^*) \in \mathcal{T}_1$ **then**
      $\mathsf{counter} \leftarrow \mathsf{counter} + +$
      $j \leftarrow \ell - |u|$
      $x_i \leftarrow u \times 2^j$
    **end if**
  **end for**
  **if** $(\mathsf{counter} == n)$ **then**
    $b \leftarrow \mathcal{S}^C(x_1, \ldots, x_n)$
    **if** $(b == 1)$ **then**
      Calculate $(c_i, w_i)_{i\in[n]}$ using model-extraction attack
      Run Algorithm 5 and calculate $(u, v)$, $h$
      $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup (u, v)$
      **if** $\exists\, u_1, u_2\ (u_1 \neq u_2)$, such that $(u_1, v) \in \mathcal{T}_1$ and $(u_2, v) \in \mathcal{T}_1$ **then**
        **return** $\perp$
      **else**
        **for** $i = 1$ to $\alpha$ **do**
          $val_i \leftarrow h'_i$
          $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (h_i, val_i)$
        **end for**
      **end if**
    **end if**
    $val \leftarrow\$ \{0,1\}^q$
    $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (\mathbf{h}^*, val)$
  **end if**
**end if**
**return** $val$

The simulated view is distributed identically to the real view. Hence $\mathcal{A}$ cannot distinguish between the real and simulated obfuscation instances. □

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 $\lambda$.

**Lemma 10.** *Let $\lambda \in \mathbb{N}$ be the security parameter and let $\ell$, $n$, $\alpha$ be polynomials in $\lambda$. Let $\mathcal{D}_\lambda$ be an ensemble of evasive decision tree distributions, and let $\mathcal{O}(1^\lambda, C)$ be the obfuscation of $C \leftarrow \mathcal{D}_\lambda$. Consider Algorithms 7 and 8 and random oracles $H : \{0,1\}^* \to \{0,1\}^\omega$ 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*

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

*where $\mathcal{S}$ is a PPT algorithm with oracle access to $C$.*

*Proof.* The simulation fails if and only if there is a conflict in table $\mathcal{T}_1$, and $\mathcal{S}$ halts. Conflicts may arise when $\mathcal{S}$ has already responded to an oracle query $\mathbf{u}^*$ to $H$ with $v \leftarrow\$ \{0,1\}^\omega$, and later on query $\mathbf{h}^*$ to $H_c$, it queries the oracle of $C$, and populates $\mathcal{T}_1$ with $(w, v)$ such that $w \neq \mathbf{u}^*$. Let $\eta = \eta(\lambda)$ be the number of entries in $\mathcal{T}_1$. 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 $\eta$ values in table $\mathcal{T}_1$. Since $H : \{0,1\}^* \rightarrow \{0,1\}^\omega$, there are $2^\omega$ choices for the 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 $\frac{1}{2^\omega}$. Continuing the same way, when there are $\eta - 1$ entries in $\mathcal{T}_1$, the probability of collision is $\frac{(\eta-1)}{2^\omega}$. Assuming all the samples are independent, the final step is to draw a conclusion about $\mathcal{S}$'s probability of failure, given by

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

□

## 7 Conclusion

In this paper, we have introduced a new special-purpose virtual black-box obfuscator for evasive decision trees. While doing so, we have presented an encoder for hiding parameters in an interval-membership function. Our obfuscation construction blows up exponentially in the depth of the tree, hence an interesting problem would be to investigate solutions that work for more general class of evasive decision trees.

## 8 Acknowledgements

We thank Phillip Rogaway for discussions on methods for obfuscating inequalities. We thank the Marsden Fund of the Royal Society of New Zealand for supporting this research. We thank the reviewers of INDOCRYPT 2023 for their insightful comments.

## References

1. Ateniese, G., Mancini, L.V., Spognardi, A., Villani, A., Vitali, D., Felici, G.: 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)

2. Banerjee, S., Galbraith, S.D., Khan, T., Castellanos, J.H., Russello, G.: Preventing reverse engineering of control programs in industrial control systems. In: Proceedings of the 9th ACM Cyber-Physical System Security Workshop. pp. 48–59 (2023)
3. Barak, B., Bitansky, N., Canetti, R., Kalai, Y.T., Paneth, O., Sahai, A.: Obfuscation for evasive functions. In: Theory of Cryptography Conference. pp. 26–51. Springer (2014)
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. In: Annual international cryptology conference. pp. 1–18. Springer (2001)
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. Barni, M., Failla, P., Lazzeretti, R., Paus, A., Sadeghi, A.R., Schneider, T., Kolesnikov, V.: Efficient privacy-preserving classification of ecg signals. In: 2009 First IEEE International Workshop on Information Forensics and Security (WIFS). pp. 91–95. IEEE (2009)
7. Bartusek, J., Lepoint, T., Ma, F., Zhandry, M.: New techniques for obfuscating conjunctions. In: EUROCRYPT. pp. 636–666. Springer (2019)
8. Bishop, A., Kowalczyk, L., Malkin, T., Pastro, V., Raykova, M., Shi, K.: A simple obfuscation scheme for pattern-matching with wildcards. In: Annual International Cryptology Conference. pp. 731–752. Springer (2018)
9. Blurock, E.S.: Automatic learning of chemical concepts: Research octane number and molecular substructures. Computers & chemistry **19**(2), 91–99 (1995)
10. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Journal of biomedical informatics **50**, 234–243 (2014)
11. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. Cryptology ePrint Archive (2014)
12. Boyle, E., Ishai, Y., Meyer, P., Robere, R., Yehuda, G.: On low-end obfuscation and learning. In: 14th Innovations in Theoretical Computer Science Conference (ITCS 2023). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2023)
13. 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 (2007)
14. Canetti, R.: Towards realizing random oracles: Hash functions that hide all partial information. In: Annual International Cryptology Conference. pp. 455–469. Springer (1997)
15. Canetti, R., Rothblum, G.N., Varia, M.: Obfuscation of hyperplane membership. In: Theory of Cryptography Conference. pp. 72–89. Springer (2010)
16. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 563–577 (2022)
17. Decaestecker, C., Remmelink, M., Salmon, I., Camby, I., Goldschmidt, D., Petein, M., Van Ham, P., Pasteels, J.L., Kiss, R.: 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)
18. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1322–1333 (2015)

19. Galbraith, S.D., Zobernig, L.: Obfuscated fuzzy hamming distance and conjunctions from subset product problems. In: Theory of Cryptography Conference. pp. 81–110. Springer (2019)
20. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing **45**(3), 882–929 (2016)
21. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). pp. 612–621. IEEE (2017)
22. Karnouskos, S.: Stuxnet worm impact on industrial cyber-physical system security. In: IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society. pp. 4490–4494. IEEE (2011)
23. Kesarwani, M., Mukhoty, B., Arya, V., Mehta, S.: Model extraction warning in mlaas paradigm. In: Proceedings of the 34th Annual Computer Security Applications Conference. pp. 371–380 (2018)
24. Lee, T., Edwards, B., Molloy, I., Su, D.: Defending against model stealing attacks using deceptive perturbations. arXiv preprint arXiv:1806.00054 (2018)
25. Ma, J.P., Tai, R.K., Zhao, Y., Chow, S.S.: Privacy-preserving decision trees evaluation via linear functions. In: Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22. pp. 494–512. Springer (2017)
26. Pal, S., Gupta, Y., Shukla, A., Kanade, A., Shevade, S., Ganapathy, V.: A framework for the extraction of deep neural networks by leveraging public data. arXiv preprint arXiv:1905.09165 (2019)
27. Quiring, E., Arp, D., Rieck, K.: Forgotten siblings: Unifying attacks on machine learning and digital watermarking. In: 2018 IEEE European symposium on security and privacy (EuroS&P). pp. 488–502. IEEE (2018)
28. Silverstein, C., Shieber, S.M.: Predicting individual book use for off-site storage using decision trees. The Library Quarterly **66**(3), 266–293 (1996)
29. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: 25th USENIX security symposium (USENIX Security 16). pp. 601–618 (2016)
30. Wee, H.: On obfuscating point functions. In: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing. pp. 523–532 (2005)
31. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). pp. 600–611. IEEE (2017)
32. Zheng, H., Ye, Q., Hu, H., Fang, C., Shi, J.: BDPL: A boundary differentially private layer against machine learning model extraction attacks. In: European Symposium on Research in Computer Security. pp. 66–83. Springer (2019)