Samuel Adams, Chaitali Choudhary, Martine De Cock*, Rafael Dowsley*, David Melanson, Anderson Nascimento*, Davis Railsback, and Jianwei Shen

Privacy-preserving training of tree ensembles over continuous data

Abstract: Most existing Secure Multi-Party Computation (MPC) protocols for privacy-preserving training of decision trees over distributed data assume that the features are categorical. In real-life applications, features are often numerical. The standard "in the clear" algorithm to grow decision trees on data with continuous values requires sorting of training examples for each feature in the quest for an optimal cut-point in the range of feature values in each node. Sorting is an expensive operation in MPC, hence finding secure protocols that avoid such an expensive step is a relevant problem in privacy-preserving machine learning. In this paper we propose three more efficient alternatives for secure training of decision tree based models on data with continuous features, namely: (1) secure discretization of the data, followed by secure training of a decision tree over the discretized data; (2) secure discretization of the data, followed by secure training of a random forest over the discretized data; and (3) secure training of extremely randomized trees ("extra-trees") on the original data. Approaches (2) and (3) both involve randomizing feature choices. In addition, in approach (3) cut-points are chosen randomly as well, thereby alleviating the need to sort or to discretize the data up front. We implemented all proposed solutions in the semi-honest setting with additive secret sharing based MPC. In addition to mathematically proving that all proposed approaches are correct and secure, we experimentally evaluated and compared them in terms of classification accuracy and runtime. We privately train tree ensembles over data sets with thousands of instances or features in a few minutes, with accuracies that are at par with those obtained in the clear. This makes our solution more efficient than the existing approaches, which are based on oblivious sorting.

Keywords: Machine Learning, Privacy, Secure Multi-Party Computation, Decision Tree Ensembles, Random Forest, Training

DOI 10.2478/popets-2022-0042

Received 2021-08-31; revised 2021-12-15; accepted 2021-12-16.

1 Introduction

Secure Multi-Party Computation (MPC) is a powerful tool for achieving Privacy-Preserving Machine Learning (PPML). For example, instantiating PPML based on Secure Multi-Party Computation [1] enables multiple parties to work together to train an ML model over their combined data, without any of the parties learning anything about each other's data.

Recent advances in MPC-based protocols for *training* of ML models over distributed data are primarily focused on secure training of neural network architectures [2–5]. While deep learning is state-of-the-art for tasks that relate to perception, such as computer vision and natural language processing, in domains with structured information, the best results are often obtained with tree ensemble methods, such as random forests and boosted decision trees [6]. The latter also have the advantages of being faster to train and being easier to interpret. Advances on MPC-based training of tree based classifiers are fairly limited. While there is work on *secure inference* with pre-trained decision trees and tree ensembles [7–9], work on *secure training* itself is limited to the training of individual decision trees (DTs). Several authors have proposed a secure version of Quinlan's ID3 algorithm [10] for training DTs with *categorical* features [11–14]. Existing proposals for training DTs

Samuel Adams, Chaitali Choudhary, David Melanson, Davis Railsback: University of Washington Tacoma, E-mail: {sdadams, cc201, mence40, drail}@uw.edu
***Corresponding Author: Martine De Cock:** University of Washington Tacoma, E-mail: mdecock@uw.edu; Ghent University, E-mail: martine.decock@ugent.be
***Corresponding Author: Rafael Dowsley:** Monash University, E-mail: rafael.dowsley@monash.edu
***Corresponding Author: Anderson Nascimento:** University of Washington Tacoma, E-mail: andclay@uw.edu
Jianwei Shen: University of Arizona, E-mail: sjwjames@email.arizona.edu

with *continuous* features [15–18] are based on Quinlan's C4.5 algorithm [19], an algorithm that involves sorting, a time-consuming operation in the MPC setting.

Secure DT learning with MPC is challenging for a variety of reasons. For algorithms in general to be secure in MPC, measures must be taken to ensure that the number of executions of instructions is not dependent on specific values of the input, because that in itself could leak information. In the context of ML algorithms, where models are trained privately and not revealed to the parties, this means for instance that one should not rely on early stopping conditions, or on control flow logic. Furthermore, for efficiency considerations, dependency on previous multiplication results should be minimized, and operations like division, analytic function evaluation, and integer logic should be avoided where possible. All of these Achilles' heels of MPC are inherent requirements of traditional DT training algorithms. Indeed, trees are grown recursively, which implies several layers of dependency on previous results. Furthermore, the "decision" component of a tree indicates a stopping condition has been met (thereby potentially revealing information, if one is not careful), and the information gain metric for greedy selection of splitting features requires computing division and analytic functions. In this paper, we introduce novel methods to contend with these requirements.

We propose three alternative strategies for secure training of tree based models over data with continuous feature values, none of which requires sorting of feature values. Two of our approaches rely on privacy-preserving discretization of the range of feature values. After this step, any of the existing algorithms for secure training of a DT over categorical data can be used. In our case, we use the SID3T training algorithm proposed by de Hoogh et al. [14]. This constitutes our first approach.

Next, we present a novel protocol for secure training of a random forest (RF) over data with categorical values, by extending de Hoogh et al.'s secure DT training algorithm [14] with a protocol for privacy-preserving random feature selection. Combined with the secure discretization protocol from approach 1, this allows us to securely train RFs over data with continuous values, constituting our second approach.

When growing a tree for data with continuous features (e.g. *price*), the standard "in the clear" C4.5 algorithm sorts the feature values to look for a cut-off point (e.g. $\leq$ 100\$) that will reduce class label impurity the most at the next level of the tree. To circumvent expensive secure sorting operations, as our third approach, we propose secure training of tree ensembles based on randomized choices for the cut-off points. In the clear, this idea, combined with randomized feature selection, is known as *extremely randomized trees*. Such "extra-trees classifiers" achieve state-of-the-art accuracy and are fast to train over data sets with numerical features [20]. Summarizing, in this paper:

- We propose an MPC-based protocol π_{DISC} for privacy-preserving discretization of a range of continuous feature values, in scenarios where the feature values are distributed across different parties.
- We present the first MPC-based protocol π_{RF} for training of a random forest (RF).
- We propose the first MPC-based protocol π_{XT} for training of an extra-trees classifier (XT).
- As a side result, we propose several improvements and optimizations to important building blocks of privacy-preserving machine learning protocols such as secure comparisons.

Combined, these protocols allow to train tree based models over distributed data with continuous values in a variety of ways: (1) secure discretization of the data with π_{DISC}, followed by secure training of a DT over the discretized data with the π_{SID3T} protocol from de Hoogh et al. [14]; (2) secure discretization of the data with π_{DISC}, followed by secure training of a random forest over the discretized data with π_{RF}; and (3) secure training of extremely randomized trees on the original data with π_{XT}. All our approaches accommodate scenarios in which the trained ML models have to remain private, i.e. secret shared across the parties, as well as scenarios in which the trained ML models are disclosed at the end. Furthermore, all our approaches work in scenarios where the data is horizontally partitioned (each party has some of the rows or instances), scenarios where the data is vertically partitioned (each party has some of the columns or features), and even in scenarios where each computing party only has secret shares of the data to begin with.

We compare the accuracy and the runtime of the proposed approaches on publicly available benchmark data sets with thousands of instances or features. Our solutions are simple, and, for the most part, use building blocks that already exist in the literature. Importantly, our solutions work. We obtain accuracies that are at par with those that can be obtained with existing sorting based MPC protocols for training of tree based models, while being much faster. Our results show that a full sort is not necessary for MPC-based training of tree based models.

Related works. The majority of the existing work on the training of DTs is for categorical data only [11–14]. In a preliminary attempt to extend MPC-based privacy-preserving DT training algorithms to continuous variables, Xiao et al. [15] proposed a straightforward adaptation of the C4.5 algorithm with a privacy-preserving bubble sort algorithm, which is not practical. Moreover, the authors did not give security proofs for their proposed protocols. Shen et al. [16] extended the result presented in [15] to vertically partitioned data.

More recently, Abspoel et al. [18] proposed a new MPC-based adaptation of the C4.5 algorithm, using sorting networks to obliviously presort the feature values. Subsequently they consider each feature value as a candidate cut-off point, and compute the Gini index for each. They estimate that training one DT of depth 4 on a data set with 8192 instances and 11 features would take slightly over 8 min when run on three m5d.2xlarge EC2 instances connected via a LAN. They extrapolate from this that training an ensemble of 200 such trees, each over a sample of 8192 instances and 11 features drawn from a much larger data set of training instances and features, could be done in less than 28 hours. Their solution does not include a mechanism for performing bagging (selection of the instances used in each tree) and subspace sampling (selection of the features used in each tree) in a privacy-preserving manner. In this paper, we propose MPC protocols to this end in what is, to the best of our knowledge, the first end-to-end protocol for MPC-based training of random forests. We also present a working implementation that allows to train accurate tree ensembles over data sets with thousands of instances or features in a matter of minutes, i.e. our solution is much faster than the existing ones. These improvements in efficiency stem from the fact that we do not privately sort feature values, and from the fact that we need to perform far less Gini index computations, because we perform a much more straightforward discretization of the data, while maintaining high accuracy.

Our work on MPC protocols for privacy-preserving training of random forests (RFs) and extra-trees classifiers (XTs) was carried out independently from simultaneous work on MPC-based training of gradient boosted decision tree models (XGBoost) [21]. RFs and XTs are inherently different from XGBoost. Whereas for XGBoost an ensemble of trees is trained in sequence by adding, at each step, the tree with the greatest accuracy improvement, for RFs and XTs, many trees are trained independently on different random subsamples of the data. RFs, XTs, and XGBoost are all popular tree ensemble methods in data science that may outperform one another in predictive accuracy depending on the data set and the task at hand.

2 Preliminaries

Security setting. We consider *honest-but-curious, static adversaries*, as is common in MPC-based PPML (see e.g. [14, 22]). An honest-but-curious adversary (also known as passive or semi-honest adversary) follows the instructions of the protocol, but tries to gather additional information. Secure protocols prevent the latter. A static adversary chooses the parties that he wants to corrupt before the protocol execution. Our security model and proofs are in Appendix A.

Fixed point representation. The protocols proposed in this paper are designed for training of DT based models on data with continuous feature values. The training data consists of a set S of training examples $\langle (x_1, x_2, \ldots, x_f), y \rangle$. The feature values $x_1, x_2, \ldots, x_f$ are real numbers, while the class label y is categorical. The goal is to learn a function from the data that maps previously unseen feature values to a corresponding class label, e.g. to determine whether a patient has a disease or not based on blood pressure, temperature etc. The kind of functions that we consider in this paper are DT based models. Our assumption is that, instead of residing in one place, the data set S is distributed across multiple data owners.

During the execution of the protocols however, operations are performed on additive shares in a ring $\mathbb{Z}_q$, for some appropriately chosen integer q. In this work we mostly use $q = 2^\lambda$. The feature values x in $\mathbb{R}$ first need to be converted into values $Q(x)$ in $\mathbb{Z}_{2^\lambda}$ by the data owners. To this end we use a fixed point representation with two's complement for negative numbers:

$$Q(x) = \begin{cases} 2^\lambda - \lfloor 2^a \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^a \cdot x \rfloor & \text{if } x \geq 0 \end{cases} \tag{1}$$

When converting $Q(x)$ into its bit representation, it consists of λ bits in total. The first a bits from the right hold the fractional part of x, the next b bits represent the non-negative integer part of x, and the most significant bit (MSB) represents the sign (positive or negative). It is important to choose λ large enough to be able to represent the largest numbers produced during the protocols. When multiplying fixed point numbers, the number of fractional bits doubles and must be truncated to remain

in the proper range. Therefore, λ should be chosen to be at least $2(a+b)$. It is also important to choose b large enough to represent the maximum possible value of the integer part of all x's.

After the conversion, each data owner secret shares the feature values on its end. In general, a number z in $\mathbb{Z}_q$ is split in m shares by picking $z_1, z_2, \ldots, z_m \in \mathbb{Z}_q$ uniformly at random subject to the constraint that $z = \sum_i z_i \mod q$. All computations are modulo q and the modular notation is henceforth omitted for conciseness. We denote this secret sharing by $[\![z]\!]_q$, which can be thought of as a shorthand for $(z_1, z_2, \ldots, z_m)$. For the case of $q = 2^\lambda$, we simplify the notation to $[\![z]\!]$. Note that no information about the secret value z is revealed by any proper subset of the m shares, but the secret shared value can be trivially revealed by combining all shares. The class label of training examples is secret shared in the same manner as the feature values.

The Trusted initializer setting and adversarial model. It is well-known that unconditionally secure computations are impossible in a two-party setting unless additional (computational and/or setup) assumptions are in place. We work in the commodity-based cryptographic model, where a trusted initializer (TI) pre-distributes correlated randomness to the parties participating in the protocol. For ease of explanation we will describe our protocols assuming $m = 2$ computing parties (plus the TI). We denote these computing parties by *Alice* and *Bob*. In particular, we make extensive use of pre-distributed multiplication triples. This technique was originally proposed by Beaver [23] and is regularly used to enable very efficient solutions in the context of PPML (see e.g. [2, 5, 24, 25]). The TI additionally generates random values in $\mathbb{Z}_q$ and delivers them to Alice so that she can use them to secret share her inputs. If Alice wants to secret share an input x, she picks an unused random value r (note that Bob does not know r), and sends $c = x - r$ to Bob. Her share x_A of x is then set to $x_A = r$, while Bob's share x_B is set to $x_B = c$. The secret sharing of Bob's inputs is done similarly using random values that the TI only delivers to him. After the setup phase, the TI is not involved in any other part of the execution and does not learn any data from the parties. In case a TI is not available or desirable, Alice and Bob can simulate the role of the TI, at the cost of additional pre-processing time and computational assumptions, see [2]. We prove our protocols secure in the universal composability framework considering honest-but-curious adversaries [26], see Appendix A.

Operations on secret shared values. Given secret shared values $[\![x]\!]_q$ and $[\![y]\!]_q$, and a constant c, Alice and Bob can trivially perform the following operations locally:

- Addition ($z = x + y$): Alice and Bob just add their local shares of x and y. This operation will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + [\![y]\!]_q$.
- Subtraction ($z = x - y$): Alice and Bob subtract their local shares of y from that of x. This operation will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q - [\![y]\!]_q$.
- Multiplication by a constant ($z = cx$): Alice and Bob multiply their local shares of x by c. This operation will be denoted by $[\![z]\!]_q \leftarrow c[\![x]\!]_q$.
- Addition of a constant ($z = x + c$): Alice adds c to her share x, while Bob keeps the same share of x. This operation will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$.

We use the same protocol π_{DMM} for secure (matrix) multiplication of secret shared values as in [22] and denote by π_{DM} the protocol for the special case of multiplication of scalars. The notation $[\![Z]\!]_q \leftarrow [\![X]\!]_q \cdot [\![Y]\!]_q$ is used to denote the multiplication of two secret shared matrices X and Y, and the notation $[\![v]\!]_q * [\![w]\!]_q$ is used to denote the element wise product of two secret shared vectors v and w.

When working with fixed-point representations over $\mathbb{Z}_q$ with a fractional bits, every multiplication generates an extra a bits of unwanted fractional representation. Having a secure way to "chop off" the extra fractional bits generated by multiplication is a requirement to efficiently work with fixed-point secret shares. In the two-party scenario with shares in $\mathbb{Z}_{2^\lambda}$, it is possible to perform this truncation with a local probabilistic protocol – hereafter, π_{trunc}– that with overwhelming probability in the security parameter $\lambda - (a + b)$ introduces an error of at most 1 in the least significant bit [2]. Since all operations to compute π_{trunc} are local, the performance overhead of truncating all multiplication results with this method is essentially zero. We refer to [2] for a detailed description of π_{trunc}.

Often in MPC, there are problems that are best solved with integer arithmetic performed over $\mathbb{Z}_q$ (for a large q), while others (such as secure comparisons) are best solved over $\mathbb{Z}_2$. We work using a combination of these two techniques. Thus, it is necessary to be able to convert secret shares from one modulus to the other. For example, to determine if $[\![a]\!]_q$ is equal to $[\![b]\!]_q$, one would first convert $[\![a]\!]_q$ and $[\![b]\!]_q$ into bitwise sharings over $\mathbb{Z}_2$ and then confirm that each bit is identical using binary operations. The result, itself a bit shared over

$\mathbb{Z}_2$, needs to be converted back to a sharing over $\mathbb{Z}_q$ for use in subsequent computations with integers. A quite efficient protocol for converting from $\mathbb{Z}_{2^\lambda}$ shares to bitwise sharings over $\mathbb{Z}_2$, denoted π_{decomp}, can be found in [4]. π_{decomp} is not fully described in this paper because novel methods for computing control flow logic are developed in the following section that do not require a full decomposition into $\mathbb{Z}_2$ shares.

Regarding the opposite direction, the conversion from a bit shared over $\mathbb{Z}_2$ to a sharing of 0 or 1 over $\mathbb{Z}_q$, hereafter π_{2toQ} [27], is still necessary for our purposes. The intuition for π_{2toQ} is that for a bit $[\![b]\!]_2$ shared over $\mathbb{Z}_2$ between two parties, there are four possible secret shares of which three are valid $\mathbb{Z}_q$ sharings and one is not. If $[\![b]\!]_2 = b_0 + b_1 \mod 2 = 1$, then $(1,0)$ and $(0,1)$ are the only possible sharings. Similarly, if $[\![b]\!]_2 = b_0 + b_1 \mod 2 = 0$, then $(0,0)$ and $(1,1)$ are the only possible sharings. In the case of secret shares $(0,0)$, $(0,1)$, and $(1,0)$, it holds automatically that $b_0 + b_1 \mod 2 = b_0 + b_1 \mod q$ for all $q > 2$. However, the problematic sharing $(1,1)$ – which encodes 0 – sums to 2 when regarded as a $\mathbb{Z}_q$ sharing. Hence, the problem of converting from $\mathbb{Z}_2$ to $\mathbb{Z}_q$ is reduced to mapping a secret sharing of 2 to a secret sharing of 0 as described in Protocol 1. Each invocation of π_{2toQ} requires one secure multiplication over $\mathbb{Z}_q$, $2\lceil \log(q) \rceil$ bits of data transfer, and one round of communication.

Protocol 1: Secure protocol π_{2toQ} converts a secret bit from a $\mathbb{Z}_2$ sharing to a $\mathbb{Z}_q$ sharing.

Input : $[\![b]\!]_2 := (b_0,\ b_1)$
Output: $[\![b]\!]_q$
1 Alice creates the sharing $[\![b_0]\!]_q = (b_0, 0)$
2 Bob creates the sharing $[\![b_1]\!]_q = (0, b_1)$
3 $[\![b]\!]_q \leftarrow [\![b_0]\!]_q + [\![b_1]\!]_q - 2 \cdot [\![b_0]\!]_q \cdot [\![b_1]\!]_q$
4 **return** $[\![b]\!]_q$

Protocol for secure DT training. Each internal node in a DT tests the value of a particular feature and branches out accordingly, while each leaf node contains a class label. In the clear, training of a DT is done by growing the tree from the root to the leaf nodes in a recursive manner. For each internal node, the feature is selected that splits the set of training instances that have reached that node in subsets that are as homogeneous as possible regarding the class label value. MPC protocols for secure training of DTs commonly use the Gini impurity to this end (the lower the impurity, the higher the homogeneity). As a sub-protocol for performing the secure training of a DT with categorical data we use protocol π_{SID3T} that is a slightly modified version of the protocol SID3T of De Hoogh et al. [14]. The differences are the following:

- De Hoogh et al. [14] used secret shares in a prime field because their multiplication protocol required the existence of modular inverses. We instead perform multiplications using multiplication triples and can work with shares in $\mathbb{Z}_{2^\lambda}$.
- For stopping criteria, De Hoogh et al. [14] used: (1) no features remain in the training set in the node at hand, (2) all remaining instances in the node have the same class label, and (3) the number of remaining instances in the node is less than a cutoff threshold. We use only (2) and (3), and additionally work with a pre-specified maximum allowed tree depth to prevent overfitting.
- Their solution grows the DT recursively and leaks the shape of the tree. Our version grows DT's iteratively (by depth, up to a pre-specified maximum depth) and adds dummy nodes to create full trees of the pre-specified, limited depth. By adding dummy nodes, we hide true path lengths and only leak the depth of the trees. We call the last non-dummy node on each path a *classifying node*. We keep track of such nodes by cascading a secret shared bit representing whether or not an early stopping condition was already reached on the path during training.
- For each leaf node, De Hoogh et al. [14]'s protocol returns a secret-shared one-hot-encoded vector denoting the class label. In contrast, for each classifying node, our algorithm returns secret shared frequencies of each of the class labels in the subset of training instances that have reached that node. Such secret shared frequency values are inexpensive to compute because additions can be done locally by the computing parties, and the frequencies allow for weighted aggregation of class label votes of trees in an ensemble, leading to more accurate classifications.

The DT is output as an array of secret shared one-hot-encodings of the split feature at each node, in addition to the value(s) to be compared against.

3 Secure comparison protocol

Secure comparison of integers is a well-studied problem in this domain. Specifically, many solutions exist to compute the output of non-linear functions of the form

$$[\![x]\!]_q \geq^? [\![y]\!]_q \ :\ [\![1]\!]_2 \ \textbf{else}\ [\![0]\!]_2,$$

$$[\![x]\!]_q \stackrel{?}{=} [\![y]\!]_q \ : \ [\![1]\!]_2 \ \textbf{else} \ [\![0]\!]_2.$$

The most common solutions involve first converting the inputs to their corresponding bitwise sharings $[\![x]\!]_q \rightarrow [\![x_\lambda]\!]_2 \cdots [\![x_1]\!]_2$ and $[\![y]\!]_q \rightarrow [\![y_\lambda]\!]_2 \cdots [\![y_1]\!]_2$ for $\lambda \geq \lceil \log(q) \rceil$ such that $\sum x_i 2^i = x$ and $\sum y_i 2^i = y$. Afterward, a series of bitwise operations are carried out to determine which of the two bit strings is greater than or equal to the other. The efficiency of a given protocol depends on the constraints of the secure computational environment for which it is designed. We direct the reader to [28] for a comprehensive discussion of comparison protocol design. For the computational environment used in this work, (i.e. 2PC, semi-honest security setting, linear secret sharing modulo 2^λ), the most similar protocol to our work is proposed by Bogdanov et al. for the Sharemind framework [29], though this framework is designed for 3PC. Our work and that of [29] rely on observations that hold for two's complement representations in $\mathbb{Z}_{2^\lambda}$, and work for numbers x and y such that $|x - y| < 2^{\lambda-1}$ holds, which can be easily enforced by only using a sub-range which is less than half of the available range $2^\lambda - 1$. This is a relatively weak limitation because any desired range can be injected into a larger integer ring. However, it fails for applications that rely on other MPC protocols for which the existence of modular inverses must be guaranteed.

The key insight we use to compute $x \stackrel{?}{\geq} y$ securely is the following: for x, y in two's complement form where $|x - y| < 2^{\lambda-1}$, we have that $y > x \iff 0 > x - y \iff \mathsf{MSB}(x-y) = 1$, where $\mathsf{MSB}(\cdot)$ denotes the *most significant bit* of a value. Hence $x \geq y \iff \mathsf{MSB}(x - y) = 0$. Similar logic can be used to derive an equality check $x \stackrel{?}{=} y$:

$$\begin{aligned} x = y &\iff (x \geq y) \wedge (y \geq x) \\ &\iff \mathsf{MSB}(x - y) = \mathsf{MSB}(y - x) = 0 \end{aligned}$$

Note that due to the two's complement format, it is not possible to have $\mathsf{MSB}(x - y) = \mathsf{MSB}(y - x) = 1$.

The above shows that the efficiency of computing $x \stackrel{?}{\geq} y$ and $x \stackrel{?}{=} y$ is limited only by our ability to extract the most significant bit of a secret-shared value. Bogdanov et al. use a recursive carry look-ahead construction to decompose the difference $x - y$ into its bitwise sharing. Other solutions designed for sharing over prime fields use a similar approach but take advantage of the fact that $\mathsf{MSB}(x) = \mathsf{LSB}(2x \mod q)$ for odd q [30], where $\mathsf{LSB}(\cdot)$ denotes the least significant bit.

In this section, we propose protocols for comparison and equality tests over $\mathbb{Z}_{2^\lambda}$ – π_{GEQ} and π_{EQ}, respectively – that circumvent a full-blown bit decomposition by extracting only the most significant bit – causing a significant reduction to the total data transfer. Our approach is based on a modification of the optimized bit decomposition protocol $\pi_{\mathsf{decompOPT}}$ presented in [4].

Note that when working with a two's complement fixed-point representation over $\mathbb{Z}_{2^\lambda}$, all bits outside of an injected ring $\mathbb{Z}_{2^{a+b}}$, where a is the number of fractional bits and b is the number of integer bits, are equivalent to the most significant bit. It follows in this case that extracting the $(a+b+1)$-th bit is equivalent to extracting the most significant bit which further reduces the depth of the arithmetic circuit.

Protocol 2: Secure protocol π_{GEQ} computes the integer comparison.

Input : $[\![x]\!]$, $[\![y]\!]$ such that $|x - y| < 2^{\lambda-1}$, $\alpha :=$ the lowest bit position guaranteed to be equal to the MSB.
Output: $[\![x]\!] \stackrel{?}{\geq} [\![y]\!] \ : \ [\![1]\!]_2$ **else** $[\![0]\!]_2$

1 Let $[\![\mathsf{diff}]\!] \leftarrow [\![x]\!] - [\![y]\!]$
2 Let $[\![\mathsf{MSB}]\!]_2 \leftarrow \pi_{\mathsf{BTX}}([\![\mathsf{diff}]\!], \alpha)$
3 **return** $1 \oplus [\![\mathsf{MSB}]\!]_2$

Protocol 3: Secure protocol π_{EQ} computes the integer equality test.

Input : $[\![x]\!]$, $[\![y]\!]$ such that $|x - y| < 2^{\lambda-1}$, $\alpha :=$ the lowest bit position guaranteed to be equal to the MSB.
Output: $[\![x]\!] \stackrel{?}{=} [\![y]\!] \ : \ [\![1]\!]_2$ **else** $[\![0]\!]_2$

1 Let $[\![\mathsf{d}_1]\!] \leftarrow [\![x]\!] - [\![y]\!]$ and $[\![\mathsf{d}_2]\!] \leftarrow [\![y]\!] - [\![x]\!]$
2 Let $[\![\mathsf{MSB}_1]\!]_2 \leftarrow \pi_{\mathsf{BTX}}([\![\mathsf{d}_1]\!], \alpha)$ and $[\![\mathsf{MSB}_2]\!]_2 \leftarrow \pi_{\mathsf{BTX}}([\![\mathsf{d}_2]\!], \alpha)$// run in parallel
3 **return** $1 \oplus [\![\mathsf{MSB}_1]\!]_2 \oplus [\![\mathsf{MSB}_2]\!]_2$

In our protocols, α is always set to be the highest bit position in the ring. However, in the case that it is public knowledge that the ring size is much larger than the secret data elements are likely to be, α can be decreased by a user-defined amount to improve performance. Extracting a lower bit position decreases the number of required communication rounds.

The two-party protocol $\pi_{\mathsf{decompOPT}}$ for performing a full decomposition of a $\mathbb{Z}_{2^\lambda}$-shared secret into bitwise sharings over $\mathbb{Z}_2$ is, to our knowledge, the most efficient in the literature. It is based on a *matrix composition network* that computes the difference between each bitwise sum of two secret shares and the corresponding "actual" bit of the secret value in $\mathbb{Z}_{2^\lambda}$. See [4] for a complete de-

scription. An important aspect of this approach is that computing the difference for the α-th bit depends only on $\lceil \log(\alpha-1) \rceil$ rounds of matrix composition, where each matrix composition requires 4 bits of data transfer. In addition, the difference for the α-th bit is independent of all results for lower order bits.

In Protocols 2 and 3, we make use of a straightforward modification of $\pi_{\mathsf{decompOPT}}$ for extracting the α-th bit from a $\mathbb{Z}_{2^\lambda}$-shared secret between two parties, hereafter π_{BTX} (see Appendix B for details). The total number of communication rounds to extract the α-th bit is $\lceil \log(\alpha-1) \rceil + 1$ with total data transfer of $6\alpha - 10$ bits. It follows that π_{GEQ} has the same number of rounds and the same data transfer as $\pi_{\mathsf{decompOPT}}$, and π_{EQ} has the same number of rounds but twice the data transfer of $\pi_{\mathsf{decompOPT}}$.

4 Secure discretization

Discretization or "binning" is a common form of data preprocessing, aimed at grouping continuous or numerical values into a smaller number of bins (buckets). In a data set with information about social media users, the feature *age* could for instance be discretized into the bins 0-24, 25-34, 35-49, 50+. The threshold values for the bins are typically derived from the data in an unsupervised manner. We use *equal-width binning*, which means that the range of feature values is divided into a predefined number of bins of the same width.

Let D be a vector containing the original feature values (e.g. the ages of all the users in the data set). The range of D is bounded by the smallest and the largest value occurring in D, i.e. $\min(D)$ and $\max(D)$. To divide this range into p bins of the same width, thresholds need to be placed at

$$h_i = \min(D) + i \cdot \frac{\max(D) - \min(D)}{p} \tag{2}$$

for $i = 1, \ldots, p-1$. The challenge is that neither Alice nor Bob may have direct access to $\min(D)$ and $\max(D)$ because they each may only have shares of the values of D. This means that they need to jointly execute a secure protocol, hereafter π_{minmax}, for computing the minimum and the maximum values of D.

To compute secret sharings of $\min(D)$ and $\max(D)$ without revealing any information about D, a secure protocol can be formulated similarly to a naive sequential search solution in the clear. That is, start by comparing the first and second elements of D to determine an initial estimate of the max and min. Next, iterate through all remaining elements and adjust the max and min estimates when a new largest or smallest element is found. After the n-th element of D is checked, the estimates are guaranteed to be the global min and max. The only necessary adaptation for this algorithm to act as an oblivious protocol is to require that the comparisons between the current estimates and each new element of D are performed with π_{GEQ} and that the reassignments are handled with multiplication rather than control flow logic. For example, the comparison based branch operation "**if** $a \geq b$ **then** $b = a$" can be rephrased as

$$\begin{aligned} [\![c]\!] &\leftarrow \pi_{\mathsf{2toQ}}(\pi_{\mathsf{GEQ}}([\![a]\!], [\![b]\!])) \\ [\![b]\!] &\leftarrow [\![c]\!] \cdot [\![a]\!] + (1 - [\![c]\!]) \cdot [\![b]\!] \end{aligned} \tag{3}$$

where c is 1 or 0, depending on the outcome of the comparison $a \geq b$. This form of conditional assignment does not allow Alice nor Bob to learn anything about which branch of the control flow sequence was followed to arrive at the outcome. An additional detail is that because π_{GEQ} returns secret shares over $\mathbb{Z}_2$, the result must first be converted to a ring representation with π_{2toQ} before the multiplication can be carried out.

Protocol π_{minmax} has a linear number of communication rounds when carried out in the naive formulation that is described in Protocol 4. However, it can be improved straightforwardly with the same optimisation technique used for securely computing the repeated product over a vector of values in which pairwise products are taken until only one value remains [22]. The protocol π_{minmax} is analogous to repeated multiplication because both multiplication and the max/min functions are associative. So, the sequence of many repeated applications can be altered to reduce the total number of consecutive, mutually dependent applications. The basic observation is that the global minimum of D is contained in the set of all pairwise minima of D. More-

Protocol 4: Secure min/max-finding protocol π_{minmax}

Input : $[\![D]\!]$, number n of elements in $[\![D]\!]$
Output: $[\![d_{\mathsf{min}}]\!]$, $[\![d_{\mathsf{max}}]\!]$
Let $[\![\geq^?]\!] \leftarrow \pi_{\mathsf{2toQ}}(\pi_{\mathsf{GEQ}}([\![d_2]\!], [\![d_1]\!]))$
Let $[\![d_{\mathsf{min}}]\!] \leftarrow [\![\geq^?]\!] \cdot [\![d_1]\!] + (1 - [\![\geq^?]\!]) \cdot [\![d_2]\!]$
Let $[\![d_{\mathsf{max}}]\!] \leftarrow [\![\geq^?]\!] \cdot [\![d_2]\!] + (1 - [\![\geq^?]\!]) \cdot [\![d_1]\!]$
for $i \leftarrow 3$ **to** n **do**
    $[\![\geq^?_{\mathsf{min}}]\!] \leftarrow \pi_{\mathsf{2toQ}}(\pi_{\mathsf{GEQ}}([\![d_i]\!], [\![d_{\mathsf{min}}]\!]))$
    $[\![\geq^?_{\mathsf{max}}]\!] \leftarrow \pi_{\mathsf{2toQ}}(\pi_{\mathsf{GEQ}}([\![d_i]\!], [\![d_{\mathsf{max}}]\!]))$
    $[\![d_{\mathsf{min}}]\!] \leftarrow [\![\geq^?_{\mathsf{min}}]\!] \cdot [\![d_{\mathsf{min}}]\!] + (1 - [\![\geq^?_{\mathsf{min}}]\!]) \cdot [\![d_i]\!]$
    $[\![d_{\mathsf{max}}]\!] \leftarrow [\![\geq^?_{\mathsf{max}}]\!] \cdot [\![d_i]\!] + (1 - [\![\geq^?_{\mathsf{max}}]\!]) \cdot [\![d_{\mathsf{max}}]\!]$
end
return $[\![d_{\mathsf{min}}]\!]$, $[\![d_{\mathsf{max}}]\!]$

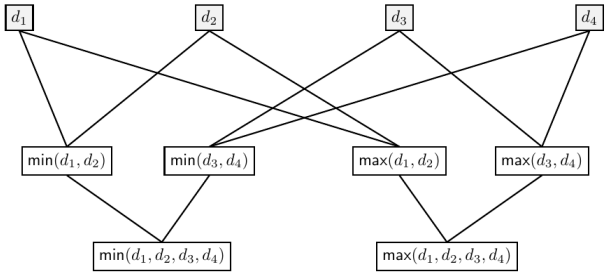

Fig. 1. Optimized π_{minmax}: an example circuit to compute π_{minmax} on an input vector of size $n = 4$.

over, if the minimum is computed between each pair of entries, and this process is repeated until only one pair remains, the result of the final pairwise comparison is $\min(D)$. The same principle extends to finding the global maximum. See Figure 1 for an illustration.

As a result, π_{minmax} can be computed in a circuit of depth $\lceil \log(n) \rceil$, where at each layer there are $\lceil \log(a + b) \rceil + 2$ rounds of communication, where a, b are the number of fractional and integer bits, respectively, used in the representation. The data transfer complexity is given by the sum over two invocations of π_{GEQ} and two invocations of π_{2toQ} indexed by the number of operand pairs at each level of the circuit. Then, for n integers, the total number of rounds is $\lceil \log(n) \rceil (\lceil \log(a + b) \rceil + 2)$ with total data transfer of $\sum_{i=0}^{\lceil \log(n) - 1 \rceil} 2^{i+1}(6(a + b) - 10 + 2\lceil \log(q) \rceil)$.

At the end of this protocol, Alice and Bob have secret sharings of $\min(D)$ and $\max(D)$, which they can then securely combine to compute secret sharings of each of the h_i thresholds in Equation (2). For example, if $\min(D) = 0$, $\max(D) = 150$, and $p = 6$, then Alice and Bob would compute secret shares of the thresholds $25, 50, 75, 100$, and 125. We assume that p is publicly known, i.e. Alice and Bob know how many bins need to be created. As explained above, they may however not know the value of $\min(D)$ or of $\max(D)$ in the clear, and our discretization protocol does not leak this value. In Protocol 5, d_i is the i-th element of D. D is a list of secret shared fixed point numbers to be discretized.

After they have computed shares of the h_i thresholds, Alice and Bob can map each value d_j from D into its correct bin number $bin(d_j)$ by executing protocol π_{DISC} (described in Protocol 5). Note that Alice and Bob each only have a share of d_j, and a share of each threshold h_i. They could run the secure comparison protocol $\pi_{\mathsf{GEQ}}(d_j, h_i)$ for each of the threshold values h_i and count in how many cases the comparison yielded a "true" response. For example, for value $d_j = 80$

Protocol 5: Secure equal-width discretization protocol π_{DISC}

Input : $[\![D]\!]$, number n of elements in $[\![D]\!]$, public number of buckets p
Output: $[\![D']\!]_2$:= one-hot-encoding of the bucket membership of each $d \in D$

1 The parties call π_{minmax} on $[\![D]\!]$ and receive $[\![d_{\mathsf{min}}]\!]$, $[\![d_{\mathsf{max}}]\!]$
2 $[\![d_{\mathsf{range}}]\!] \leftarrow [\![d_{\mathsf{max}}]\!] - [\![d_{\mathsf{min}}]\!]$
3 **for** $i \leftarrow 1$ **to** $p - 1$ **do**
4 $\quad [\![h_i]\!] \leftarrow [\![d_{\mathsf{min}}]\!] + \pi_{\mathsf{trunc}}(\frac{i}{p} \cdot [\![d_{\mathsf{range}}]\!])$
5 **end**
6 **for** $i \leftarrow 1$ **to** n **do**
7 $\quad [\![e_j]\!]_2 \leftarrow \pi_{\mathsf{GEQ}}([\![d_i]\!], [\![h_j]\!])$ for $j \in 1, \ldots, p - 1$
8 $\quad [\![d'_{i,(0)}]\!]_2 \leftarrow 1 - [\![e_1]\!]_2$
9 $\quad [\![d'_{i,(j)}]\!]_2 \leftarrow [\![e_j]\!]_2 \cdot (1 - [\![e_{j+1}]\!]_2)$ for $j \in 1, \ldots, p - 2$
10 $\quad [\![d'_{i,(p-1)}]\!]_2 \leftarrow [\![e_{p-1}]\!]_2$
11 **end**
12 **return** $[\![D']\!]_2$

and thresholds $25, 50, 75, 100, 125$, the first 3 comparison tests with π_{GEQ} would result in secret shares of 1, while the remaining 2 tests would result in secret shares of 0. Adding those up securely, Alice and Bob would derive shares of 3, which is the correct bin number of $d_j = 80$, assuming that we start counting bins at 0.

However, in order to expedite the process of converting this discretized data into the required format used by the decision tree learning protocols that follow, we choose to output a one-hot-encoding of $bin(d_j)$ where the $bin(d_j)$-th bit position (with lowest order on the left) is a secret sharing of 1 and all other bits are secret sharings of 0. Building from the previous example where $p = 6$ and $bin(d_j) = 3$, π_{DISC} outputs secret shares of the vector $(0, 0, 0, 1, 0, 0)$.

This conversion to one-hot-encoding is carried out by noting that any value compared against a set of $p-1$ increasing thresholds will return $0 \leq k \leq p - 1$ true results followed by $p - k - 1$ false results. The position of the 1 in the one-hot-encoding vector is determined by the position of the first false result. In the protocol that follows, the notation $d'_{i,(j)}$ means the j-th bit of the one-hot-encoding vector for d_i.

Protocol π_{DISC} (detailed in Protocol 5) adds only additional $\lceil \log(a + b) \rceil + 2$ communication rounds after π_{minmax}, where a, b are the fractional and integer precision of the injected fixed point subring, respectively. All calls to π_{GEQ} are mutually independent (Line 7), so they are computed in a single batch in $\lceil \log(a + b) \rceil + 1$ rounds. Similarly, all products $e_j \cdot (1 - e_{j+1})$ (Line 9) are mutually independent and require one round. Comput-

ing the range and thresholds $[\![h_i]\!]$ (Line 2-5) is entirely comprised of local operations as all constants i/p are public and π_{trunc} is a local protocol in the 2PC scenario.

5 Secure random forest training

DT ensembles gained much popularity during the 2000s and have remained state-of-the-art methods for many classification tasks to date. A DT ensemble consists of a set of DTs that each infer a class label for a new instance; the final label is determined through (weighted) majority voting. DT ensembles differ from each other in the way they are trained. A successful approach, known as random forest (RF), combines *bagging* and *subspace sampling* to make the DTs in the ensemble sufficiently different from each other [31].

Bagging refers to the fact that, given a training data set S with n examples, for each DT a bootstrap replica of S is created by sampling n times with replacement from the data set S. In this paper, we present a slight adaptation of the traditional RF training algorithm in which we sample only $s \leq n$ times. Working with a smaller data set allows to train a RF more efficiently in case n is very large. *Subspace sampling* refers to the fact that for each DT, only $k < f$ randomly selected features of the original feature set are retained. The resulting RF training algorithm – in the clear – is presented in Algorithm 6. Note that the use of ID3 [10] on Line 4 indicates that the feature values are assumed to be categorical; otherwise one would typically use C4.5 [19] instead.

Our approach for training a RF in a privacy-preserving manner over data with continuous feature values is to first discretize the feature values using π_{DISC}. Next we need techniques for randomly selecting features and samples in a secure manner, similar to Line 2 and 3 in Algorithm 6. Finally, we train DTs as in Line 4, using the secure protocol π_{SID3T} adapted from de Hoogh et al. [14]. π_{SID3T} assumes that the input data is presented in a one-hot-encoded (OHE) format, because that allows for efficient calculation of the Gini index which is needed to select split features while training a DT. That is the reason why we designed π_{DISC} to already output one-hot-encodings. Note that after a discretization of all feature values of the data set S, using p bins per feature, the secret shared data set $[\![S_{\mathsf{disc}}]\!]_2$ consists of a matrix S_{disc} of size $n \times f \cdot p$ containing the one-hot-encodings. For oblivious selection of k features we use a $f \cdot p \times k \cdot p$ selection matrix FS that is generated by the TI and secret shared with the parties. In FS, identity submatrices of size $p \times p$ are used to indicate that a feature is selected, and the remaining positions are filled with $p \times p$ submatrices of zeroes. Since each feature is selected at most once, no identity submatrices are aligned horizontally or vertically. Note that m such matrices are randomly populated in this manner by the TI, with m the total number of DTs in the RF. In order to extract the desired features, the secure protocol for random feature selection calls the secure matrix multiplication protocol π_{DMM} to multiply the OHE-style data set $[\![S_{\mathsf{disc}}]\!]_2$ with a

Algorithm 6: Algorithm for training a random forest classifier.

Input : A set S with n training samples (each sample has f features and one class label), the number m of trees in an ensemble, the number of features k used in each tree, the number of samples s used in each tree, the depth d of each tree.

Output: An ensemble of trees $T = t_1, t_2, \ldots, t_m$

1 **for** $j \leftarrow 1$ **to** m **do**
2 Randomly select k of the f features of S.
3 Randomly select with replacement s samples $S' = i_1, i_2, \ldots, i_s$ among all samples of S after features are selected.
4 Train a decision tree t_i of depth d on S' using ID3.
5 **end**
6 **return** $T = t_1, t_2, \ldots, t_m$

Protocol 7: Secure protocol π_{RF} for training a random forest with continuous data.

Input : A secret shared set $[\![S]\!]$ with n training samples (each sample has f features), the number m of trees in an ensemble, the number of buckets p for each feature, the number of features k used in each tree, the number of samples s used in each tree, the depth d of each tree.

Output: A random forest model $[\![RF]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$.

1 (Offline Phase) The TI generates and secret shares m 0/1 valued feature selection matrices $[\![FS^{(1)}]\!]_2, \ldots, [\![FS^{(m)}]\!]_2$ of size $f \cdot p \times k \cdot p$, and m 0/1 valued sample selection matrices $[\![SS^{(1)}]\!]_2, \ldots, [\![SS^{(m)}]\!]_2$ of size $s \times n$
2 Discretize each feature of $[\![S]\!]$ into p bins using protocol π_{DISC} to get $[\![S_{\mathsf{disc}}]\!]_2$
3 **for** $i \leftarrow 1$ **to** m **do**
4 $[\![S_{\mathsf{FS}}]\!]_2 \leftarrow [\![S_{\mathsf{disc}}]\!]_2 \cdot [\![FS^{(i)}]\!]_2$.
5 $[\![S_{\mathsf{SR}}]\!]_2 \leftarrow [\![SS^{(i)}]\!]_2 \cdot [\![S_{\mathsf{FS}}]\!]_2$.
6 Use π_{SID3T} to securely train a decision tree $[\![t_i]\!]$ of depth d with the data set $[\![S_{\mathsf{SR}}]\!]_2$.
7 **end**
8 **return** $[\![RF]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$.

feature selection matrix FS that is secret shared by the TI. For example, if $f = 3$, $p = 3$ and $k = 2$, the matrix

$$FS = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

will retain the first 3 and the last 3 columns of S_{disc}, thereby effectively selecting the one-hot-encodings corresponding to the first and third features of S.

The procedure used to sample the s instances with replacement is similarly done by a multiplication with a $s \times n$ matrix SS that is secret shared by the TI. The only difference is that a column in SS can contain multiple ones, as the choice is with replacement.

Protocol π_{RF} for secure training RFs is described in Protocol 7. The loop in π_{RF} can be executed in parallel. The original ID3 protocol will not grow a tree to pre-specified depth d if an early termination condition is satisfied (e.g. all training examples in a branch have the same class label). We have modified it in π_{SID3T} such that the secure trees will always grow to depth d by adding dummy nodes where necessary. If a node satisfies an early termination condition, then the node will override the classifications of all of its child nodes (dummy nodes) in an oblivious manner. This has the security advantage of not revealing the true depth of each sub-tree, while concealing which nodes actually classify.

6 Secure extra trees training

Besides the RF training algorithm from the previous section, several other successful algorithms exist for training ensembles of decision trees. One of these algorithms, used to train so-called "Extremely Randomized Trees", or "extra-trees" (XT) for short, was developed specifically for data with numerical features [20]. In addition to randomly selecting subsets of features during the tree construction process, the XT training algorithm also randomly selects a threshold α_j for each feature a_j to effectively turn the numerical feature a_j into a binary feature, based on whether the feature value is greater than or equal to α_j, or not.

An algorithm for training an XT classifier in the clear, adapted from [20], is presented in Algorithm 8.

Algorithm 8: Algorithm for training an extra-trees classifier.

Input : A training set S with continuous data and n samples (each sample has f features), the number k of features to consider in each tree, the number m of trees in the ensemble, the depth d of each tree.

Output: An ensemble of trees $XT = t_1, \ldots, t_m$.

For each feature j, find its minimum and maximum values, $\min_j$ and $\max_j$.
for $i \leftarrow 1$ **to** m **do**
  Select k random indices $j_1, \ldots, j_k \in \{1, \ldots, f\}$.
  for $\ell \leftarrow 1$ **to** k **do**
    For a uniformly random $r \in (0, 1)$, $\alpha_\ell \leftarrow r \cdot (\max_{j_\ell} - \min_{j_\ell}) + \min_{j_\ell}$.
    for $s \leftarrow 1$ **to** n **do**
      if $S[s, j_\ell] \geq \alpha_\ell$ **then**
        $S'[s, \ell] \leftarrow 1$
      else
        $S'[s, \ell] \leftarrow 0$
      end
    end
  end
  Train a decision tree t_i of depth d on S' using ID3.
end
return $XT = t_1, \ldots, t_m$.

It constructs an ensemble of m decision trees. As before, S can be thought of as an $n \times f$ matrix in which the rows correspond to instances and the columns to features. As in Section 5, each decision tree is trained over a randomly chosen subset of k of the f available features. Moreover, each continuous feature a_j is binarized by choosing a random threshold α_j in the range of possible values of a_j, and replacing the feature value by 1 if it meets the threshold, and 0 otherwise. Each such binarization is specific for a particular feature in a particular tree of the ensemble; for another tree in the ensemble the same feature a_j might be reused with a different random binarization. Note that all loops in the algorithm can be executed in parallel. Algorithm 8 differs from the original extremely randomized trees algorithm [20] in the sense that in the latter the random choices for the features and the random choices for the cut-off points are made for each node in each decision tree, whereas in Algorithm 8 they are made once for each decision tree. We have observed that the difference between the two approaches can have some detrimental effect on the accuracy. To remedy this, we can sample the subspace of S with replacement, potentially choosing multiple splits per feature. So for example, if we have a data set where $f = 30$, we can let $k = 60$, thus increasing the diversity of the data set. In our tests, this

Protocol 9: Protocol π_{XT} for securely training an extra-trees classifier.

Input : A secret shared training set $[\![S]\!]$ with continuous data and n samples (each sample has f features), the number k of features to consider in each tree, the number m of trees in the ensemble, the depth d of each tree.

Output: A secret shared ensemble of trees $[\![XT]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$.

1 (Offline Phase) The TI secret shares m random 0/1-valued feature selection matrices $[\![FS^{(1)}]\!], \ldots, [\![FS^{(m)}]\!]$ of size $f \times k$, where each column contains a single 1, and no rows have more than a single 1. The TI also distributes $k \cdot m$ uniformly random ratios $[\![r^{(1)}]\!], \ldots, [\![r^{(k \cdot m)}]\!]$ $\in [1, 2^a - 1]$ (which approximates $r \in (0, 1)$ in $\mathbb{R}$).
2 Compute the vectors $[\![min]\!]$ and $[\![max]\!]$, by using $([\![min_j]\!], [\![max_j]\!]) \leftarrow \pi_{\mathsf{minmax}}([\![S_j]\!], n)$ for each column S_j $(j = 1, \ldots, f)$ of S.
3 **for** $i \leftarrow 1$ **to** m **do**
4 $[\![S_{\mathsf{FS}}]\!] \leftarrow [\![S]\!] \cdot [\![FS^{(i)}]\!]$
5 $[\![\boldsymbol{r}]\!] \leftarrow ([\![r^{((i-1)k+1)}]\!], \ldots, [\![r^{(ik)}]\!])$
6 $[\![\boldsymbol{\alpha}]\!] \leftarrow \pi_{\mathsf{trunc}}([\![\boldsymbol{r}]\!] * (([\![max]\!] - [\![min]\!]) \cdot [\![FS^{(i)}]\!])) + [\![min]\!] \cdot [\![FS^{(i)}]\!]$
7 **for** $\ell \leftarrow 1$ **to** k **do**
8 **for** $p \leftarrow 1$ **to** n **do**
9 $[\![D[p, \ell, 1]]\!]_2 \leftarrow \pi_{\mathsf{GEQ}}([\![S_{\mathsf{FS}}[p, \ell]]\!], [\![\boldsymbol{\alpha}[\ell]]\!])$
10 $[\![D[p, \ell, 0]]\!]_2 \leftarrow 1 - [\![D[p, \ell, 1]]\!]_2$
11 **end**
12 **end**
13 Let $[\![t_i]\!]$ be the decision tree of depth d trained using π_{SID3T} with the data set $[\![D]\!]_2$.
14 **end**
15 **return** $[\![XT]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$.

approach seems to close the gap in accuracy caused by discretizing per tree instead of per node.

Our Protocol π_{XT} for securely training an XT classifier is given in Protocol 9. As with the RF approach from Section 5, at the start of the secure XT training protocol, Alice and Bob have secret shares of the training data set S. At the end of the protocol, they have secret shares of an XT classifier $XT = t_1, \ldots, t_m$. Protocol 9 uses several building blocks that have already been introduced and explained before.

In the offline phase, the TI generates and secret shares feature selection matrices $FS^{(1)}$, $FS^{(2)}$, …, $FS^{(m)}$. The feature selection in Line 4 of Protocol 9 is based upon the secure matrix multiplication protocol π_{DMM}, in the same way as in Section 5. Note that here the feature selection matrices are of size $f \times k$ (with 1's representing the selections), as the features are not represented using one-hot-encodings at this point. The TI also generates the equivalent of the r values from Line 5 in Algorithm 8 that are used for random selection of cut-off points. Note that, while in Algorithm 8, each r is a real number between 0 and 1, in Line 1 of Protocol 9, the randomly chosen values are integers between 1 and 2^a, where a is the number of fractional bits in the fixed-point representation that we use throughout this paper (see Section 2).

In Line 2 in Protocol 9, the parties use Protocol 4 (π_{minmax}) to compute the minimum and maximum value of each feature. In Line 5, we select a subset of the random ratios of size k which will be used to calculate the vector $[\![\boldsymbol{\alpha}]\!]$. This vector is calculated on Line 6, and will retain secret shared values whose sum is a random value between the minimum and maximum associated to all features that are dictated by the feature selection matrix $[\![FS^{(i)}]\!]$. Note that on Line 6, $\boldsymbol{\alpha}$, $\boldsymbol{r}$ are $1 \times k$ vectors, max and min are $1 \times f$ vectors, while $FS^{(i)}$ is a $f \times k$ matrix.

The loop on Line 7-12 creates an one-hot-encoding of the binarized version of the data set. Each continuous feature value $S_{\mathrm{FS}}[p, \ell]$ is compared with the threshold $\alpha[\ell]$ using the secure comparison protocol π_{GEQ}. The result, which is the binarized version of the data set encoded as OHE, is stored in the matrix $[\![D]\!]_2$, which is secret shared among the parties. Finally, the parties execute the secure ID3 decision tree training protocol π_{SID3T} to jointly train a decision tree over $[\![D]\!]_2$. Similarly to Section 5, our version of π_{SID3T} grows each tree to the same specified depth d, to hide the true structure of the tree.

The two main differences between π_{XT} and π_{RF} are: (1) that the π_{XT} uses (a modified version of) *all* training examples for each tree, rather than using bootstrap replicas generated with bagging, and (2) that π_{XT} splits the range of feature values by randomly selecting a single threshold. This reduces the computational complexity with respect to π_{RF}, as π_{RF} trains p-nary trees while π_{XT} only trains binary trees.

7 Analysis and results

7.1 Complexity analysis

We start by describing the asymptotic computational and communication complexities of our protocols. We also provide a comparison with the only work in the literature that addresses the problem of training tree-based models based on continuous data with MPC [18]. Let m be the number of features, S_k the attribute sub-

Data set	#inst	#feat	Model	Model size as #nodes		Accuracy		Time
				Sklearn	Secure (dummy %)	Sklearn	Secure	Secure
BC	569	30	DT	16	626 (28%)	93.1%	90.2%	5.3 sec
			RF	954	21,700 (51%)	93.7%	93.2%	18.5 sec
			XT	1,351	1,650 (58%)	96.3%	96.5%	35.2 sec
ECG	14,552	140	DT	3	3 (0%)	100.0%	100.0%	5.7 sec
			RF	60	60 (0%)	100.0%	100.0%	79.6 sec
			XT	60	60 (0%)	100.0%	100.0%	43.6 sec
BACK	310	12	DT	18	626 (25%)	79.0%	70.0%	3.1 sec
			RF	1,872	312,600 (47%)	82.3%	68.0%	9.8 sec
			XT	2,677	3,250 (77%)	83.4%	81.3%	39.2 sec
IV-GSE	225	12,634	DT	3	3 (0%)	64.9%	59.6%	91.1 sec
			RF	60	60 (0%)	63.6%	62.3%	12.6 sec
			XT	124	2,970 (71%)	63.4%	63.5%	12.6 sec

Table 1. Accuracy and runtime results for tree based models. All results are obtained with 5-fold cross-validation.

set size, n the number of instances, S_n the instance subset size, d the desired tree depth, and T the number of trees per ensemble. Our protocol for training decision trees π_{SID3T} requires $O(mn + T2^d(S_k^2 + n))$ secure binary multiplications, $O(Tn(m + 2^d S_k))$ secure multiplications over $\mathbb{Z}_q$, and $O(d\log(m))$ rounds. Our protocol for training random forests over continuous data π_{RF} requires $O(T2^d(S_k^2 + S_n) + TnS_k(S_n + m))$ secure binary multiplications, $O(T2^d S_k S_n)$ multiplications over $\mathbb{Z}_q$, and $O(d\log(Sk))$ rounds. Finally, our protocol for secure training of extra trees π_{XT} requires $O(mn + T2^d(S_k^2 + n))$ secure binary multiplications, $O(Tn(m + 2^d S_k))$ secure multiplications over $\mathbb{Z}_q$, and $O(\log(n) + d\log(S_k))$ rounds.

The protocol proposed in [18] (where a sorting based approach is employed) uses $O(dn\log^2(n))$ rounds of communication and $O(mn\log(n)(2^d + \log(n))$ $\mathbb{Z}_q$ multiplications. When all features are continuous, each of our algorithms outperforms the protocol in [18] in terms of communication significantly, except for the pathological case of data sets with exponentially more features than instances. This is because we avoid secure sorting. The comparison between the number of multiplications required is most straightforward with DT as that case is similar to the one covered in [18]. Our algorithm decreases required multiplications by a multiplicative factor of $\log(n)$. For a small data set with a mere 1k entries our solutions are 10 times more efficient. The improvement in round complexity is even more dramatic: from $O(d(n\log^2(n) + \log(m)))$ to $O(d\log(m))$.

7.2 Empirical accuracy results

We implemented the proposed protocols in Rust[1] and performed accuracy and runtime experiments on 4 different data sets, shown in Table 1, namely the Breast Cancer[2] data set (BC), the ECG Heartbeat[3] data set (ECG), the Lower Back Pain Symptoms[4] data set (BACK) and the Track IV-GSE 2034 data set (IV-GSE) from the iDASH 2019 competition on secure genome analysis.[5] All data sets are for binary classification problems. As shown in Table 1, the data sets vary in the number of continuous valued input features, as well as in the number of instances. The original ECG data set had several columns that mostly contained a value of 0, and were unhelpful to train on. We removed every feature that contained 80% or more values of zero, reducing the ECG data set from 188 features to 140. All results in Table 1 are obtained with 5-fold cross-validation, i.e. the accuracies and runtimes are all averages obtained over 5 folds.

The "Accuracy–Sklearn" column in Table 1 contains accuracy results obtained by training tree based models over each data set in the clear, with the well known Scikit-learn library [32]. This library has state-of-the-art implementations of non privacy-preserving versions of the ML algorithms considered in Table 1. We used grid

1 https://bitbucket.org/uwtppml/rustlynx/src/tree_ensembles/

2 https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

3 https://www.kaggle.com/shayanfazeli/heartbeat

4 https://www.kaggle.com/sammy123/lower-back-pain-symptoms-data set

5 http://www.humangenomeprivacy.org/

search to find hyperparameter values that yield good accuracy results, in line with those reported in the literature for each of the data sets; see Table 2 and below for more details. As can be observed in the "Accuracy–Sklearn" column, for some data sets (e.g. **IV-GSE**) it is substantially harder to obtain very high accuracy than for others (e.g. **ECG**).

The "Accuracy–Secure" column in Table 1 contains accuracy results obtained with our MPC protocols for training tree based models when the data sets are secret shared among two computing parties. The DT and RF results were obtained by running π_{DISC} followed by respectively π_{SID3T} and π_{RF}. The XT results were obtained by running π_{XT}. For the ring $\mathbb{Z}_{2^\lambda}$ we used $\lambda = 64$, using $a = 10$ bits for the fractional part and $b = 22$ bits for the integer part in the fixed point representation (see Section 2). Computations on fixed point numbers can cause accuracy loss, particularly when products of small numbers are computed in succession. The effect is equivalent to rounding all intermediate results of a long product to some number of decimal places instead of rounding the final result. We contend with this issue by scaling the data sets by a factor of 1000 before converting it to fixed point which avoids small values without requiring prior knowledge of the data. We note that scaling values in this manner does not have any effect on trees learned by ID3 in the clear.

During classification, to tally votes amongst the trees in the RF and XT classifiers trained with our secure protocols, we apply the same soft voting mechanism as what is used in Sklearn (alternative methods could be used as well, if desired). To this end, as explained in Section 2, for each classifying node, our decision tree learning protocol returns secret shared frequencies of each of the class labels in the subset of training instances that have reached that node. In soft voting each tree returns a probability distribution of the class labels as its vote. So for example, if the active classifying node in a tree has 70 positive examples, and 30 negative examples, then it would return a vote for 70% positive, and 30% negative. These proportions are then added up amongst all trees, and the class label with the most votes wins.

As we explained in prior sections, we made a variety of adaptations to the original tree model training algorithms (as implemented in Sklearn) to create MPC-friendly versions. The main high level distinction is that our protocols for DT, RF, and XT all rely on a round of discretization that creates a data set for each tree, while in Sklearn's implementations such discretization is performed for each node. In our protocols for DT and RF, our static discretization step is very explicit, as the computing parties first run π_{DISC} to discretize the data with equal-width binning, and subsequently train a DT or a RF on the discretized data with π_{SID3T} or π_{RF}. In our protocol π_{XT} for XT, discretization is performed once for each tree, by randomly choosing feature thresholds that remain fixed for the entire tree, while in Sklearn such thresholds are randomly chosen per node. This explains most of the differences between the "Sklearn Acc" and "Secure Acc" column results in Table 1, along with the fact that the implementations in Sklearn are refined with some bells and whistles that we did not include in our secure version. For example, the implementation of tree learning in Sklearn has an additional stopping criterion that stops growing a tree branch if the feature values are constant across the training instances in that branch. Not including this stopping criterion in our protocol was a deliberate choice to reduce communication and memory usage, but it also impacts accuracy. Despite these differences, Table 1 convincingly shows that our protocols are competitive with the in-the-clear algorithms in Sklearn in terms of accuracy.

A notable exception is the **BACK** data set, on which π_{SID3T} and π_{RF} clearly under-perform in terms of accuracy. The cause for this degradation in accuracy is that in π_{SID3T} and π_{RF} the data is first discretized with equal-width binning, and then a DT and RF are trained on the discretized data, respectively. In Sklearn on the other hand, the models are trained directly on the original, undiscretized data, with binning performed dynamically by looking for an optimal split point in each node. This enables the algorithm to take into account interdependencies between features as well as the class labels of the training examples, which can be beneficial depending on the data set and problem [33]. To verify this hypothesis we manually discretized the **BACK** data set with equal-width binning, and re-ran the tests with Sklearn, which gave us the same level of accuracy as obtained with π_{RF}, and even lower accuracy than π_{SID3T}. It's clear that the **BACK** data set benefits greatly from dynamic discretization. As explained in Section 6, π_{XT} offers a way to compensate for this lack of dynamic discretization by sampling the feature space more extensively and with replacement. The accuracy of π_{XT} on **BACK** is nevertheless still somewhat lower than the algorithms in the clear, but this is likely, as mentioned above, because the implementation of tree learning in Sklearn stops growing a branch when all feature values are constant. This stopping condition is likely to be satisfied for small data sets, and deep tree depths, and our tests on **BACK** satisfies both of these conditions (see

Table 2). On **BC** and **IV-GSE**, π_{XT} and π_{RF} were able to classify within $\pm 1.3\%$ of Sklearn, while π_{SID3T} underperformed. Lastly, on **ECG**, all protocols were able to learn the decision boundaries to perfectly separate the positive from the negative instances.

Finally, Table 1 contains two columns that compare the size of the models generated with Sklearn and with our MPC protocols, measured in terms of number of nodes. For the secure version, we also indicate what percentage of nodes are dummy nodes. The model size and the percentage of dummy nodes vary heavily depending on the hyperparameter choices (see Table 2). For example, the models generated from the **ECG** data contain 0 dummy nodes because the trees are of depth 1, forcing each node to be used in the classification phase. In contrast, when generating ensembles using π_{RF} and π_{XT} on the **BACK** dataset, we used many trees, all of which were relatively deep, creating ample opportunity to reach an early stopping condition. Dummy nodes are used to hide the true shape of the trees; their presence or absence has no impact on the accuracy.

Regarding the number of nodes in each model, an obvious outlier is π_{RF} using **BACK**. The number of nodes grows exponentially with respect to the depth, with the base of the exponent being the number of bins. It may then seem odd that π_{RF} performs roughly 4 times faster than π_{XT} despite producing nearly 100 times more nodes than π_{XT} when trained on **BACK**. This is because despite there being so many more nodes to generate, it is far easier to generate said nodes using π_{RF} since we use a small horizontal subset of the original data (see Table 2), whereas π_{XT} generates its nodes with the entire dataset in mind. We continue our discussion of runtime results below.

7.3 Runtime results

All our experiments were run on Microsoft Azure L48s_v2 machines with 48 vCPUs, 384.0 GiB Memory. Each of the two computing parties *Alice* and *Bob* ran on a separate machine (connected via Gigabit Ethernet network) which means that the results in the last column of Table 1 cover communication time in addition to computation time. The trusted initializer was implemented on a third machine. All reported runtimes are for the online phases of the protocols. As can be seen, our protocols are very fast, even on data sets with thousands of instances or features.

We repeat the accuracy and runtime results from Table 1 in Table 2, along with our choices for the hyperparameter values. We obtained these hyperparameter values by performing grid search while aiming for state-of-the-art accuracies on the respective data sets. The attentive reader will notice that the hyperparameter values differ between the algorithms as implemented in Sklearn, and our MPC-based protocols. As we explain in more detail below, the incentive to choose hyperparameter values differently for the MPC-based protocols goes hand in hand with the adaptations made to make the original ML algorithms more MPC-friendly.

For the Sklearn results in Table 2, the hyperparameter *sel. feat.* denotes how many features were randomly selected for each tree in the RF and XT classifiers (the original number of features for each data set is recalled in the first column of Table 2). *Trees* and *depth* denote how many trees were trained in each ensemble, and to what depth each tree was to be trained to. The parameter ϵ denotes what fraction of training instances needed to remain in a node for that node to branch out further. For example, if we have a data set of 1000 instances, and $\epsilon = 5\%$, then if the number of training instances that have reached a node during tree construction is 50 instances or less, the growing stops in that branch and the node becomes a classifying node. We used the same ϵ values for the secure protocols as for Sklearn (not repeated in the table for conciseness).

There are two additional columns of hyperparameters in Table 2 for the secure protocols, namely *bins*, which is the number of buckets of equal width that π_{DISC} generates for DT and RF, and *sel. inst.*, which denotes how many random instances were selected, per tree and with replacement, from each discretized set by π_{RF}. For the Sklearn implementation, we followed the convention of choosing *sel. inst.* equal to the total number of instances (which is recalled for each data set in the first column of Table 2) while for π_{RF} we systematically chose a smaller value for *sel. inst.* for efficiency reasons, as explained in Section 5.

$\mathbb{Z}_2$*-triples* and $\mathbb{Z}_q$*-triples* denote the number of Beaver triples consumed by the secure protocols at runtime. Table 2 expresses these values as multiples of 10^6 rounded to 4 significant digits. For brevity, the correlated randomness required to compute random feature selection (Line 4 of π_{RF} and π_{XT}) is omitted from the total due to its differing structure. π_{XT} incurs an additional tensor triple (U, V, W) sampled from $(\mathbb{Z}_q^{\mathsf{inst} \times \mathsf{feat}}, \mathbb{Z}_q^{\mathsf{trees} \times \mathsf{feat} \times \mathsf{sel.feat}}, \mathbb{Z}_q^{\mathsf{trees} \times \mathsf{inst} \times \mathsf{sel.feat}})$, where $W = U \cdot V$ contracted over feat. Similarly, π_{RF} requires a tensor of size $(\mathbb{Z}_2^{\mathsf{inst} \times \mathsf{feat} \cdot \mathsf{bins}}, \mathbb{Z}_2^{\mathsf{trees} \times \mathsf{feat} \cdot \mathsf{bins} \times \mathsf{sel.feat}}, \mathbb{Z}_2^{\mathsf{trees} \times \mathsf{inst} \times \mathsf{sel.feat} \cdot \mathsf{bins}})$. In addition, π_{RF} incurs another tensor triple of correlated randomness to

Sklearn Results							Secure Protocol Results								
Data		sel.feat.	trees	depth	ϵ	Acc	bins	sel.feat.	sel.inst.	trees	depth	Time	Acc	$\mathbb{Z}_2$-triples	$\mathbb{Z}_q$-trples
BC	**DT**	–	1	4	5%	93.1%	5	–	–	1	4	5.3 s	90.2%	13.89	44.16
inst: 569	**RF**	17	70	4	5%	93.7%	6	30	200	100	3	18.5 s	93.2%	25.79	646.7
feat: 30	**XT**	19	90	5	5%	96.3%	–	128	–	50	5	35.2 s	96.5%	201.7	739.2
ECG	**DT**	–	1	1	5%	100.0%	2	–	–	1	1	5.7 s	100.0%	677.9	25.37
inst: 14,552	**RF**	120	20	1	5%	100.0%	2	120	100	20	1	79.6 s	100.0%	939.7	49.69
feat: 140	**XT**	120	20	1	5%	100.0%	–	256	–	20	1	43.6 s	100.0%	3972	549.4
BACK	**DT**	–	1	4	1%	79.0%	5	–	–	1	4	3.1 s	70.0%	3.122	10.01
inst: 310	**RF**	8	90	5	1%	82.3%	5	10	30	100	5	9.8 s	68.0%	77.92	550.5
feat: 12	**XT**	10	120	6	1%	83.4%	–	64	–	50	6	39.2 s	81.3%	65.39	411.4
IV-GSE	**DT**	–	1	1	5%	64.9%	5	–	–	1	1	91.1 s	59.6%	2243	229.1
inst: 225	**RF**	1000	20	1	1%	63.6%	2	128	20	20	1	12.6 s	62.3%	678.1	14.53
feat: 12,634	**XT**	112	10	3	1%	63.4%	–	128	–	90	5	12.6 s	63.5%	3971	549.4

Table 2. Accuracy, runtime results (in seconds), and number of millions of pre-distributed triples for tree based models, along with the hyperparameter values that yielded these results. We used the same ϵ value for the secure protocols as for the Sklearn results. All results are collected with 5-fold cross-validation.

compute instance selection (Line 5). This triple is sampled from $(\mathbb{Z}_2^{\text{inst} \times \text{sel.feat} \cdot \text{bins}}, \mathbb{Z}_2^{\text{trees} \times \text{inst} \times \text{sel.inst}}, \mathbb{Z}_2^{\text{trees} \times \text{sel.inst} \times \text{sel.feat} \cdot \text{bins}})$. We direct the reader to [2] for a comprehensive discussion of the generalization of correlated randomness for matrix multiplication.

Our runtimes for DT are far shorter than those of XT and RF on the data sets with a small number of features, namely **BC**, **ECG**, and **BACK**, and much longer on data set **IV-GSE**, which has +12K features. This is entirely with expectations, as π_{RF} and π_{XT} have a built-in mechanism for random feature selection, which can greatly reduce the number of features that need to be considered per tree without harming the accuracy, while π_{SID3T} has to consider all features.

On the flip-side, this random feature selection in π_{RF} and π_{XT} does not come for free, and neither does the random instance selection in π_{RF}. As explained in Section 5 and 6, such oblivious extraction of subsets of the data is realized through secure matrix multiplication. This matrix multiplication only requires a single round of communication, but the local complexity is quite high. Further optimization could be done by offloading the work onto a GPU.

Recall from Section 6 that to compensate for the static discretization per tree, in π_{XT} the parties jointly select features *with replacement*, and choose a different split for each. This explains why *sel. feat.* for XT in the secure protocol results columns in Table 2 can exceed the total amount of features in the data set. This design choice benefits π_{XT} greatly, making it fast and the most competitive with Sklearn's implementation in terms of accuracy. Given our choice to select a pool of random features per tree as opposed to per node, we also save valuable computation time. Take our **BC** results for XT as an example. In Sklearn's implementation, each node must generate 19 unique features. If each of their trees are fully grown, as ours must be to protect from side channel attacks, they would have to generate $(2^5 - 1) \cdot 19 = 589$ total features and cut-off points. In contrast, we generate a pool of 128 features up-front, saving us an immense amount of time, without harming accuracy.

# of elements	10^6	10^7	10^8	10^9
π_{2toQ}	0.1 sec	0.4 sec	3.7 sec	58.1 sec
π_{GEQ}	0.4 sec	2.0 sec	17.0 sec	152.2 sec
π_{minmax}	1.9 sec	7.1 sec	50.1 sec	572.7 sec

Table 3. Runtimes of protocol executions on varying input sizes. π_{GEQ} compares two vectors of size # elements while π_{2toQ} and π_{minmax} perform operations on one vector of size # elements.

	Instances		
Bins	10^3	10^4	10^5
2	2.3 sec	13.6 sec	106.5 sec
3	2.8 sec	19.9 sec	139.6 sec
5	3.3 sec	24.1 sec	199.0 sec
8	9.2 sec	28.8 sec	374.1 sec

Table 4. Runtime of π_{DISC} for a varying number of instances and bins, and a fixed amount of 1,000 features.

Tables 3 and 4 show the time it takes to perform some sub-protocols for multiple input lengths. Table 3 shows the runtimes of π_{2toQ}, π_{GEQ}, and π_{minmax} up to 10^9 elements, where π_{2toQ} and π_{minmax} performed their operations on a single vector of that length, and π_{GEQ} performed operations between two vectors of that length. We stop at 10^9 because that is the point our virtual machines run out of RAM. Table 4 shows the runtimes for π_{DISC} for a varying number of bins and instances, where the amount of features is fixed at 1,000. These runtimes are quite fast compared to previous iterations

and are notably sublinear over certain ranges. This is the result of optimized multithreading and socket-level parallelization.

To experimentally validate our claim that our discretization based approach for training DTs over data with continuous values outperforms existing approaches based on oblivious sorting, we generated an artificial data set with 8192 instances and 11 features, i.e. the same dimensions used in [18]. The runtime to train a DT of depth 4 and 2 bins is 3.51 sec for the online phase with our approach, which is implemented in a trusted initialized model with two computing parties, compared to the estimated 8.27 min for the combined offline and online phase in [18], which is implemented using a 3-party protocol with replicated secret sharing tolerating one corruption.

8 Hyperparameter tuning

An important question in our work is which values to choose for the hyperparameters, such as the number of trees and the number of randomly selected features per tree. We recommend to use related data sets that are publicly available for performing such tuning.

To the best of our knowledge, the problem of selecting hyperparameters in a privacy-preserving way has not been addressed before in the literature. For example, when privately training neural networks, the network topologies, number of neurons, among other parameters, are assumed to be known in advance [2].

Interesting directions for future work are direct adaptations of grid-search by training multiple models with multiple different hyperparameters jointly amongst the parties. Then, in conjunction with protocols for privacy-preserving inference [8, 22], we could perform a secure comparison of all of the results, obliviously select the most accurate model with the best hyperparameters, and use that to classify new, unseen instances.

We consider that the proposal and analysis of such protocols for the selection of parameters in a privacy-preserving way is an important future research direction, but outside the scope of our paper.

9 Conclusion

In this paper we have presented several cryptographic protocols that enable two or more parties to train decision trees or decision tree ensemble classifiers over their joint continuous valued data, while keeping the data private, i.e. neither of the parties has to show their data to anyone in the clear. Our results demonstrate that it is possible to efficiently train tree-based models with good accuracy while completely avoiding a full private sorting, by performing various forms of discretization in a privacy-preserving manner instead.

The results of our protocols demonstrate competitive accuracy with their state-of-the-art, in-the-clear counterparts. Beyond accuracy, each one of our protocols demonstrated high efficiency in terms of runtime, and is fast enough to be used in practice. Our MPC protocol for extra-trees classifiers is the clear winner among our three secure approaches for training tree based models on continuous valued data, as it is both accurate and fast. When confronted with the need to securely train a tree-based model over data that is distributed across different parties, we therefore recommend protocol π_{XT} as the method of choice.

The accuracy of machine learning algorithms can be highly dependent on specific properties of data sets. Thus, it is important to have a portfolio of privacy-preserving machine-learning algorithms readily available for use. The presented protocols fill an important gap in the literature, presenting the first protocols for trees and tree ensembles that can handle continuous data without relying on a full sorting of the data set.

Acknowledgements

Funding support for project activities has been provided by The University of Washington Tacoma's Founders Endowment fund. The authors would like to thank Microsoft for the donation of cloud computing credits through the UW Azure Cloud Computing Credits for Research program.

References

[1] R. Cramer, I. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[2] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE Symposium on Security and Privacy (SP)*, pp. 19–38, 2017.

[3] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. on Privacy Enhancing Technologies*, no. 3, pp. 26–49, 2019.

[4] M. De Cock, R. Dowsley, A. C. A. Nascimento, D. Railsback, J. Shen, and A. Todoki, "High performance logistic regression for privacy-preserving genome analysis," *BMC Medical Genomics*, vol. 14(23), 2021.

[5] C. Guo, A. Hannun, B. Knott, L. van der Maaten, M. Tygert, and R. Zhu, "Secure multiparty computations in floating-point arithmetic," *arXiv:2001.03192*, 2020.

[6] T. G. Dietterich, "Ensemble methods in machine learning," in *International Workshop on Multiple Classifier Systems*, vol. 1857 of *LNCS*, pp. 1–15, Springer, 2000.

[7] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests.," *Proc. on Privacy Enhancing Technologies*, no. 4, pp. 335–355, 2016.

[8] K. Fritchman, K. Saminathan, R. Dowsley, T. Hughes, M. De Cock, A. Nascimento, and A. Teredesai, "Privacy-preserving scoring of tree ensembles: A novel framework for AI in healthcare," in *IEEE Big Data*, pp. 2413–2422, 2018.

[9] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, "Sok: Modular and efficient private decision tree evaluation," *Proc. on Privacy Enhancing Technologies*, no. 2, p. 187–208, 2019.

[10] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[11] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Annual International Cryptology Conf.*, pp. 36–54, 2000.

[12] J. Vaidya and C. Clifton, "Privacy-preserving decision trees over vertically partitioned data," in *IFIP Annual Conf. on Data and Appl. Security and Privacy*, pp. 139–152, 2005.

[13] S. Samet and A. Miri, "Privacy preserving ID3 using Gini index over horizontally partitioned data," in *2008 IEEE/ACS Intern. Conf. on Comp. Syst. and Appl.*, pp. 645–651, 2008.

[14] S. de Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, "Practical secure decision tree learning in a teletreatment application," in *Intern. Conf. on Financial Cryptography and Data Security*, pp. 179–194, Springer, 2014.

[15] M.-J. Xiao, K. Han, L.-S. Huang, and J.-Y. Li, "Privacy preserving C4.5 algorithm over horizontally partitioned data," in *Fifth International Conference on Grid and Cooperative Computing (GCC'06)*, pp. 78–85, IEEE, 2006.

[16] Y. Shen, H. Shao, and L. Yang, "Privacy preserving C4.5 algorithm over vertically distributed datasets," in *Intern. Conf. on Networks Security, Wireless Communications and Trusted Computing*, vol. 2, pp. 446–448, IEEE, 2009.

[17] G. Behera, "Privacy preserving C4.5 using Gini index," in *2nd National Conference on Emerging Trends and Applications in Computer Science*, pp. 1–4, 2011.

[18] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," in *Proc. on Privacy Enhancing Technologies*, no. 1, pp. 167–187, 2021.

[19] J. R. Quinlan, *C4.5: programs for machine learning*. Elsevier, 2014.

[20] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[21] K. Deforth, M. Desgroseilliers, N. Gama, M. Georgieva, D. Jetchev, and M. Vuille, "XORBoost: Tree boosting in the multiparty computation setting." Cryptology ePrint Archive, Report 2021/432, 2021. https://eprint.iacr.org/2021/432.

[22] M. De Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W.-S. Poon, and S. Truex, "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 217–230, 2019.

[23] D. Beaver, "Commodity-based cryptography," in *STOC*, vol. 97, pp. 446–455, 1997.

[24] B. David, R. Dowsley, R. Katti, and A. C. Nascimento, "Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols," in *International Conference on Provable Security*, pp. 354–367, Springer, 2015.

[25] M. De Cock, R. Dowsley, A. C. A. Nascimento, and S. C. Newman, "Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data," in *8th ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 3–14, 2015.

[26] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science*, pp. 136–145, IEEE Computer Society, 2001.

[27] D. Reich, A. Todoki, R. Dowsley, M. De Cock, and A. Nascimento, "Privacy-preserving classification of personal text messages with secure multi-party computation," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3752–3764, 2019.

[28] N. Attrapadung, G. Hanaoka, S. Kiyomoto, T. Mimoto, and J. C. N. Schuldt, "A taxonomy of secure two-party comparison protocols and efficient constructions," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 102-A, no. 9, pp. 1048–1060, 2019.

[29] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*, pp. 192–206, Springer, 2008.

[30] T. Nishide and K. Ohta, "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol," in *International Workshop on Public Key Cryptography*, pp. 343–360, Springer, 2007.

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

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] S. Garcia, J. Luengo, J. A. Sáez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 734–750, 2012.

[34] R. Dowsley, *Cryptography Based on Correlated Data: Foundations and Practice*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2016.

A Security model and proofs

The security model considered in this work is the Universal Composability (UC) framework [26], the gold standard for formally defining and analyzing the security of cryptographic protocols. Any protocol that is proven UC-secure, can be arbitrarily composed with other copies of itself and of other protocols (even with arbitrarily concurrent executions) while preserving security. That is an extremely useful property that allows the modular design of cryptographic protocols. UC-security is also a necessity for cryptographic protocols running in complex environments such as the Internet. This appendix only gives a short overview of the UC framework for the specific case of protocols with two participants (denoted Alice and Bob). See [1] for more details.

In the UC framework the security is analyzed by comparing a real world with an ideal world. In the real world Alice and Bob interact between themselves and with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. The environment $\mathcal{Z}$ captures all external activities to the protocol instance under consideration, and is responsible for giving the inputs and getting the outputs from Alice and Bob. The adversary $\mathcal{A}$ can corrupt either Alice or Bob, in which case he gains the control over that participant. The network scheduling is assumed to be adversarial and thus $\mathcal{A}$ is responsible for delivering the messages between Alice and Bob. In the ideal world, there is an ideal functionality $\mathcal{F}$ that captures the perfect specification of the desired outcome of the computation. $\mathcal{F}$ receives the inputs directly from Alice and Bob, performs the computations locally following the primitive specification and delivers the outputs directly to Alice and Bob. A protocol π executed between Alice and Bob in the real world UC-realizes the ideal functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between: (1) an execution of the protocol π in the real world with participants Alice and Bob, and adversary $\mathcal{A}$; (2) and an ideal execution with dummy parties (that only forward inputs/outputs), $\mathcal{F}$ and $\mathcal{S}$.

We design our protocols in the trusted initializer (TI) model, which is formalized by the trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$. The TI pre-distributes correlated randomness to Alice and Bob, but neither takes part in any part of the protocol execution nor learns any inputs or outputs of Alice and Bob.

Functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$

$\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ is parameterized by algorithm $\mathcal{D}$ that samples correlated randomness. Upon initialization, run $(D_A, D_B) \xleftarrow{\$} \mathcal{D}$, and deliver D_A to Alice and D_B to Bob.

Simplifications: The messages of ideal functionalities are formally public delayed outputs, meaning that $\mathcal{S}$ is first asked whether they should be delivered or not (this is due to the modeling that the adversary controls the network scheduling). This detail as well as the session identifications are omitted from the description of our functionalities for the sake of readability.

Our protocols are information-theoretically secure and the simulation strategy is quite simple: all the messages look uniformly random from the recipient's point of view, except for the messages that open a secret shared value to a party, but these ones can be easily simulated using the output of the respective functionalities. A simulator $\mathcal{S}$, having the leverage of being able to simulate the ideal functionalities that capture the trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ and the ideal functionalities that are UC-realized by the sub-protocols, can easily extract the necessary values and perform a perfect simulation of a real protocol execution; therefore making the real and ideal worlds indistinguishable for any environment $\mathcal{Z}$. The simulation strategy will be described briefly in our proofs.

The protocol for secure matrix multiplication π_{DMM} UC-realizes the distributed matrix multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ [22, 34] in the trusted initializer model.

Functionality $\mathcal{F}_{\mathsf{DMM}}$

$\mathcal{F}_{\mathsf{DMM}}$ is parameterized by the size q of the ring $\mathbb{Z}_q$ and the dimensions (i, j) and (j, k) of the matrices.

Input: Upon receiving a message from Alice/Bob with its shares of $[\![X]\!]_q$ and $[\![Y]\!]_q$, verify if the share of X is in $\mathbb{Z}_q^{i \times j}$ and the share of Y is in $\mathbb{Z}_q^{j \times k}$. If it is not, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other party about the receipt.

Output: Upon receipt of the shares from both parties, reconstruct X and Y from the shares, compute $Z = XY$ and create a secret sharing $[\![Z]\!]_q$ to distribute to Alice and Bob: a corrupt party fixes its share of the output to any chosen matrix and the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

The protocol π_{2toQ} for converting a secret bit from a $\mathbb{Z}_2$ sharing to a $\mathbb{Z}_q$ sharing UC-realizes the share conversion functionality $\mathcal{F}_{\mathsf{2toQ}}$ [27].

Functionality $\mathcal{F}_{\mathsf{2toQ}}$

$\mathcal{F}_{\mathsf{2toQ}}$ is parameterized by the size of the field q.

Input: Upon receiving a message from Alice/Bob with her/his share of $[\![x]\!]_2$, record the share, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct x, then create and distribute to Alice and Bob the secret sharing $[\![x]\!]_q$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the uncorrupted parties' shares are then created by picking uniformly random values subject to the correctness constraint.

The bit extraction protocol π_{BTX} is a straightforward simplification of the bit decomposition protocol $\pi_{\mathsf{decompOPT}}$ from [4] and UC-realizes the bit extraction functionality $\mathcal{F}_{\mathsf{BTX}}$.

Functionality $\mathcal{F}_{\mathsf{BTX}}$

$\mathcal{F}_{\mathsf{BTX}}$ is parameterized by the bit-length λ of the secret shared input value x and extracts the α-th bit from the secret shared value.

Input: Upon receiving a message from Alice or Bob with its share of $[\![x]\!]$, record the share, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the value $x = x_\lambda \cdots x_1$ from the shares, and distribute a new secret sharing $[\![x_\alpha]\!]_2$ of the bit x_α. Before the output delivery, the corrupt party fixes its shares of the output to any desired value. The shares of the uncorrupted parties are then chosen uniformly at random subject to the correctness constraints.

The secure comparison π_{GEQ} protocol is trivially correct. The simulator $\mathcal{S}$ internally simulates an execution of Protocol π_{GEQ} for the adversary $\mathcal{A}$ that controls the corrupted party. Using the fact that he is the one simulating the trusted initializer in this execution, $\mathcal{S}$ can extract the shares of the inputs x and y that belong to the corrupted party and forward them to $\mathcal{F}_{\mathsf{GEQ}}$. $\mathcal{S}$ can then fix in $\mathcal{F}_{\mathsf{GEQ}}$ the corrupted party's share of the output to the value that matches what $\mathcal{A}$ gets in the simulated execution of π_{GEQ}. The simulation is perfect, and therefore the environment $\mathcal{Z}$ cannot distinguish the real and ideal worlds, and π_{GEQ} UC-realizes $\mathcal{F}_{\mathsf{GEQ}}$.

Functionality $\mathcal{F}_{\mathsf{GEQ}}$

$\mathcal{F}_{\mathsf{GEQ}}$ runs with Alice and Bob and is parameterized by the bit-length λ of the values x, y to be compared. x and y are guaranteed to be such that $|x - y| < 2^{\lambda-1}$.

Input: Upon receiving a message from Alice or Bob with its share of $[\![x]\!]$ and $[\![y]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the values x and y. If $x \geq y$, distribute a new secret sharing $[\![1]\!]_2$; otherwise a new secret sharing $[\![0]\!]_2$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The uncorrupted parties' shares are created by picking uniformly random values subject to the correctness constraints.

The secure equality π_{EQ} protocol is also trivially correct and the simulator for $\mathcal{F}_{\mathsf{EQ}}$ uses a similar strategy as the one for $\mathcal{F}_{\mathsf{GEQ}}$ to get a perfect simulation. Thus π_{EQ} UC-realizes $\mathcal{F}_{\mathsf{EQ}}$.

Functionality $\mathcal{F}_{\mathsf{EQ}}$

$\mathcal{F}_{\mathsf{EQ}}$ runs with Alice and Bob and is parameterized by the bit-length λ of the values x, y to be compared. x and y are guaranteed to be such that $|x - y| < 2^{\lambda-1}$.

Input: Upon receiving a message from Alice or Bob with its share of $[\![x]\!]$ and $[\![y]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the values x and y. If $x = y$, distribute a new secret sharing $[\![1]\!]_2$; otherwise a new secret sharing $[\![0]\!]_2$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The uncorrupted parties' shares are created by picking uniformly random values subject to the correctness constraints.

It is trivial to verify that the protocol π_{minmax} correctly outputs secret sharings corresponding to the minimum and maximum values of the input vector. The simulator $\mathcal{S}$ uses the fact that he is responsible for simulating $\mathcal{F}_{\mathsf{GEQ}}$ in the internal simulated execution of π_{minmax} with $\mathcal{A}$ in order to extract the corrupted party's shares of $d_1, \ldots, d_n$. $\mathcal{S}$ can then forward those input shares to $\mathcal{F}_{\mathsf{minmax}}$ and adjust the shares of the outputs d_{min} and d_{max} in the functionality to match what $\mathcal{A}$ sees in the simulated execution of π_{minmax}. The simulation is perfect and π_{minmax} UC-realizes $\mathcal{F}_{\mathsf{minmax}}$.

Functionality $\mathcal{F}_{\mathsf{minmax}}$

$\mathcal{F}_{\mathsf{minmax}}$ is parameterized by the size n of the vector.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![d_1]\!], \ldots, [\![d_n]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the values $d_1, \ldots, d_n$. Compute the minimum d_{min} and maximum d_{max} values contained in the vector. Distribute new secret sharings $[\![d_{min}]\!]$ and $[\![d_{max}]\!]$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The uncorrupted parties' shares are created by picking uniformly random values subject to the correctness constraints.

It is easy to verify that the secure discretization protocol π_{DISC} correctly computes the bins thresholds and the one-hot encodings of the bin membership of each $d \in D$. The simulator $\mathcal{S}$ for $\mathcal{F}_{\mathsf{DISC}}$ internally runs an execution of π_{DISC} for $\mathcal{A}$ and uses the fact that he simulates $\mathcal{F}_{\mathsf{minmax}}$ in order to extract the corrupted party's shares of the values $d_1, \ldots, d_n$ that he needs to forward to $\mathcal{F}_{\mathsf{DISC}}$. It can then trivially adjust the corrupted party's output shares in $\mathcal{F}_{\mathsf{DISC}}$ to matches the ones in the simulated execution of π_{DISC}. This is a perfect simulation strategy and π_{DISC} UC-realizes $\mathcal{F}_{\mathsf{DISC}}$.

Functionality $\mathcal{F}_{\mathsf{DISC}}$

$\mathcal{F}_{\mathsf{DISC}}$ is parameterized by the number of elements n in the vector D and the number of bins p.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![D]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the values $d_1, \ldots, d_n$. Compute the minimum d_{min} and maximum d_{max} values contained in the vector, and the thresholds of the p equal-width bins. Map the values $d \in D$ to the respective bins and create D' that contains one-hot-encodings of the bin membership of each $d \in D$. Distribute $[\![D']\!]_2$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

We also use as a building block a protocol π_{SID3T} that UC-realizes a decision tree training functionality $\mathcal{F}_{\mathsf{SID3T}}$. In particular, we use a slightly modified version of the protocol by De Hoogh et al. [14].

Functionality $\mathcal{F}_{\mathsf{SID3T}}$

$\mathcal{F}_{\mathsf{SID3T}}$ runs with Alice and Bob to train a decision tree model with categorical values. It is parameterized by the number of samples n and of features f in the data set S, the number p of categories per feature, and the depth d of each tree. The input data is presented in a one-hot-encoding (OHE) format and the Gini index is used to select the split features.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![S]\!]_2$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the data set S and locally train a decision tree t using the same criteria and representation format as π_{SID3T}. Distribute $[\![t]\!]$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

When we concatenate the protocols π_{DISC} and π_{SID3T}, they do not leak any information due to the UC-security of their building blocks. This concatenated protocol UC-realizes $\mathcal{F}_{\mathsf{DISC+DT}}$. The simulator $\mathcal{S}$ can easily extract the corrupted party's shares of the data set by using the fact that it simulates $\mathcal{F}_{\mathsf{DISC}}$ in the internal simulated execution with the adversary $\mathcal{A}$. $\mathcal{S}$ can provide this information to $\mathcal{F}_{\mathsf{DISC+DT}}$ and adjust the outputs of $\mathcal{F}_{\mathsf{DISC+DT}}$ to match the internal execution with $\mathcal{A}$.

Functionality $\mathcal{F}_{\mathsf{DISC+DT}}$

$\mathcal{F}_{\mathsf{DISC+DT}}$ runs with Alice and Bob and first performs a discretization of the continuous-valued data set S and then trains a decision tree on the categorical data. It is parameterized by the number of samples n and of features f in the data set S, the number of bins p per feature and the depth d of each tree.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![S]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, compute locally the discretization of S followed by the secret shared tree $[\![t]\!]$ using the same procedures as π_{DISC} followed by π_{SID3T} (the necessary correlated randomness is generated locally). Distribute $[\![t]\!]$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

The protocol π_{RF} for training a random forest on data that is initially discretized clearly does not leak any information due to the UC-security of its building blocks. The simulator $\mathcal{S}$ can easily extract the corrupted party's shares of the data set by using the fact that it simulates $\mathcal{F}_{\mathsf{DISC}}$ in the internal simulated execution of

π_{RF} with the adversary $\mathcal{A}$. $\mathcal{S}$ can provide this information to $\mathcal{F}_{\mathsf{DISC+RF}}$ and adjust the outputs of $\mathcal{F}_{\mathsf{DISC+RF}}$ to match the internal execution of π_{RF}, thus π_{RF} UC-realizes $\mathcal{F}_{\mathsf{DISC+RF}}$.

Functionality $\mathcal{F}_{\mathsf{DISC+RF}}$

$\mathcal{F}_{\mathsf{DISC+RF}}$ runs with Alice and Bob and first performs a discretization of the continuous-valued data set S and then trains a random forest on the categorical data. It is parameterized by the number of samples n and of features f in the data set S, the number of bins p per feature, the number k of features to use in each tree, the number of samples s used in each tree, the number m of trees in the ensemble and the trees' depth d.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![S]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, compute locally the discretization of S followed by the secret shared trees $[\![t_1]\!], \ldots, [\![t_m]\!]$ using the same procedures as π_{RF} (the necessary correlated randomness is generated locally). Distribute $[\![RF]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The uncorrupted parties' shares are created by picking uniformly random values subject to the correctness constraints.

The protocol π_{XT} for training an extra-trees model clearly does not leak any information due to the UC-security of its building blocks. The simulator $\mathcal{S}$ can easily extract the corrupted party's shares of the data set by using the fact that it simulates $\mathcal{F}_{\mathsf{DMM}}$ in the internal simulated execution of π_{XT} with the adversary $\mathcal{A}$. $\mathcal{S}$ can provide this information to $\mathcal{F}_{\mathsf{XT}}$ and adjust the outputs of $\mathcal{F}_{\mathsf{XT}}$ to match the internal execution of π_{XT}, therefore π_{XT} UC-realizes $\mathcal{F}_{\mathsf{XT}}$.

Functionality $\mathcal{F}_{\mathsf{XT}}$

$\mathcal{F}_{\mathsf{XT}}$ runs with Alice and Bob to train an extra-trees model. It is parameterized by the number of samples n and of features f in the data set S, the number k of features to use in each tree, the number m of trees in the ensemble and the depth d of each tree.

Input: Upon receiving a message from Alice or Bob with its shares of $[\![S]\!]$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, compute the secret shared trees $[\![t_1]\!], \ldots, [\![t_m]\!]$ locally using the same procedures as π_{XT} (the necessary correlated randomness is generated locally). Distribute $[\![XT]\!] = [\![t_1]\!], \ldots, [\![t_m]\!]$. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The uncorrupted parties' shares are chosen uniformly at random subject to the correctness constraints.

B Efficient bit extraction

The protocol for secure bit extraction, π_{BTX}, is an of the matrix composition network approach to bit decomposition presented in [4] as $\pi_{\mathsf{decompOPT}}$. Here, we summarize the notions used to derive $\pi_{\mathsf{decompOPT}}$ and then derive our modification π_{BTX}.

The decomposition of a secret-shared value $[\![x]\!]_{2^\lambda} = a + b \mod 2^\lambda$ into bitwise shares in $\mathbb{Z}_2$ is modeled as an adder circuit where, beginning with the least significant bit of a, b, all subsequent bits are computed as a bitwise sum plus the carry-over from the previous bitwise sum. Formally, we consider two signals which depend on a, b: Generate ($g_i = a_i b_i$) generates a carry bit at the i-th position, and Propagate ($p_i = a_i \oplus b_i$) propagates a carry bit, if it exists [4]. Then, the i-th bit of the sum is given by $s_i = p_i \oplus c_{i-1}$ where the i-th *carry bit* is given by $c_i = g_i + p_i c_{i-1}$.

It was noted in [4] that a matrix representing this formula for the carry bit can be computed efficiently in the MPC setting with minimal dependency on the results for previous carry bits. Moreover, all carry bits can be computed in advance by composing entries of the matrix representation. Let M_i denote the matrix that holds the i-th Generate and Propagate bits.

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix} = M_i \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix}.$$

Then, all carry bits c_i can be derived by the product $M_i \cdot (c_{i-1}, 1)$, but they can be equally be computed by the composition of all matrices $M_i M_{i-1} \ldots M_1 \cdot (0, 1)$. Further, by noting that the product

$$M_i M_{i-1} \ldots M_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p'_i & g'_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} g'_i \\ 1 \end{bmatrix},$$

the i-th carry bit c_i is implicitly the upper right-hand entry of the composed matrix, g'_i. Given that matrix composition is left-associative, the set of all matrices needed for the final result can be computed in a log depth circuit – hereafter $\mathsf{ComposeNet}_\lambda$ – by, at the i-th layer, computing all compositions $M_{1.j}$ that require fewer than 2^{i-1} composition operations. The constraint is added that each $M_{1.j}$ should be the composition of the "largest" matrix from the previous layer, $M_{1.2^{i-2}}$,

Protocol 10: Secure optimized bit decomposition protocol $\pi_{\mathsf{decompOPT}}$.

Input : $[\![x]\!]_{2^\lambda}$
Output: $[\![x_1]\!]_2 ... [\![x_\lambda]\!]_2$

Party i regards its share x_i as $p_{i,1}, \ldots, p_{i,\lambda}$ s.t.
$[\![p_j]\!]_2, = p_{1,j} \oplus p_{2,j}$ for $j = 1, \ldots, \lambda$
Party 1 creates the sharing $[\![g_{1,j}]\!]_2 = (p_{1,j}, 0)$.
Party 2 creates the sharing $[\![g_{2,j}]\!]_2 = (0, p_{2,j})$.
$[\![g_j]\!]_2 \leftarrow [\![g_{1,j}]\!]_2 [\![g_{2,j}]\!]_2$
$[\![M_j]\!]_2 \leftarrow \begin{bmatrix} [\![p_j]\!]_2 & [\![g_j]\!]_2 \\ 0 & 1 \end{bmatrix}$ for all j
$\{[\![M_{1.j}]\!]_2 \mid 1 \le j < \lambda\} \leftarrow \mathsf{ComposeNet}_\lambda([\![M]\!]_2)$
$[\![c_j]\!]_2 \leftarrow$ the upper right entry of $[\![M_{1.j}]\!]_2$
$[\![s_1]\!]_2 \leftarrow [\![p_1]\!]_2$
$[\![s_j]\!]_2 \leftarrow [\![p_j]\!]_2 \oplus [\![c_{j-1}]\!]_2$ for all $j > 1$
return $[\![s_1]\!]_2 ... [\![s_\lambda]\!]_2$

Protocol 11: Secure protocol π_{BTX} extracts the α-th bit from a secret shared value.

Input : $[\![x]\!]_{2^\lambda}$, α
Output: $[\![x_\alpha]\!]_2$

Party i regards the lowest α bits of its share, x_i, as $p_{i,1}, \ldots, p_{i,\alpha}$ s.t.
$[\![p_j]\!]_2, = p_{1,j} \oplus p_{2,j}$ for $j = 1, \ldots, \alpha$
Party 1 creates the sharing $[\![g_{1,j}]\!]_2 = (p_{1,j}, 0)$.
Party 2 creates the sharing $[\![g_{2,j}]\!]_2 = (0, p_{2,j})$.
$[\![g_j]\!]_2 \leftarrow [\![g_{1,j}]\!]_2 [\![g_{2,j}]\!]_2$
$[\![M_j]\!]_2 \leftarrow \begin{bmatrix} [\![p_j]\!]_2 & [\![g_j]\!]_2 \\ 0 & 1 \end{bmatrix}$ for $j = 1, ..., \alpha - 1$
$[\![M_{1.(\alpha-1)}]\!]_2 \leftarrow \mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha([\![M]\!]_2)$
$[\![c_{\alpha-1}]\!]_2 \leftarrow$ the upper right entry of $[\![M_{1.(\alpha-1)}]\!]_2$
return $[\![p_\alpha]\!]_2 \oplus [\![c_{\alpha-1}]\!]_2$

with the remainder $M_{2^{i-2}+1.j}$. If $M_{2^{i-2}+1.j}$ doesn't exist in the network, it is added recursively [4]. For a ring element with bit length λ, the depth of $\mathsf{ComposeNet}_\lambda$ is $\lceil \log(\lambda - 1) \rceil$. This is because the λ-th sum bit depends only on $c_{\lambda-1}$. An example of $\mathsf{ComposeNet}_\lambda$ for $\lambda = 9$ is given in Figure 2. The optimized bit decomposition protocol $\pi_{\mathsf{decompOPT}}$ of [4] is described in Protocol 10.

A key aspect of $\mathsf{ComposeNet}_\lambda$ is that the matrix composition $M_{1.j}$, which is used to derive c_j, depends only on $\lceil \log(j) \rceil$ rounds of matrix composition. Moreover, given that $M_{1.j}$ depends on a small subset of branches in $\mathsf{ComposeNet}_\lambda$, it is possible to reduce the circuit significantly to extract only one bit. Call the reduced circuit that extracts the α-th bit $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$. This circuit can be constructed by removing nodes from $\mathsf{ComposeNet}_\lambda$ that are not parents of $M_{1.(\alpha-1)}$. In an alternative view, it can be constructed iteratively by, at each layer, computing the pairwise matrix compositions of all nodes until the final layer contains only $M_{1.(\alpha-1)}$.

The bit extraction protocol π_{BTX}, described in Protocol 11, follows exactly the same structure as $\pi_{\mathsf{decompOPT}}$. The only difference is that the reduced matrix composition network $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$ is computed in lieu of $\mathsf{ComposeNet}_\lambda$ and the depth of the circuit depends only on $\lceil \log(\alpha - 1) \rceil$. An example of $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$ for $\alpha = 9$ is given in Figure 2.

Efficiency discussion: Prior to computing $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$, there is one round of communication required to compute all g_j. In total, this step requires $\alpha - 1$ $\mathbb{Z}_2$ multiplications and a total data transfer of $2(\alpha - 1)$ bits. Computing $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$ takes $\lceil \log(\alpha - 1) \rceil$ rounds of communication and, for $\alpha - 1 = 2^k$, requires $\sum_{i=0}^{\log(\alpha-1)-1} 2^i = \alpha - 2$ matrix compositions, each with 4 bits of data transfer. The complexity differs by one matrix composition for $\alpha - 1 \neq 2^k$. All other operations are local and thus add virtually nothing to the overall running time. In short, the round complexity of π_{BTX} is $\lceil \log(\alpha - 1) \rceil + 1$ and it transfers $6\alpha - 10$ bits of data.

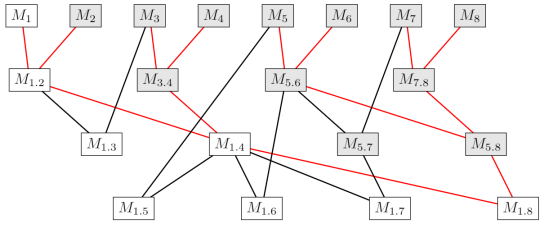

Fig. 2. $\mathsf{ComposeNet}_\lambda$ for $\lambda = 9$, with all matrix compositions $M_1, M_{1.2}, M_{1.3}, \ldots, M_{1.(\lambda-1)}$ [4]. The red subgraph computes $\mathsf{ComposeNet}^{\mathsf{BTX}}_\alpha$ for $\alpha = 9$; the matrix composition $M_{1.(\alpha-1)}$.