Two-Party Decision Tree Training from Updatable Order-Revealing Encryption*

Robin Berger[1], Felix Dörre[1], and Alexander Koch[2]

[1] KASTEL Security Research Labs, Karlsruhe Institute of Technology, Karlsruhe, Germany robin.berger@kit.edu, felix.doerre@kit.edu
[2] CNRS and IRIF, Université Paris Cité, Paris, France alexander.koch@irif.fr

Abstract. Running machine learning algorithms on encrypted data is a way forward to marry functionality needs common in industry with the important concerns for privacy when working with potentially sensitive data. While there is already a variety of protocols in this setting based on fully homomorphic encryption or secure multiparty computation (MPC), we are the first to propose a protocol that makes use of a specialized Order-Revealing Encryption scheme. This scheme allows to do secure comparisons on ciphertexts and update these ciphertexts to be encryptions of the same plaintexts but under a new key. We call this notion Updatable Order-Revealing Encryption (uORE) and provide a secure construction using a key-homomorphic pseudorandom function.
In a second step, we use this scheme to construct an efficient three-round protocol between two parties to compute a decision tree (or forest) on labeled data provided by both parties. The protocol is in the passively-secure setting and has some leakage on the data that arises from the comparison function on the ciphertexts. We motivate how our protocol can be compiled into an actively-secure protocol with less leakage using secure enclaves, in a graceful degradation manner, e.g. falling back to the uORE leakage, if the enclave becomes fully transparent. We also analyze the leakage of this approach, giving an upper bound on the leaked information. Analyzing the performance of our protocol shows that this approach allows us to be much more efficient (especially w.r.t. the number of rounds) than current MPC-based approaches, hence allowing for an interesting trade-off between security and performance.

Keywords: Secure Computation, Order-Revealing Encryption, Decision Tree Learning, Enclaves, Privacy-Preserving Machine Learning

1 Introduction

Privacy-preserving machine learning has gained a lot of traction in recent years, due to the tremendous benefits of having automated data-driven decision making, while caring for the legitimate privacy interests of the user, especially in a

* This version of the contribution has been accepted for publication, after peer review but is not the Version of Record. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-54770-6_12

multiparty setting. More importantly, due to legal requirements, many uses of machine learning (ML) algorithms would not even be possible in a setting where access to sensitive information (such as medical data) is required.

One ML algorithm that is relatively popular, due to its simplicity and interpretability, is decision tree learning. It is usually employed in the more general decision forest version. Here, we use a training data set of entries with many attributes and a label, to build a decision tree. This tree branches based on thresholds w.r.t. the attribute values, and has labels annotated to its leaf nodes. To classify a new entry, one follows the tree from the root to a leaf node by comparing the attributes against the thresholds. The classification result is the label annotated to the reached leaf node.

Training such trees via secure multiparty computation has already been proposed by [2, 16]. Because this is in a strong privacy model, these protocols are relatively expensive regarding computation and round complexity. The overall round complexity of [16] is $\mathcal{O}(h(\log m + \log n))$, where h is the a priori fixed depth of the tree, m is the number of attributes and n is the number of data entries. Hence, in the setting where they add a realistic 50 ms delay to each message on the network, the overall running time of the protocols increases by several orders of magnitude.

This motivates the following research question: Can we greatly improve the performance and round complexity of the protocol by allowing for some leakage. (This leakage can later be partly avoided again, with the help of secure enclaves.) Ideally, we would want a small constant number of rounds.

As comparisons or sorting are the main ingredients to Decision Tree training algorithms, we propose to use an extended version of Order-Revealing Encryption (ORE). In ORE schemes, the values are encrypted using a secret key, but given two ciphertexts, one can evaluate the order of the messages they encrypt without knowing this key. We extend this by a way to update the ciphertexts to a new key, in a setting where the key space is a multiplicative group. Hence, given a ciphertext $c = \mathsf{Enc}(k, m)$ and second key k', one can run $\mathsf{Upd}(k', c)$ to obtain a new ciphertext c' that is equal to $\mathsf{Enc}(k \cdot k', m)$, making use of key-homomorphic pseudorandom functions (PRFs).

Using this new primitive, which we believe to be of independent interest, we construct a conceptually simple protocol, using only three rounds, that allows two parties to jointly compute a decision tree on their data. Here, the two parties A and B both have a horizontal partition of the data set and B does the main tree computation. In a nutshell this works as follows:

- To have all data points ORE-encrypted under the same keys (using one key per attribute), the parties proceed in a Diffie-Helman key exchange-like manner: B encrypts his data under his keys k_j. A updates these ciphertexts with her keys k'_j, obtaining ciphertexts under $k_j \cdot k'_j$. B then updates these ciphertexts using k_j^{-1}, obtaining ciphertexts under keys k'_j.
- A also sends her own data points encrypted under k'_j to B, together with the used labels (outcome attributes).

– Now, B has all the data points under the same keys k'_j, and can use the comparison function and the labels to compute a decision tree that uses the encrypted values as thresholds.
– B sends this encrypted tree to A, who can now decrypt it using her keys k'_j.

This is a three-round protocol in the honest-but-curious setting, and hence much faster than the MPC protocols of Hamada et al. [16] and Abspoel, Escudero, and Volgushev [2], if one assumes plausible latency. However, this also comes at the cost of considerable overall leakage due to what can be inferred from the comparisons (and the leakage of the ORE scheme), if the scheme is used in the bare (non-enclave) version. We give a more general analysis on the leakage, as well as an information-theoretic upper bound thereof in Section 5.

1.1 Related Work

Order-Revealing Encryption Order-Revealing Encryption (ORE) was introduced by Boneh et al. [5] as a more flexible and more secure notion of Order-Preserving Encryption (OPE). In contrast to OPE, where the natural ordering of ciphertexts must be identical to the natural ordering of the messages they encrypt, Order-Revealing Encryption allows to define a dedicated comparison function on the ciphertexts for evaluating the natural order of the elements contained within the ciphertexts. The main motivation for this was to enable efficient search operations in encrypted databases. Following the introduction of ORE, Chenette et al. [10] formalized security of ORE schemes, as well as giving a construction of such a scheme. This construction inspired a new scheme by Lewi and Wu [23], which is in a slightly different setting, namely the left–right framework, where there are "left" ciphertexts and "right" ciphertexts and a left ciphertext can be compared only with a right ciphertext. To allow for ORE schemes to be used in a multi-user setting, Li, Wang, and Zhao [24] introduced the notion of delegatable ORE schemes, where it is possible to issue comparison tokens, which allows one to compare ciphertexts of different users. In a similar way, Lv et al. [27] extend ORE schemes to a multi-user setting. One problem with this approach, however, is that if one party has a comparison token allowing another party's ciphertexts to be compared to her own, it can decrypt the other party's ciphertexts again.

As ORE schemes allow comparisons of elements, they inherently leak information about the messages encrypted in ciphertexts, as soon as more than one message is encrypted under the same key. This has led to several investigations on how severely this leakage affects the data privacy. Grubbs et al. [15] and Durak, DuBuisson, and Cash [13] have shown that under some circumstances, ORE schemes provide no meaningful security. Jurado, Palamidessi, and Smith [18] however show that ORE schemes still provide some security if the message space is significantly larger than the amount of messages encrypted.

Privacy-Preserving Machine Learning Since machine learning models have become more widespread, there has been work towards being able to perform the training process thereof in a privacy-preserving manner. Several machine learning models have been considered in this setting, ranging from simple regression tasks

[8, 19] to neural networks. Approaches for the latter include Fully-Homomorphic Encryption (FHE) [22], and MPC protocols [20]. Multiparty computation has also been used for training decision trees, where it is the most prevalent approach since the work by Lindell and Pinkas [25]. In their work, they discuss how MPC protocols can use the ID3 algorithm for training a decision tree. Since then, there have been several improvements over this work [12, 17].

Since the ID3 algorithm only supports discrete attributes, these approaches are not applicable to a setting with with continuous attributes. To overcome this issue, [2] use a variation of the C4.5 algorithm, which also supports continuous attributes. Their protocol works by computing the training process for each possible node in the resulting decision tree. However, this results in the runtime of their protocol being linear in the maximum number of possible nodes in a decision tree, and therefore exponential in the depth of the tree. Hamada et al. [16] improve over this with a protocol, which is linear in the depth of the tree, by partitioning the dataset in an oblivious way and performing the training once for each layer, considering this partitioning. This comes at the cost of requiring many network rounds. While these two state-of-the-art-approaches perform well under ideal circumstances, due to their runtime [2] is not applicable in a setting, where a high-depth decision tree is to be trained and [16] is not applicable in a high-latency environment.

Privacy-Preserving protocols based on fully-homomorphic encryption (FHE) are mostly restricted to decision tree evaluation (e.g. [11, 14]) and do not consider the secure training of decision trees. Most of those papers considering training a decision tree, consider a different setting. For example [3] consider the setting, where their goal is to outsource the training to a server, while some of the computationally intensive steps (for FHE) are still done by the client. Hence, it is not directly comparable to our work. Vaidya et al. [32] consider a setting, where the training data is partitioned vertically and FHE is only used to evaluate a heuristic in the training process, but the remaining computation is done in plain text. So far, there are no efficient decision tree training protocols, that perform the entire training using FHE. This is due to the fact that comparisons are computationally expensive in FHE. Indeed, Liu et al. [26] aim to provide a FHE-bsed protocol allowing comparisons in a multi-user setting, however a single plaintext to ciphertext comparison in their case takes a computation time of 100 ms and 1 s for a ciphertext-ciphertext comparison, rendering this approach infeasable in our setting.

1.2 Our Contribution

Our main contributions are as follows:

- We extend the notion of Order-Revealing Encryption (ORE) to Updatable ORE and give an instantiation thereof.
- Using this Updatable ORE scheme, we construct a three-round two-party protocol to compute a decision tree on a horizontal partitioning of a dataset with both parties providing training data. With this approach, we can apply the same training algorithms as used in plaintext training.

- We describe how this protocol can be combined with enclaves providing different security guarantees, in order to eliminate or reduce the introduced ORE leakage and to make the protocol actively secure. We also use an information-theoretic approach to quantify and give an upper bound for the ORE leakage.
- We implemented and experimentally verified the efficiency of our protocol, showing that it is faster than current state-of-the-art protocols, while achieving this speedup at the cost of some information leakage.

1.3 Outline

We introduce necessary preliminaries including the universal composability (UC) model, ORE and decision tree learning in Section 2. We propose our notion of Updatable Order-Revealing encryption and also present a construction and a security proof in Section 3. Section 4 contains the decision tree learning protocol, together with its security proof, and a remark on how to translate it into an actively-secure version using secure enclaves in a graceful-degradation manner. In Section 5, we discuss the implications of the ORE leakage on the protocol. Finally in Section 6, we evaluate our constructions based on a practical implementation.

2 Preliminaries

In the remainder of this work, PPT refers to a probabilistic Turing Machine with a polynomial runtime bound. Furthermore, $(\mathbb{G}, p, [1])$ is an additive group of prime order $p > 3$. We use the additive implicit notation for group operations with the group generator $[1]$. In implicit notation, $x \cdot [y] = [x \cdot y]$. In this notation, the DDH assumption means that $([1], [x], [y], [x \cdot y])$ and $([1], [x], [y], [z])$ are computationally indistinguishable for $x, y, z \leftarrow \mathbb{Z}_p^\times$. A function $\mathsf{PRF}\colon \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$ is a pseudorandom function (PRF), if oracle access to PRF with a random key $k \leftarrow \mathbb{Z}_p$ is computationally indistinguishable to oracle access to a random function. A PRF is key-homomorphic, if for all messages $m \in \{0,1\}^*$ and keys $k, k' \in \mathbb{Z}_p$

$$\mathsf{PRF}(k, m) + \mathsf{PRF}(k', m) = \mathsf{PRF}(k + k', m)$$

and therefore also

$$a \cdot \mathsf{PRF}(k, m) = \mathsf{PRF}(a \cdot k, m)$$

for all $a \in \mathbb{Z}_p$.

Naor, Pinkas, and Reingold [28] have constructed a key-homomorphic PRF under the DDH assumption in the random oracle model for $\mathsf{RO}\colon \{0,1\}^* \to \mathbb{G} \setminus \{[0]\}$:

$$\mathsf{PRF}(k, m) \coloneqq k \cdot \mathsf{RO}(m) \tag{1}$$

2.1 The Universal Composability Model

The universal composability (UC) model introduced by Canetti [7] is a well established security model for cryptographic protocols. It extends from the real-ideal paradigm, meaning that the security of a protocol is captured by an ideal functionality, that is secure by definition. If a protocol is then shown to be secure, relative to an ideal functionality (in which case we say that the *protocol realizes the ideal functionality*), all security guarantees present in the ideal functionality carry over to the protocol. Protocols secure in the universal composition theorem remain secure under universal composition.

For a more in-depth introduction to the UC framework, see Appendix A or [7].

2.2 Order-Revealing Encryption

We follow the definition in [10]. An *ORE scheme* is defined as a 3-tuple of PPT algorithms $\mathsf{ORE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp})$ over message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$, where

- $\mathsf{Gen}(1^\kappa)$ returns a secret key $k \in \mathcal{K}$
- $\mathsf{Enc}(k, m)$ takes a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ as input and returns a ciphertext $c \in \mathcal{C}$
- $\mathsf{Cmp}(c_0, c_1)$ is deterministic, takes two ciphertexts $ct_0, ct_1 \in \mathcal{C}$ and returns a bit b or $\perp$.

We require correctness for the scheme:

$$\forall m_0, m_1 \in \mathcal{M}, k \leftarrow \mathsf{Gen}(1^\kappa): \\ \Pr\left[\mathsf{Cmp}(\mathsf{Enc}(k, m_0), \mathsf{Enc}(k, m_1)) = 1 \Leftrightarrow m_0 < m_1\right] \geq 1 - \mathrm{negl}(\kappa).$$

In addition, ORE schemes have an implicit $\mathsf{Dec}(k, c)$ algorithm, which takes a secret key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$. Dec is implicitly defined using Enc and Cmp to perform a binary search on the message space and return the result or $\perp$ if the ciphertext is invalid under key k.

Security of ORE Schemes For the security definition of an ORE schemes w.r.t. some leakage function $\mathcal{L}(m_1, \ldots, m_t)$, we use the security notion defined in [10]. Let $\mathsf{ORE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp})$ be an ORE scheme. For some $q = \mathrm{poly}(\kappa)$, let $\mathcal{A}$ be a stateful adversary and let $\mathcal{S}^{\mathsf{ORE}}$ be a stateful simulator. Then, the experiments $\mathsf{REAL}^{\mathsf{ORE}}_{\mathcal{A}}$ and $\mathsf{SIM}^{\mathsf{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}$ are defined as in Fig. 1. We say an ORE scheme ORE is *secure w.r.t. the leakage function* $\mathcal{L}$, if there exists a PPT simulator $\mathcal{S}^{\mathsf{ORE}}$, such that for all PPT adversaries $\mathcal{A}$, the games $\mathsf{REAL}^{\mathsf{ORE}}_{\mathcal{A}}$ and $\mathsf{SIM}^{\mathsf{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}$ are computationally indistinguishable.

Best-Possible Leakage for ORE schemes Naturally, one is interested in an ORE scheme with the least amount of leakage. Given t ORE ciphertexts $c_1, \ldots, c_t \in \mathcal{C}$ of messages $m_1, \ldots, m_t \in \mathcal{M}$ under the same key, then for each pair (c_i, c_j), one can inevitably learn whether $m_i \leq m_j$ by using the comparison algorithm. We

$\mathsf{REAL}^{\mathsf{ORE}}_{\mathcal{A}}(\kappa)$:
1: $k \leftarrow \mathsf{Gen}(1^\kappa)$
2: $m \leftarrow \mathcal{A}(1^\kappa)$
3:
4: $ct_1 \leftarrow \mathsf{Enc}(k, m_1)$
5: **for** $i = 2, \ldots, q$ **do**
6: $\quad m \leftarrow \mathcal{A}(ct_1, \ldots, ct_{i-1})$
7:
8: $\quad ct_i = \mathsf{Enc}(k, m)$
9: **end for**
10: output $(ct_1, \ldots, ct_q)$ and the state of $\mathcal{A}$

$\mathsf{SIM}^{\mathsf{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\kappa)$:
1: $M = \emptyset$
2: $m \leftarrow \mathcal{A}(1^\kappa)$
3: M.append(m)
4: $ct_1 \leftarrow \mathcal{S}^{\mathsf{ORE}}(\mathcal{L}(M))$
5: **for** $i = 2, \ldots, q$ **do**
6: $\quad m \leftarrow \mathcal{A}(ct_1, \ldots, ct_{i-1})$
7: $\quad M$.append(m)
8: $\quad ct_i \leftarrow \mathcal{S}^{\mathsf{ORE}}(\mathcal{L}(M))$
9: **end for**
10: output $(ct_1, \ldots, ct_q)$ and the state of $\mathcal{A}$

Fig. 1: Definition of experiments $\mathsf{REAL}^{\mathsf{ORE}}_{\mathcal{A}}$ and $\mathsf{SIM}^{\mathsf{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}$ for ORE scheme ORE and stateful TMs $\mathcal{A}$ and $\mathcal{S}^{\mathsf{ORE}}$, cf. [10].

call a leakage function $\mathcal{L}_{\text{ideal}}$ *best-possible* or *ideal*, if one learns nothing else, i.e., if the leakage is given by

$$\mathcal{L}_{\text{ideal}}(m_1, \ldots, m_t) = \{(i, j) \mid m_i \leq m_j\}.$$

Note that unfortunately there is no known ORE scheme for superpolynomial message space with this ideal leakage, except for one that uses very strong assumptions, such as multilinear maps [5], rendering this scheme unsuitable in practice.

Assumptions on the Leakage Functions In the remainder of this work, we make a few assumptions about the leakage function.

Assumption 1. $\forall m : \mathcal{L}(m) = \emptyset$.

While one could think of ORE schemes, for which this assumption does not hold, the assumption holds for established schemes like the ones in [10, 23], as well as our ORE scheme.

Assumption 2. For t messages $m_1, \ldots, m_t$ and index $0 \leq i \leq t$, it holds that $L^{\leq i} := \mathcal{L}(m_1, \ldots, m_i)$ can be efficiently computed from $L = \mathcal{L}(m_1, \ldots, m_t)$.

While this assumption does not have to hold for all ORE leakage functions, it does hold for leakage functions that tightly capture the leakage of the respective ORE scheme. Again, this assumption holds for the ORE schemes in [10, 23], as well as for our ORE scheme.

2.3 Decision Tree Training

We consider a domain with data points x with X continuous attributes x_j and a label $\ell(x)$ from a small discrete set of labels. In this context, a decision tree is a binary tree, where each leaf node contains a label and each inner node contains

a tuple (j, t), where j is an index of an attribute and t is a value of the j-th attribute. When performing a classification on a data point x at an inner node (j, t), we recurse to the left child node, if $x_j \leq t$, and to the right child node, otherwise. This is repeated until reaching a leaf node, where the label of the node is returned.

Decision tree training is the task of building a decision tree given a set of labeled training data. Established decision tree frameworks like [6, 1] use variations of the recursive C4.5 algorithm by Quinlan [30]. An adaption of this algorithm can be seen in Algorithm 1. One source of variation in this algorithm is the heuristic H used in Line 11. A common example used here is the *Information Gain heuristic*. Following the definition in [21], it is defined as

$$H'(S) = -\sum_{l \in \mathbb{L}} \frac{|\{x \in S \colon \ell(x) = l\}| \log(|\{x \in S \colon \ell(x) = l\}|)}{|S|},$$

$$H(S, L, R) = H'(S) - \frac{|L|}{|S|} H'(L) - \frac{|R|}{|S|} H'(R),$$

where $\mathbb{L}$ is the set of labels. This heuristic describes the information-theoretic gain when separating the set S into sets L and R. Another common heuristic is the GINI Index, which performs nearly identically as the information gain heuristic [31]. The other common variation is the threshold value used in Line 13, where most frameworks use an intermediate value between the threshold and the next larger occurring attribute value. While in a classical setting both the labels

Algorithm 1 Decision tree training with a heuristic H.

```
1: function TrainDecisionTree(data)
2:     assert ∀x, y ∈ data : ∃j : x_j ≠ y_j
3:     if ∀x ∈ data: ℓ(x) = ℓ(data_0) then
4:         return LeafNode(ℓ(data_0))
5:     end if
6:     j* := thresh* := ⊥, h* := −∞
7:     for 1 ≤ j ≤ X do
8:         L := ∅
9:         for thresh ∈ {x_j | x ∈ data} in ascending order do
10:            L := L ∪ {x ∈ data | x_j = thresh}
11:            h := H(data, L, data \ L)
12:            if h > h* then
13:                j* := j, thresh* := thresh, h* := h
14:            end if
15:        end for
16:    end for
17:    L := {x ∈ data | x_j* ≤ thresh*}, R := data \ L
18:    return InnerNode(j*, thresh*, TrainDecisionTree(L), TrainDecisionTree(R))
19: end function
```

as well as the attribute values, are numeric values, we note that the training algo-

rithm itself only needs to evaluate the order of attribute values. This is required in Lines 9, 10 and 17. While one could think of a heuristic H which requires additional operations on the attributes, established heuristics like Information Gain do not consider the attribute values, but only take the labels into consideration.

It is possible to use different training algorithms and variations of trees, such as XBoost [9], which trains gradient-boosted trees. In this approach, the training process requires performing arithmetic operations on the attributes. Another variation is the use of decision forests consisting of multiple decision trees. A classification of a data point in such a decision forest is done by classifying it with each tree and performing a majority vote on the resulting labels. Bentéjac, Csörgő, and Martínez-Muñoz [4] have empirically shown that in a real-world scenario, gradient boosted trees, although requiring arithmetic operations on the training data, do not perform significantly better than decision forest that can be trained with Algorithm 1 only comparing elements and performing equality checks on the labels.

3 Updatable Order-Revealing Encryption

We now augment the definition of ORE to allow for updating a ciphertext from one key to another, while retaining the messages contained in the ciphertexts.

Definition 1 (Updatable ORE). *A 4-tuple of PPT algorithms* $\mathsf{ORE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp}, \mathsf{Upd})$ *is an* Updatable ORE (uORE) scheme *over key space* $\mathcal{K} = \mathbb{Z}_p^\times$*, message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*, if*

- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp})$ *is an ORE scheme over key space* $\mathcal{K}$*, message space* $\mathcal{M}$*, and ciphertext space* $\mathcal{C}$*.*
- $\mathsf{Upd}(k, c)$ *takes a key* $k \in \mathcal{K}$ *and a ciphertext* $c \in \mathcal{C}$ *as input and outputs a new ciphertext* $c' \in \mathcal{C}$*.*
- Enc *and* Upd *are deterministic.*

ORE *is correct, if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp})$ *is a correct ORE scheme and satisfies the updatability property:*

$$\forall k, k' \in \mathcal{K} \colon \mathsf{Upd}(k', \mathsf{Enc}(k, m)) = \mathsf{Enc}(k \cdot k', m)$$

Moreover, ORE *is a* secure uORE scheme w.r.t. a leakage function $\mathcal{L}$ *iff* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp})$ *is a secure ORE scheme w.r.t.* $\mathcal{L}$*.*

Note that in our definition, we require that the key space $\mathcal{K} \coloneqq \mathbb{Z}_p^\times$. This means that any key k is invertible modulo p.

For our construction of an uORE scheme, we adapt the scheme from [10]:

Construction 1 *Let* $(\mathbb{G}, p, [1])$ *be an additive group with prime order* $p > 3$ *and let* $\mathsf{PRF} \colon \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$ *be a key-homomorphic PRF with key space* $\mathbb{Z}_p$ *and message space* $\{0,1\}^*$*. Then, we define the uORE scheme* $\mathsf{ORE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp}, \mathsf{Upd})$ *with message space* $\mathcal{M} = \{0,1\}^n$ *for a parameter* n*, and ciphertext space* $\mathcal{C} = \mathbb{G}^n$*, as follows:*

- $\mathsf{Gen}(1^\kappa)$*: Return a uniformly random* $k \leftarrow \mathbb{Z}_p^\times$
- $\mathsf{Enc}(k, m = (m_1, \ldots, m_n))$*: For* $i = 1, \ldots, n$*, set*

$$u_i = (1 + m_i) \cdot \mathsf{PRF}(k, (m_1, \ldots, m_{i-1})).$$

  Return $ct = (u_1, \ldots, u_n)$.
- $\mathsf{Cmp}(ct = (u_1, \ldots, u_n), ct' = (u'_1, \ldots u'_n))$*: Find the smallest* i*, such that* $u_i \neq u'_i$*. If such an* i *exists and* $u'_i = 2 \cdot u_i$*, return* 1*. Otherwise, return* 0.
- $\mathsf{Upd}(k', ct = (u_1, \ldots, u_n))$*: Set* $u'_i = k' \cdot u_i$*. Return* $ct' = (u'_1, \ldots, u'_n)$.

Our construction is similar to the one by Chenette et al. [10]. In both cases, the key generation algorithm $\mathsf{Gen}(1^\kappa)$ samples a random element from the PRF key space, and $\mathsf{Enc}(k, (m_1, \ldots, m_n))$ is done bit by bit, by first computing $u'_i = \mathsf{PRF}(k, (m_1, \ldots, m_{i-1}))$ and then returning $u_i = u'_i$ if $m_i = 0$. If $m_i = 1$, both schemes return $u_i = \pi(u'_i)$ for an efficiently invertible permutation π. In both schemes, $\mathsf{Cmp}((u_1, \ldots, u_n), (u'_1, \ldots, u'_n))$ works by identifying the smallest i, for which $u_i \neq u'_i$ and checking if $u'_i = \pi(u_i)$ with the same permutation.

Comparing these two schemes, the only two differences are the PRF and the permutation π being used. In our scheme, we require the PRF to be key-homomorphic, which does not need to be the case in their scheme. This allows them to use $\mathbb{Z}_3$ as output space of the PRF and $\mathbb{Z}_3^n$ as the ciphertext space, whereas our used ciphertext space is $\mathbb{G}^n$. Moreover, we use $\pi(x) = 2 \cdot x$ as a permutation, whereas in their scheme, $\pi(x) = x + 1 \mod 3$ is used.

Because of this similarity, their security proof and ORE simulator also applies to our construction, when adjusting the permutation. Hence, both schemes are secure under the same leakage function:

Theorem 1. *Construction 1 is secure with the leakage function*

$$\mathcal{L}(m_1, \ldots, m_t) = \{(i, j, \mathsf{hsb}(m_i \oplus m_j)) \mid m_i < m_j\},$$

where $\mathsf{hsb}(x)$ *returns the position of the highest set bit of* x.

Because of the similarity to the proof of [10], we will only give a proof intuition: For (u)ORE security to hold, there needs to be a simulator $\mathcal{S}^{\mathsf{ORE}}$ that, given only the leakage of messages, needs to be able to generate ciphertexts that are indistinguishable from encryptions of the messages with a random but (during the games) fixed key. In a first step, one replaces the PRF with a random function via lazy sampling. When asked to generate the first ciphertext, $\mathcal{S}^{\mathsf{ORE}}$ samples n elements from the output space of the PRF uniformly at random and outputs them as the first ciphertext. When asked to generate ciphertexts for any subsequent messages, it learns the position of the leftmost differing bit between this message and previous messages, as well as the message bit at these positions. This allows it to answer with consistent parts of ciphertexts, where message prefixes are equal, and with $\pi(x)$ or $\pi^{-1}(x)$ where the most significant difference is. Finally, for all other positions, for which no common prefix with another message exists, it proceeds as in the case for the first message, sampling and outputting elements from the output space of the PRF.

Since this proof only requires the security of PRF and the efficient computation/inversion of π, the proof from [10] directly translates to our setting.

Remark 1. The leakage $\mathcal{L}$ of our scheme is actually sufficient to use faster sorting algorithms than plain comparison-based algorithms. For example MSD radix sort uses exactly the information provided by $\mathcal{L}$ to allow sorting in linear time. Sorting all ciphertexts by the first bit is easy, as there are only two comparable group elements for the first bit. After that, sorting the two partitions after the second bit is possible with the same approach. This reduces the amount of group operations required for sorting from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$. This is especially interesting, as a main use-case for ORE are encrypted databases, where sorting is a major concern. Large databases often use advanced sorting algorithms that are not comparison-based, so the additional leakage of $\mathcal{L}$ over $\mathcal{L}_{\text{ideal}}$ can be used to speed up sorting.

Theorem 2. *Construction 1 is a correct ORE scheme.*

Proof. Fix two messages $m, m' \in \{0,1\}^n$ with $m = (m_1, \ldots, m_n)$ and $m' = (m'_1, \ldots, m'_n)$. Then, we show that the correctness property holds for any $k \leftarrow \mathsf{Gen}(1^\kappa)$, $(u_1, \ldots, u_n) \leftarrow \mathsf{Enc}(k, m)$ and $(u'_1, \ldots, u'_n) \leftarrow \mathsf{Enc}(k, m')$. We consider each case separately.

- $m < m'$**:** In this case, there exists an i, such that $m_j = m'_j$ for $j < i$ and $m_j = 0$ and $m'_j = 1$. In this case, it holds that $u_j = u'_j$ for $j < i$. For PRF output $o = \mathsf{PRF}(k, (m_1, \ldots, m_{i-1}))$, and by definition of Enc, it holds that $u_i = o$ and $u'_i = 2 \cdot o$. If $o \neq 0_{\mathbb{G}}$, u_i and u'_i are different and Cmp returns 1 in this case. The probability for the event that $o = 0_{\mathbb{G}}$ is negligible (which follows from the fact that this probability is $1/p$ for a random function and because PRF is a PRF). Therefore, Cmp will output 1 with overwhelming probability.
- $m = m'$**:** In this case $u_i = u'_i$ for all i, as Enc is deterministic and Cmp will output 0.
- $m > m'$**:** Similarly to the first case, there exists an i, such that $m_j = m'_j$ for $j < i$ and $m_j = 1$ and $m'_j = 0$. With the same argument as in the case for $m < m'$, we know that $u_j = u'_j$ for $j < i$ and $u_i = 2 \cdot u'_i$. Since $\mathbb{G}$ is of prime order with $p > 3$, it also holds that $u'_i \neq 2 \cdot u_i = 2 \cdot 2 \cdot u'_i$ and therefore Cmp returns 0 with overwhelming probability. □

Theorem 3. *Construction 1 satisfies the updatability property.*

Proof. The updatability follows from the key-homomorphism of PRF. For all $k, k' \in \mathcal{K}$ and $m \in \{0,1\}^n$, it holds that

$$\begin{aligned}
\mathsf{Upd}(k', \mathsf{Enc}(k, m)) &= \mathsf{Upd}(k', ((1 + m_i) \cdot \mathsf{PRF}(k, m_{1..i-1}))_{i=1,\ldots,n}) \\
&= (k' \cdot (1 + m_i) \cdot \mathsf{PRF}(k, m_{1..i-1}))_{i=1,\ldots,n} \\
&= ((1 + m_i) \cdot \mathsf{PRF}(k' \cdot k, m_{1..i-1}))_{i=1,\ldots,n} \\
&= \mathsf{Enc}(k' \cdot k, m),
\end{aligned}$$

where $m_{1..i-1} \coloneqq (m_1, \ldots, m_{i-1})$. □

4 Secure Decision Tree Training

In our protocol, we want to train a decision tree without revealing the training data, using the previously constructed Updatable ORE scheme. The core idea is to have training data from one party uORE encrypted under a key which the party itself does not know. To accomplish this, we make use of the updatability of the created ciphertexts. In the second step, we apply the decision tree training algorithm in Algorithm 1 to the ORE ciphertexts from the previous step. Here, we make use of the fact that Algorithm 1 only requires the comparability of attributes, and is deterministic.

In principle, any decision tree training algorithm with these properties can be used. While the determinism of the training algorithm is required, de-randomization can be done by prepending the protocol with a secure coin-toss and using these coins as input for the decision tree training. If randomness in the training algorithm does not have a security impact, an alternative and more performant way of de-randomization is to use a non-cryptographic PRG with a fixed seed.

Let us first formalize an ideal functionality that captures the security of secure decision tree protocols.

Definition 2 (Ideal functionality $\mathcal{F}_{\mathsf{DTTrain}}$). *$\mathcal{F}_{\mathsf{DTTrain}}$ in Fig. 2 models the security of the decision tree training protocol. A graphical representation thereof is in Fig. 3. In this setting, there are two parties A and B. The training data has X attributes and discrete labels.*

- Upon receiving input $(\texttt{Input}, n_{\mathsf{B}}, m^{\mathsf{B}}_{1,1}, \dots m^{\mathsf{B}}_{n_{\mathsf{B}},X})$ from B, send $(\texttt{InputReceived}, n_{\mathsf{B}})$ to A
- Upon receiving $(\texttt{Input}, n_{\mathsf{A}}, m^{\mathsf{A}}_{1,1}, \dots, m^{\mathsf{A}}_{n_{\mathsf{A}},X}, l^{\mathsf{A}}_1, \dots, l^{\mathsf{A}}_{n_{\mathsf{A}}})$ from A, compute $L_j := \mathcal{L}(m^{\mathsf{A}}_{1,j}, \dots, m^{\mathsf{A}}_{n_{\mathsf{A}},j}, m^{\mathsf{B}}_{1,j}, \dots, m^{\mathsf{B}}_{n_{\mathsf{B}},j})$ and send $(\texttt{Leakage}, (L_1, \dots, L_X), (l^{\mathsf{A}}_1, \dots, l^{\mathsf{A}}_{n_{\mathsf{A}}}))$ to B.
- Upon receiving input $(\texttt{Labels}, l^{\mathsf{B}}_1, \dots, l^{\mathsf{B}}_{n_{\mathsf{B}}})$ from B, compute the decision tree $tree = \mathsf{train}(m^{\mathsf{A}}_{1,1}, \dots, m^{\mathsf{A}}_{n_{\mathsf{A}},X}, m^{\mathsf{B}}_{1,1}, \dots, m^{\mathsf{B}}_{n_{\mathsf{B}},X}, l^{\mathsf{A}}_1, \dots, l^{\mathsf{A}}_{n_{\mathsf{A}}}, l^{\mathsf{B}}_1, \dots, l^{\mathsf{B}}_{n_{\mathsf{B}}})$ and output $(\texttt{Trained}, tree)$ to A

Fig. 2: Ideal functionality $\mathcal{F}_{\mathsf{DTTrain}}$.

Construction 2 (Protocol π_{DTTrain}) *Here, we define the two-party protocol π_{DTTrain} between parties A and B. As before, let X be the number of attributes of the training data. Also, let $m^{\mathsf{A}}_{i,j}$ be the j-th attribute of the i-th training data of A with the labels l^{A}_i (respectively for the dataset of B), and let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Cmp}, \mathsf{Upd})$ be an uORE scheme. Then we define the protocol π_{DTTrain} as follows:*

B: – *Generate ORE keys $k^{\mathsf{B}}_{i,j}$ for $1 \leq i \leq n_{\mathsf{B}}, 1 \leq j \leq X$*

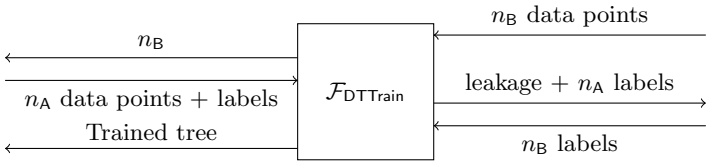

Fig. 3: Graphical representation of $\mathcal{F}_{\mathsf{DTTrain}}$. Some details about the concrete values being sent to/from the ideal functionality are omitted.

– *Send* $c^{\mathsf{B}}_{i,j} = \mathsf{Enc}(k^{\mathsf{B}}_{i,j}, m^{\mathsf{B}}_{i,j})$ *for* $1 \le i \le n_\mathsf{B}, 1 \le j \le X$ *to* A

A: – *Generate ORE keys* k^{A}_j *for* $1 \le j \le X$
– *Send* $C^{\mathsf{A}}_{i,j} = \mathsf{Enc}(k^{\mathsf{A}}_j, m^{\mathsf{A}}_{i,j})$ *for* $1 \le i \le n_\mathsf{A}, 1 \le j \le X$ *to* B
– *Send labels* l^{A}_i *for* $1 \le i \le n_\mathsf{A}$ *to* B
– *Send* $c'^{\mathsf{B}}_{i,j} = \mathsf{Upd}(k^{\mathsf{A}}_j, c^{\mathsf{B}}_{i,j})$ *for* $1 \le i \le n_\mathsf{B}, 1 \le j \le X$ *to* B

B: – *Compute* $C^{\mathsf{B}}_{i,j} = \mathsf{Upd}(1/k^{\mathsf{B}}_{i,j}, c'^{\mathsf{B}}_{i,j})$ *for* $1 \le i \le n_\mathsf{B}, 1 \le j \le X$
– *Train the decision tree on the data points* $(C^{\mathsf{A}}_i, l^{\mathsf{A}}_i)_{i=1\ldots,n_\mathsf{A}}$ *and* $(C^{\mathsf{B}}_i, l^{\mathsf{B}}_i)_{i=1\ldots,n_\mathsf{B}}$*, obtaining a trained decision tree.*
– *Send the tree to* A

A: – *Decrypt and output the decision tree: For each inner node in the tree containing an attribute id* j *and an encrypted value* v*, replace* v *with* $\mathsf{Dec}(k^{\mathsf{A}}_j, v)$.

Note that this protocol has a constant number of rounds, as only three messages are exchanged. Here, A learns no additional information beyond what can be learned from her own training data and the trained decision tree. B receives only the leakage of the ORE scheme of both parties training data for each attribute separately and the respective label. Note that because A uses different keys to encrypt the values of different attributes, B only receives the leakage $\mathcal{L}(m^{\mathsf{A}}_{1,j}, \ldots m^{\mathsf{A}}_{n_\mathsf{A},j}, m^{\mathsf{B}}_{1,j}, \ldots m^{\mathsf{B}}_{n_\mathsf{B},j})$ for each j. This leakage is much smaller than the entire leakage $\mathcal{L}(m^{\mathsf{A}}_{1,1}, \ldots m^{\mathsf{A}}_{n_\mathsf{A},X}, m^{\mathsf{B}}_{1,1}, \ldots m^{\mathsf{B}}_{n_\mathsf{B},X})$, because attribute values of different attributes cannot be compared.

Theorem 4. π_{DTTrain} *securely realizes* $\mathcal{F}_{\mathsf{DTTrain}}$ *for static corruption and semi-honest adversaries if the uORE scheme is secure with leakage* $\mathcal{L}$*.*

To show this theorem, we follow the UC framework and construct a simulator, such that the real world running the protocol and ideal world with $\mathcal{F}_{\mathsf{DTTrain}}$ are indistinguishable. Concretely, for any PPT-environment controlling a corrupted party A or B, we show that the environment cannot distinguish between a real interaction of the corrupted party with the uncorrupted one and an interaction of the corrupted party with the ideal functionality through the simulator. Because we consider semi-honest adversaries, the environment chooses the inputs of the honest and corrupted parties and learns their output. It also learns sent and received messages, internal state and randomness of corrupted parties.

We give two different simulators, one for the case where A is corrupted and one for the case where B is corrupted. (If both parties are corrupted, no meaningful security guarantees are left.)

For the case, where A is corrupted, we use the simulator $\mathcal{S}$ from Fig. 4. To

- When receiving $(\texttt{InputReceived}, n_\mathsf{B})$ from $\mathcal{F}_{\mathsf{DTTrain}}$, invoke the n_B instances of the ORE simulator with the leakage $\emptyset$ (possible by Assumption 1) and send the resulting ciphertexts to A.
- When receiving ciphertexts $C^\mathsf{A}_{i,j}$, updated ciphertexts $c'^\mathsf{B}_{i,j}$ and labels l^A_i from A, extract A's ORE keys k^A_j and messages $m^\mathsf{A}_{i,j}$. Send $(\texttt{Input}, n_\mathsf{A}, m^\mathsf{A}_{1,1}, \ldots, m^\mathsf{A}_{n_\mathsf{A},X}, l^\mathsf{A}_1, \ldots, l^\mathsf{A}_{n_\mathsf{A}})$ to $\mathcal{F}_{\mathsf{DTTrain}}$ in the name of A.
- When receiving $(\texttt{Trained}, \mathit{tree})$ from $\mathcal{F}_{\mathsf{DTTrain}}$ for A, encrypt each value v in a node branching by attribute j as $\mathsf{Enc}(k^\mathsf{A}_j, v)$ and send the resulted encrypted tree to A.
- When receiving a query for $\mathcal{F}_{\mathsf{RO}}$, answer consistently or draw a random $x \leftarrow \mathbb{Z}_p^\times$ and return $[x]$ if the message has not previously been queried.

Fig. 4: Simulator for a corrupted A.

show its validity, we define a number of games G_i, each having (implicitly defined) ideal functionalities $\mathcal{F}_i$ and simulators $\mathcal{S}_i$, and each being a modified version of the previous one. The first game is defined, s.t. the corrupted A interacts with the honest B through the protocol and the last game is the game where the corrupted A interacts with the ideal functionality through the simulator. We show that games G_i are indistinguishable from G_{i+1}.

- G_0: The execution of π_{DTTrain} with dummy adversary $\mathcal{D}$.
- G_1: The execution with a dummy ideal functionality $\mathcal{F}_1$ that lets the adversary determine all inputs and learn all outputs of the honest party. $\mathcal{S}_1$ is the simulator that executes the protocol π_{DTTrain} honestly on behalf of the honest party using the inputs from $\mathcal{F}_1$ and making outputs through $\mathcal{F}_1$.
- G_2: The same as G_1, except that when sending the encrypted decision tree to A, instead of sending the tree, $\mathcal{S}_2$ extracts the keys k_j from A, uses them to decrypt and reencrypt the tree and sends this reencrypted tree.
- G_3: The same as G_2, except that now $\mathcal{S}_3$ does not perform the training on the ciphertexts, it now performs the training on the plaintext training data, where it extracted A's training data and received B's training data from $\mathcal{F}_4$.
- G_4: The same as G_3, except that instead of sending the values $c_{i,j} = \mathsf{Enc}(k_{i,j}, m_{i,j})$ to A, $\mathcal{S}_4$ generates $c'_{i,j} = \mathcal{S}^{\mathsf{ORE}}(\emptyset)$ and sends these values to A instead.
- G_5: The execution of the ideal functionality $\mathcal{F}_{\mathsf{DTTrain}}$ with the simulator $\mathcal{S}$ as in Fig. 4.

As we can see, the game G_0 is the game, in which the corrupted party interacts with the honest one using the protocol, whereas G_5 is the game, where the corrupted party interacts with the ideal functionality through the simulator.

Claim 1 $\mathsf{G}_0 \overset{c}{\approx} \mathsf{G}_1$

Proof. These changes are only syntactic and therefore oblivious to the environment. The claim follows. □

Claim 2 $\mathsf{G}_1 \overset{c}{\approx} \mathsf{G}_2$

Proof. The simulator participates in the protocol as the honest party would. Therefore, each inner node of the tree contains elements (j, c), where c is either a ciphertext from A, in which case $c = \mathsf{Enc}(k_j^{\mathsf{A}}, m)$ for some m, or a ciphertext from the simulator, in which case

$$c = \mathsf{Upd}(k'^{-1}, \mathsf{Upd}(k_j^{\mathsf{A}}, \mathsf{Enc}(k', m))) = \mathsf{Enc}(k_j^{\mathsf{A}}, m)$$

for some k' and m. Therefore, decrypting c with k_j^{A} and deterministically reencrypting the result again with the same key results in the same ciphertext. Hence, the games are indistinguishable. □

Claim 3 $\mathsf{G}_2 \overset{c}{\approx} \mathsf{G}_3$

Proof. It follows from the uORE correctness that evaluating the order on the ciphertexts is equivalent to evaluating the order on the plaintexts (with overwhelming probability), if the ciphertexts to be compared are ciphertexts under the same key. The decision tree training algorithm only compares ciphertexts associated with the same attribute and ciphertexts for an attribute j are all ciphertexts under key k_j^{A} (either directly or by updating and reverse-updating). Therefore, it follows from the correctness of the ORE scheme that encrypting the messages, training the decision tree on the ciphertexts and decrypting the decision tree again (as done in G_2) results (with overwhelming probability) in the same tree as directly training the decision tree on the plaintexts (as done in G_3). □

Claim 4 $\mathsf{G}_3 \overset{c}{\approx} \mathsf{G}_4$

Proof. This change is only relevant for the ciphertexts the simulator sends to A. While it receives these ciphertexts back, updated with A's keys, they are no longer used by the simulator in any further computations. Because of Assumption 1 and the fact that the keys $k_{i,j}$ are only ever used for encryption once, it holds that for all messages m, including the messages the simulator encrypts in G, that $\mathcal{L}(m) = \emptyset$. Therefore, an adversary that can distinguish between G_3 and G_4 can be used to distinguish between $\mathsf{REAL}_{\mathcal{A}}^{\mathsf{ORE}}(\kappa)$ and $\mathsf{SIM}_{\mathcal{A},\mathcal{S},\mathcal{L}}^{\mathsf{ORE}}(\kappa)$ using a hybrid argument. □

Claim 5 $\mathsf{G}_4 \overset{c}{\approx} \mathsf{G}_5$

Proof. The only difference between these two games is who performs the training of the decision tree. In G_5, the simulator $\mathcal{S}_5$ performs the training of the decision tree, while in G_6, the ideal functionality $\mathcal{F}_6$ performs the training. In both cases, the training is performed on the same training data and as training is deterministic, it results in the same decision tree in both cases. □

For the case, where B is corrupted, we use the simulator $\mathcal{S}$ from Fig. 5. Again, we define hybrid games to show its validity:

- When receiving ciphertexts $c^{\mathsf{B}}_{i,j}$ from B, extract the messages $m^{\mathsf{B}}_{i,j}$ from B and send $(\texttt{Input}, n_{\mathsf{B}}, m^{\mathsf{B}}_{1,1}, \dots m^{\mathsf{B}}_{n_{\mathsf{B}},X})$ to $\mathcal{F}_{\mathsf{DTTrain}}$ in the name of B.
- Upon receiving $(\texttt{Leakage}, (L_1, \dots, L_X), (l^{\mathsf{A}}_1, \dots, l^{\mathsf{A}}_{n_{\mathsf{A}}}))$ from $\mathcal{F}_{\mathsf{DTTrain}}$ for B, proceed as follows for all j:
  - For each i, compute the sub-leakage $L_j^{\leq i}$ from L_j. This is possible using Assumption 2.
  - Generate ciphertexts $C_{i,j} = \mathcal{S}^{\mathsf{ORE}}(L_j^{\leq i})$ for $1 \leq i \leq n_{\mathsf{A}} + n_{\mathsf{B}}$
  - Set $c'^{\mathsf{B}}_{i,j} \leftarrow \mathsf{Upd}(k^{\mathsf{B}}_{i,j}, C_{i+n_{\mathsf{A}},j})$ for $1 \leq i \leq n_{\mathsf{B}}$

  Send the $C_{i,j}$, $c'^{\mathsf{B}}_{i,j}$ and l^{A}_j to B.
- Upon receiving *tree* from B, extract labels l^{B}_j from B and send $(\texttt{Labels}, l^{\mathsf{B}}_1, \dots, l^{\mathsf{B}}_{n_{\mathsf{B}}})$ to $\mathcal{F}_{\mathsf{DTTrain}}$ in the name of B.

Fig. 5: Simulator for corrupted B.

- G'_0: The execution of π_{DTTrain} with dummy adversary $\mathcal{D}$.
- G'_1: The execution with an ideal functionality $\mathcal{F}'_1$ that lets the adversary determine all inputs and learn all outputs. $\mathcal{S}'_1$ is the simulator that executes the protocol π_{DTTrain} honestly on behalf of the honest party using the inputs from $\mathcal{F}'_1$ and making outputs through $\mathcal{F}'_1$.
- G'_2: The same as G'_1, except that when $\mathcal{S}'_2$ would update a ciphertext, which B computed as $ct = \mathsf{Enc}(k', m)$ to $ct^* = \mathsf{Upd}(k, ct)$, instead it computes $ct^* = \mathsf{Upd}(k', \mathsf{Enc}(k, m))$ using the corresponding key k' and message m it extracts from B.
- G'_3: The same as G'_2, but instead of decrypting and outputting the tree received from B, the simulator extracts the training data from B. Then it uses both parties training data and labels to train the decision tree in plain text and outputs the tree to A through $\mathcal{F}'_3$.
- G'_4: The same as G'_3, except that whenever we would perform an encryption (both in the encrypt- or update-step) as $\mathsf{Enc}(k, m)$, we compute it with $\mathcal{S}^{\mathsf{ORE}}$ using the corresponding leakage, using one instance per k.
- G'_5: The ideal functionality $\mathcal{F}_{\mathsf{DTTrain}}$ with the simulator $\mathcal{S}$.

Now we look at the case, where B is corrupted:

Claim 6 $\mathsf{G}'_0 \overset{c}{\approx} \mathsf{G}'_1$

Proof. These changes are only syntactic and therefore oblivious to the environment. The claim follows. □

Claim 7 $\mathsf{G}'_1 \overset{c}{\approx} \mathsf{G}'_2$

Proof. As the difference between these two games is the order in which Enc and Upd are applied, this claim follows directly from the updatability property. □

Claim 8 $\mathsf{G}'_2 \overset{c}{\approx} \mathsf{G}'_3$

Proof. The only difference between these two games is the decision tree that is oututted to A. In G_3, the simulator knows A's training data, because it received them from $\mathcal{F}_3$ and it knows B's training data, because it can extract them from B. From the uORE correctness and the fact that the decision tree training is deterministic, it follows that the decision tree $\mathcal{S}_3'$ computes in plaintext is identical to the one B computes in G_3' and G_2' on the ciphertexts. □

Claim 9 $\mathsf{G}_3' \overset{c}{\approx} \mathsf{G}_4'$

Proof. In G_3', the ORE encryption is performed using real keys k_j, whereas in G_4', the ORE simulator is used to generate ciphertexts. To show this indistinguishability, we define additional hybrids H_j, where for the first j attributes, the ORE simulator is used to generate ciphertexts, and for all other attributes, a real encryption is performed. It holds that $\mathsf{G}_4' = \mathsf{H}_0$ and $\mathsf{G}_5' = \mathsf{H}_X$.

The indistinguishability of these games therefore follows from the ORE-security using a standard hybrid argument. □

Claim 10 $\mathsf{G}_4' \overset{c}{\approx} \mathsf{G}_5'$

Proof. The difference between these two games is that in G_5', the simulator performs the training and sends the trained decision tree to A through the ideal functionality, while in G_6' the ideal functionality performs the training and outputs it directly. As the decision tree training algorithm is deterministic and the same input to the training process is used by $\mathcal{S}_5'$ and $\mathcal{F}_6'$, the tree is identical and the games are indistinguishable. □

The security of the protocol π_{DTTrain} (Theorem 4) now follows from Claim 1 – 5 and Claim 6 – 10.

4.1 Variations of the Training Process

In the protocol from Construction 2, we only considered standard decision tree training. When using decision trees in practice, additional steps are usually taken like pruning, limiting the depth of the tree, gradient boosted training or training a decision forest for better classification accuracy.

Performing gradient boosted training is inherently not compatible with our approach, as it requires performing arithmetic operations on attribute values, which ORE schemes do not support.

Pruning and limiting the depth of the tree could be performed by B by adding a leaf node instead of an inner node to the tree, whenever the number of datapoints at the current position in the tree is small enough or if a certain depth of the tree is reached during the training process. Both of these techniques are compatible with our decision tree training protocol. Indeed any form of pruning that adheres to the limitation that attributes can only be compared, but no arithmetic can be performed on the attribute values, is compatible with our approach.

Additionally, our protocol can be used for training a decision forest. Training a T-tree decision forest can be done as follows:

1. Each party partitions their training data into T subsets.
2. The parties run π_{DTTrain} once for each subset in parallel to perform the training on each data.
3. After receiving the T decision trees from B, A outputs the decision forest containing these trees.

This realizes a forest-variant of the ideal functionality $\mathcal{F}_{\mathsf{DTTrain}}$. The security of the protocol follows from the universal composability theorem in the UC model.

Another variation is to consider training data that is split vertically between parties, i.e. A has one part of the attributes and B has a different set of the attributes of the same dataset (and they share some kind of id attribute to match up the data). In this setting, training is easily possible with the ORE-based approach by having A encrypt her attribute values using an ORE scheme with one key per attribute and sending the ciphertexts to B. B can then train a decision tree using A's encrypted attributes and his own attributes in plaintext, as no comparison between his and A's data is required. Indeed this does not even require the ORE scheme to be updatable.

4.2 Graceful Degradation using Enclaves

Hardware enclaves allow other parties to verify the code running in them while ideally hiding all internal state. While hiding internal state is difficult due to side channels, either inherent in the enclave program (like timing or memory access patterns), or due to the used platform (like power consumption, cache timing or other microarchitecural state side channels), the minimal functionality of an enclave, namely to attest the correct program execution, is usually well hardened and implemented side-channel free. These side channels have motivated two major security models for the functionality provided by an enclave system: "regular" enclaves (hiding all internal state) and transparent enclaves (revealing all internal state, especially the used randomness, but not attestation keys). If additionally attestation keys are leaked, the enclave provides no meaningful security. In between transparent and regular, enclave models with varying amount of side channels can be useful, like explicitly modeling a memory access side channel, as done in [29].

In the minimal form of a transparent enclave, they can be seen as a generic passive-to-active compiler. Executing an arbitrary passively secure protocol inside an enclave allows other involved parties to verify the produced attestation evidence to ensure that the other parties executed their part of the protocol correctly.

When applied to the presented decision tree training protocol (Construction 2), we can go one step further: When the protocol for B is executed inside a non-transparent enclave, the leakage from the ORE scheme to B is hidden. To implement this, A needs to send her ciphertexts to the enclave of B confidentially, which can be done by performing a key exchange into the enclave. Additional care needs to be taken to ensure that the enclave program does not leak more information about the input than necessary. Also, the enclave needs to hold a

Table 1: Comparison of security under different enclave security assumptions with radix sort and comparison-based sorting.

	ours with radix	ours without radix	plaintext with radix	plaintext without radix
fully secure enclaves	✓	✓	✓	✓
with memory side channels	ORE leakage	ideal-ORE leakage	ORE-leakage	ideal-ORE leakage
transparent enclaves	ORE leakage	ORE leakage	full leakage	full leakage

significant amount of memory to store the ORE-encrypted training data. Requiring access to this amount of data will result in many memory-management operations by the enclave system and thus has an impact on the overall performance.

An overview of the provided security in the different scenarios is given in Table 1. The algorithms that are compared are:

- the proposed algorithm with MSD radix sort inside an enclave
- the proposed algorithm with comparison-based sorting, where all comparisons are done memory-oblivious (e.g. using the primitives from [29])
- the plaintext decision tree training algorithm executed inside an enclave using MSD radix sort
- the plaintext decision tree training algorithm executed inside an enclave with comparison-based sorting using memory-oblivious comparisons

When using a fully secure enclave, all computation happens inside the enclave and cannot be eavesdropped or tampered with, so all algorithms are secure. When we assume memory side channels, radix sort exploits the concrete ORE leakage and thereby leaks it via the memory access patterns, both in the plaintext and encrypted variant. The other two algorithms only use the result of pairwise comparisons and thereby leak at most an ideal-ORE leakage (cf. Section 2.2). Using comparison-based sorting and memory-oblivious comparisons, memory access patterns can only leak at most the results of those comparisons. When the enclave becomes fully transparent, all information stored inside the enclave is potentially leaked, which are ORE ciphertexts for the first two algorithms and the plain training data for the last two.

As can be seen, our algorithm provides a more graceful degradation of security when the enclave fails to provide its security promise (e.g. due to expected or unexpected side channels in the implementation), at the cost of higher computational overhead. Therefore, if the trust model of the enclave is uncertain, combining our approach with secure enclaves allows decision tree training with less leakage than solely relying on either approach exclusively.

Note that the *plaintext* algorithm needs to employ similar techniques to the decision tree evaluation algorithm from [29] to avoid additional leakage. However as the authors only describe an evaluation but no training algorithm, no direct comparison can be made with the decision tree algorithm they provide.

Table 2: Proportion of leaked bits to total bits, for leakage $\mathcal{L}$ on n uniformly random messages of length k bits.

	$k = 8, n = 8$	$k = 8, n = 256$	$k = 8, n = 512$	$k = 64, n = 1000$	$k = 64, n = 10000$	$k = 64, n = 50000$
Leaked Bits	39.1%	93.6%	97.7%	16.0%	21.3%	24.9%

5 Analysis of the leakage

Grubbs et al. [15] have shown that in the context of encrypted databases, there are datasets, such that when encrypting them using ORE schemes, no meaningful security guarantees are left. They have shown that in one of their datasets containing first and last names, they could recover 98% of all first names and 75% of all last names in a database encrypted using the scheme from [10].

While these results also apply to our scheme and therefore also to our decision tree training protocol, we want to emphasize that these results are not universally applicable to all datasets. In their example, the dataset of first names had relatively low entropy with the most common first name appearing in 5% of all cases. Jurado, Palamidessi, and Smith [18] have shown for example that attacks recovering all ORE-encrypted data is possible when the amount of ciphertexts is large compared to the message space, whereas this is not possible if the amount of ciphertexts is significantly smaller than the message space.

To give meaningful security guarantees, we analyze the leakage from an information-theoretic point of view. We consider the uniform distribution of messages, because for this distribution, each bit of the message space contains one bit of information. In a first step, we analyze the leakage when both parties use uniformly random training data. In a second step, we give an upper bound for the leakage, when an adversary (B in the case of the decision tree training protocol) selects its training data maliciously. Finally, we argue why considering only the uniform distribution is sufficient.

5.1 Leakage for Random Message Selection

In the case where both parties use uniformly random messages as inputs, we can experimentally estimate the information leaked on different datasets. We consider a bit to be leaked, if the leakage function allows to infer the value of this bit. In our experiments, we sample n data samples uniformly at random from the bitstrings of length k and count the number of leaked bits. These results are available in Table 2. As we can see, the experiments show a significant leakage when the amount of data samples is large compared to the domain, leaking nearly all bits if there are more data samples than there are possible messages in the domain. If the message space is significantly larger than the amount of messages leaking information, the amount of bits leaked is less than 25% (for messages chosen uniformly at random). This matches the result of Jurado, Palamidessi, and Smith [18], namely that for ORE to provide a benefit over comparing the

messages in plain text, the message space must be much larger than the amount of messages encrypted under the same key.

This suggests that the leakage from our ORE approach is small enough to provide some security if the attributes of the training data have sufficiently large entropy, compared to the amount of training data. An example for attributes that have naturally high entropy are geopositions with latitude and longitude.

5.2 Additional Leakage for Malicious Message Selection

We also want to give an analytical upper bound of the additionally leaked information for independently uniformly distributed training data if one party selects their inputs maliciously. This is not a classical setting for ORE, but makes sense in our case, because here, the party receiving the leakage (B in the case of π_{DTTrain}) contributes data influencing the leakage.

Let $\mathcal{L}$ be the leakage function as in Theorem 1. Consider the following experiment with an attacker choosing $N = 2^k$ messages:

1. The adversary chooses N messages $m_i^* \in \mathcal{M} = \{0,1\}^n$.
2. The experiment chooses a message $m \leftarrow \mathcal{M}$ uniformly at random.
3. The adversary learns $\mathcal{L}(m, m_1^*, \ldots, m_N^*)$.

Choosing the messages optimally to have the maximum (over the attacker-chosen $m_1^*, \ldots, m_N^*$) leaked information on average (over m), the adversary receives no more than $(k+2)$bit of information.

Consider the first attacker-chosen message m_1^*. The leakage contains the information whether the first bit of m is equal to the first bit of m_1^* (the position of the most significant different bit is $\mathsf{hsb}(m \oplus m_1^*) > 1$) or if they are unequal ($\mathsf{hsb}(m \oplus m_1^*) = 1$). Therefore, the attacker learns the first bit of m containing 1bit of information. The same argument also holds for the second bit, but only if the first bit is the same in m and m_1^*.

Therefore, he obtains the second bit (and therefore one additional bit of information) with probability $1/2$, as m's first bit was chosen uniformly at random. Generalizing this for more bits, he learns the i-th bit of m with probability $1/2^{i-1}$. For the expected information, we get the geometric series:

$$\sum_{i=1}^{n} \frac{1}{2^{i-1}} \cdot 1\text{bit} \leq 2\text{bit}.$$

To generalize the maximum leakage to $N = 2^k$ attacker-chosen messages, we consider the information stored in the first $k+1$ bits of m and the remaining bits separately. (Here, we assume $k+1 < n$, as otherwise, the attacker would choose every second message from $\mathcal{M} = \{0,1\}^n$ and learn the content of all ciphertexts.) The first $k+1$ bits of m contain $(k+1)$bit of information, so that is the maximum amount of information an attacker can infer. For the remaining message bits, the probability of m having the same $k+1$-bit prefix as any of

the m_i^*, and therefore causing a bit to leak, is $1/2^{k+1}$. Ignoring possible overlap between messages gives us an upper bound of

$$2^k \cdot \frac{1}{2^{k+1}} \cdot 2\text{bit} = 1\text{bit},$$

where the 2bit is again an upper bound for the geometric series over the expected leaked information in the remaining bits.

Combining the two leakages, we get $(k+1)$bit + 1bit $= (k+2)$bit. This establishes an upper bound for the average leakage.

While we only considered a single message m selected by the experiment, it can naturally be extended to a setting, where the experiment chooses multiple messages $m_1, \ldots, m_{N'}$ and the adversary receives $\mathcal{L}(m_1, m_1^*, \ldots, m_N^*), \ldots, \mathcal{L}(m_{N'}, m_1^*, \ldots, m_N^*)$ instead. (We are only interested in the *additional leakage* by the attacker, the leakage $\mathcal{L}(m_1, \ldots, m_{N'})$ is already analyzed in the previous subsection.) As long as these messages are chosen independently from each other, the maximum leakage *per message* m_i remains the same. This multi-message scenario captures the leakage B receives about A's training data in π_{DTTrain} for each attribute when selecting his messages maliciously.

Note that this analysis extends to the use of multiple uncorrelated attributes, due to the use of separate ORE keys per attribute. If multiple attributes are correlated, this correlation can be interpreted as information known in advance by the adversary. Hence, the upper bound on the average leakage also holds for the information given what is known to the adversary, due to a union bound.

Overall, for any adversarially selected input data of B in π_{DTTrain} consisting of $N = 2^k$ training data, the expected leaked information consists of $(\log_2(N) + 2)$bit per A's training data per attribute.

5.3 Transformation for Non-uniform Distributions

While the distribution of input data is important because changing the encoding of messages (and therefore also the distribution over the message space) influences how much and what information is leaked, we want to emphasize that this is not a restriction, as any input data distribution can be encoded in such a way that the resulting distribution is uniform. We now sketch one way to do this.

Let μ be the probability measure of the distribution over the message space $\mathcal{M}$. Assume there exists a l, such that $\mu(m) \cdot 2^l \in \mathbb{N}$ holds for all $m \in \mathcal{M}$.[3] Then we encode the messages into $\{0,1\}^l$ as follows: The valid encodings of a message m are the $\mu(m) \cdot 2^l$ bitstrings starting from the bitstring of the binary representation of $\mu(\{m' \in \mathcal{M} \mid m' < m\}) \cdot 2^l$. To encode a message, one of its valid encodings is used uniformly at random. This encoding preserves the order, but the resulting distribution is uniform over $\{0,1\}^l$.

[3] If the assumption does not hold, an approximation of μ with powers of 2^{-l} for some l results in a distribution, that is computationally indistinguishable from uniform.

Table 3: Operations per second using ORE with a 32-bit message space. Sort refers sorting 1000 ciphertexts using Java's Arrays.sort() or our own implementation of MSD radix sort.

	Our scheme	Scheme from [10]
Encryption	$1.6 \cdot 10^2$	$7.2 \cdot 10^4$
Update	$2.0 \cdot 10^2$	–
Comparison	$7.0 \cdot 10^4$	$6.9 \cdot 10^7$
Sort	$8.0 \cdot 10^0$	$3.8 \cdot 10^5$
Radix sort	$6.9 \cdot 10^1$	$1.0 \cdot 10^3$

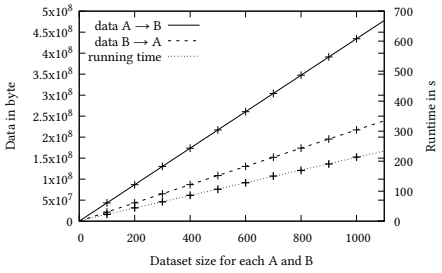

Fig. 6: Full results of the benchmarks from Table 4 where the datasets of A and B are of the same size.

6 Implementation and evaluation

We implemented the ORE scheme from [10], our updatable ORE scheme from Construction 1, as well as our decision tree training protocol from Construction 2 in Java/Kotlin[4] and used it to evaluate the practical efficiency of our protocol. For the group $\mathbb{G}$, we used the Ed25519 curve. We used the PRF from Eq. (1), implemented the random oracle using SHA-256 and mapped its output to points on the curve $\mathbb{G} \setminus \{[0]\}$. We ran our experiments on a machine with two AMD EPYC-Rome 7282 processors with 16 Cores/32 Threads and 90 GB of RAM.

6.1 Evaluation of the Updatable ORE Scheme

For completeness, we start by comparing our updatable ORE scheme with the non-updatable scheme from Chenette et al. [10]. The benchmark results can be seen in Table 3.

For encryption, our scheme is two orders of magnitude slower, because they only need to evaluate a PRF, while evaluating the key-homomorphic PRF in our case requires a scalar multiplication in the group. Updating a ciphertext is roughly as fast as encryption with the running time of both operations being dominated by the scalar multiplication.

Comparisons using our updatable scheme are three orders of magnitude slower than with the non-updatable scheme because the non-updatable scheme requires no cryptographic operations, while we require one group operation per comparison. This translates over to comparison-based sorting. Radix sort offers a running time improvement of one order of magnitude, because – in contrast to comparison-based sorting, where one group operation is required per comparison per element – only one group operation is required per element in total. Radix sort does not offer a performance improvement with the non-updatable scheme, because the running time is not dominated by comparing the elements (or bits thereof), but by the sorting algorithm itself.

[4] `https://github.com/kastel-security/ORE-Decision-Tree`

Table 4: Benchmark results of the protocol on the MNIST dataset and modified versions of the Boston Housing and Titanic datasets, and a custom dataset comparable to the one used by [16], with and without 50 ms of network latency.

Dataset	#Attributes	Compute Threads per party	Dataset Size A	Dataset Size B	Computation time	Network traffic A ← B	Network traffic A → B
MNIST	784	1	100	100	243.9 s	21.7 MB	43.4 MB
		16	100	100	22.6 s	21.7 MB	43.4 MB
		16	500	500	106.3 s	108.6 MB	217.2 MB
		16	500	1000	180.6 s	217.2 MB	325.8 MB
		16	1000	500	135.1 s	108.6 MB	325.8 MB
		16	1000	1000	213.6 s	217.2 MB	434.3 MB
Boston Housing	7	1	253	253	23.2 s	1.9 MB	3.6 MB
		16	253	253	3.7 s	1.9 MB	3.6 MB
Titanic	5	1	357	357	23.8 s	2.1 MB	3.6 MB
		16	357	357	5.0 s	2.1 MB	3.6 MB
Custom	7	16	4096	4096	20.1 s	8.8 MB	15.9 MB
Custom (with latency)		16	4096	4096	20.2 s	8.8 MB	15.9 MB

6.2 Evaluation of the Protocol

For our experiments of the training protocol, we used the unmodified MNIST dataset using all 784 attributes and datapoints of all 10 labels. We also ran experiments on the Boston Housing and Titanic dataset[5], which we modified to ignore entries with `null`-attributes, as well as discrete attributes. As required for decision tree training, we also discretized the labels. This leaves seven attributes in the Boston Housing dataset and five attributes in the Titanic dataset.

In the MNIST dataset, attribute values are 8 bit unsigned integers. In the other datasets, we converted all attributes to integers by taking their 32 bit IEEE 754 representation, reinterpreting it as a 32 bit unsigned integer and dropping the last bit, therefore obtaining an unsigned 31-bit integer, which preserves the order of all positive floating point numbers, including "+0". We used the first n_{A} datapoints as training data for A and the last n_{B} datapoints as training data for B. We used the training algorithm in Algorithm 1 with the Information Gain heuristic. As the training algorithm used here is the same as for plaintext training, the accuracy of the trained model is identical to a model trained on the same data in plaintext. The results can be seen in Table 4 and Fig. 6.

As we can see, the protocol is viable, both computationally, as well as from a network traffic perspective. We can also see that the effects of the dataset size of A are less significant than the effects of the dataset size of B, as B's data needs to be processed three times – when encrypting, when updating, when reverse updating – whereas A's data only needs to be processed once during encryption. A similar argument holds for the network traffic. While we did run the experiments on the same machine, we expect the performance to only differ insignificantly from our test results when run on separate machines over a LAN or WAN. This is because we only have three rounds of interaction, so the effects

[5] The datasets are available on `https://www.kaggle.com/`.

of network latency are insignificant. Additionally, the total network traffic is well below the limits of a normal internet uplink.

Running time comparison with Hamada et al. [16] To compare our results with the results of [16], the current state-of-the-art in MPC-based decision tree training, we generate a dataset consisting of 2^{13} samples with 11 attributes each, which results in a trained decision tree of depth 42. On this dataset, our protocol takes 20.1 s. In [16], they have performed a benchmark on a dataset of the same size and amount of attributes and a tree depth of 40 on a machine comparable to ours. In this scenario, their protocol takes 43.61 s, which is slower than our protocol by a factor of ≈ 2.

When adding 50 ms of artificial network latency, the running time of their protocol increases to 4821.56 s, which is caused by the many rounds of interaction in their protocol. In contrast to this, when adding the same artificial network latency to our protocol, the running time does not change noticeably, still only requiring 20.2 s, because our protocol only consists of three rounds. In this scenario, our protocol is faster by several orders of magnitude.

Running time comparison with Abspoel, Escudero, and Volgushev [2] In [2], the authors did not measure the running time of the entire decision tree training algorithm, but only extrapolated its runtime based on benchmarks of its basic operations. To compare the runtime performance of their approach with ours, we use their extrapolation formula for their runtime to a setting, for which we have have benchmark results with our approach. They use

$$T(N, m, \Delta) \approx m \cdot (S(N) + (2^{\Delta} - 1)I(N) + 2^{\Delta}L(N))$$

to estimate their runtime, where N, m and Δ are the number of training data, the amount of attributes and the maximum depth of the decision tree and $S(N), I(N)$ and $L(N)$ are the time for sorting, and computing an inner or leaf node on N training data points.

We use this formula to estimate the runtime of their approach to the titanic dataset, where $N = 357, m = 5$ and the depth of the resulting decision tree is $\Delta = 25$. Using the optimistic values $S(256) = 0.392$ s, $I(256) = 0.127$ s and $L(256) = 0.004$ s, we obtain a runtime estimate of $\approx 2.2 \cdot 10^7$ s in the passively secure setting. This is several orders of magnitude slower than with our approach, which is mostly caused by their training algorithm having exponential runtime in the tree depth.

Limiting the depth of the decision tree to $\Delta = 10$ only affects the runtime of our approach insignificantly, as the majority of the runtime comes from encrypting and updating the training data. Their approach, however, is significantly sped up by this change, estimated to only have a runtime of ≈ 672 s, which is still significantly slower than our approach.

Applicability comparison with [16, 2] While our protocol solely relies on Order-Revealing Encryption, both of the protocols from [16, 2] are built on top of

generic MPC primitives. Therefore, future advances in (u)ORE or general purpose MPC respectively, will lead to performance improvements of these protocols.

Due to the asymmetric nature of our protocol, it is fixed for the two-party setting with one passive corruption. In addition to this setting, Hamada et al. [16] and Abspoel, Escudero, and Volgushev [2] state that their protocols can fulfill different trust models, such as two-out-of-three corruptions, if the underlying protocol for these MPC primitives is chosen accordingly, however this may cause additional computational/network overhead.

While their protocol can only be applied to the training algorithms they consider, our protocol can generically use any decision tree training algorithm that adheres to the limitations of being deterministic and only requiring comparisons on the training data. Indeed this is even covered by our security proof.

7 Conclusion

We have constructed an Updatable Order-Revealing Encryption scheme, which allows to update ciphertexts from one key to another using a key-homomorphic PRF. This construction is secure under the same leakage function as established ORE schemes, leaking the order of the encrypted messages, as well as the position of the most significant bit in which they differ.

Using such an Updatable ORE scheme, we have constructed a passively secure protocol that allows for securely training a decision tree on two parties' inputs without revealing the inputs to the other party. This protocol can either be used by itself or can be used as a building block to train a decision forest.

We have experimentally verified this decision tree protocol and are able to compute a decision tree on the Titanic dataset, equally partitioned between both parties, within 5.0 seconds. The experiments have also shown that this approach is faster than the current state-of-the-art approaches [2, 16] and orders of magnitude faster when considering network latency or training high-depth decision trees. However, this speedup comes at the cost of some information leakage.

Analyzing the leakage of the ORE scheme, we have found that while it is significant, it is also hides a large proportion of the training data. This provides us with an interesting trade-off between security and efficiency: We leak more information but are faster than relying entirely on MPC to train a decision tree, but we are more secure, but less efficient than performing training in plaintext. We have also found that the proportional information leaked is larger on low-entropy attributes than on high entropy attributes. Whether this leakage is acceptable needs to be decided for each usecase individually. To further reduce the leakage, we show how this approach can also be used in a secure enclave, reducing the leakage even more in a graceful degradation manner, even in the presence of low-entropy attributes.

This leaves us with a special-purpose protocol for decision tree learning that is more performant than generic solutions in a scenario that fits its limitations:

- The protocol has some information leakage and can only be used in a scenario, in which this is acceptable.
- The concrete algorithm for training needs to fulfill some constraints:
  - The training algorithm needs to be deterministic.
  - The split heuristic needs to be evaluated based on comparisons only.
  - As described the protocol allows training between exactly two parties.
  - For training forests: The partitioning of each party's data needs to be independent of the data of the other party.
- The protocol requires more computation compared to the number of communication rounds, so its strength shows better in a higher-latency setting.

While some leakage is inherent with this approach, this also leads us to two interesting open questions:

- How to construct an Updatable ORE scheme with a smaller leakage or in the left-right framework of [23]?
- How to devise an efficient actively-secure protocol based on Updatable ORE schemes?

Acknowledgements

We thank the anonymous reviewers for their helpful and constructive feedback. This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. Robin Berger: This work was supported by funding from SAP Security Research. Felix Dörre: This work was supported by funding by the German Federal Ministry of Education and Research (BMBF) under the project "VE-ASCOT" (ID 16ME0275). Alexander Koch: This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

References

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[2] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. "Secure training of decision trees with continuous attributes". In: 2021.1 (Jan. 2021), pp. 167–187. DOI: `10.2478/popets-2021-0010`.

[3] Adi Akavia, Max Leibovich, Yehezkel S. Resheff, Roey Ron, Moni Shahar, and Margarita Vald. "Privacy-Preserving Decision Trees Training and Prediction". In: *ACM Trans. Priv. Secur.* 25.3 (2022), 24:1–24:30. DOI: `10.1145/3517197`. URL: `https://doi.org/10.1145/3517197`.

[4] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. "A comparative analysis of gradient boosting algorithms". In: *Artificial Intelligence Review* 54 (2021), pp. 1937–1967.

[5] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. "Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation". In: 2015, pp. 563–594. DOI: 10.1007/978-3-662-46803-6_19.

[6] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[7] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. https://eprint.iacr.org/2000/067. 2000.

[8] Kamalika Chaudhuri and Claire Monteleoni. "Privacy-preserving logistic regression". In: *Advances in neural information processing systems* 21 (2008).

[9] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[10] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. "Practical Order-Revealing Encryption with Limited Leakage". In: 2016, pp. 474–493. DOI: 10.1007/978-3-662-52993-5_24.

[11] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. "SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering". In: 2022, pp. 563–577. DOI: 10.1145/3548606.3560702.

[12] Wenliang Du and Zhijun Zhan. "Building decision tree classifier on private data". In: (2002).

[13] F. Betül Durak, Thomas M. DuBuisson, and David Cash. "What Else is Revealed by Order-Revealing Encryption?" In: 2016, pp. 1155–1166. DOI: 10.1145/2976749.2978379.

[14] Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, Celia Kherfallah, Benoît Chevallier-Mames, and Arthur Meyre. *Privacy-Preserving Tree-Based Inference with Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2023/258. https://eprint.iacr.org/2023/258. 2023.

[15] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. "Leakage-Abuse Attacks against Order-Revealing Encryption". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 655–672. DOI: 10.1109/SP.2017.44.

[16] Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Koji Chida. "Efficient decision tree training with new data structure for secure multi-party computation". In: 2023.1 (Jan. 2023), pp. 343–364. DOI: 10.56553/popets-2023-0021.

[17] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. "Practical Secure Decision Tree Learning in a Teletreatment Application". In: 2014, pp. 179–194. DOI: 10.1007/978-3-662-45472-5_12.

[18] Mireya Jurado, Catuscia Palamidessi, and Geoffrey Smith. "A Formal Information-Theoretic Leakage Analysis of Order-Revealing Encryption". In: 2021, pp. 1–16. DOI: 10.1109/CSF51468.2021.00046.

[19] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: 2020, pp. 1575–1590. DOI: 10.1145/3372297.3417872.

[20] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. "Crypten: Secure multi-party computation meets machine learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4961–4973.

[21] Miroslav Kubat and JA Kubat. *An introduction to machine learning*. Vol. 2. Springer, 2017.

[22] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network". In: *IEEE Access* 10 (2022), pp. 30039–30054.

[23] Kevin Lewi and David J. Wu. "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds". In: 2016, pp. 1167–1178. DOI: 10.1145/2976749.2978376.

[24] Yuan Li, Hongbing Wang, and Yunlei Zhao. "Delegatable Order-Revealing Encryption". In: 2019, pp. 134–147. DOI: 10.1145/3321705.3329829.

[25] Yehuda Lindell and Benny Pinkas. "Privacy Preserving Data Mining". In: 2000, pp. 36–54. DOI: 10.1007/3-540-44598-6_3.

[26] Ximeng Liu, Robert H. Deng, Kim-Kwang Raymond Choo, and Jian Weng. "An Efficient Privacy-Preserving Outsourced Calculation Toolkit With Multiple Keys". In: *IEEE Transactions on Information Forensics and Security* 11.11 (2016), pp. 2401–2414. DOI: 10.1109/TIFS.2016.2573770.

[27] Chunyang Lv, Jianfeng Wang, Shi-Feng Sun, Yunling Wang, Saiyu Qi, and Xiaofeng Chen. "Towards Practical Multi-Client Order-Revealing Encryption: Improvement and Application". In: *IEEE Transactions on Dependable and Secure Computing* (2023).

[28] Moni Naor, Benny Pinkas, and Omer Reingold. "Distributed Pseudo-random Functions and KDCs". In: 1999, pp. 327–346. DOI: 10.1007/3-540-48910-X_23.

[29] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. "Oblivious Multi-Party Machine Learning on Trusted Processors". In: 2016, pp. 619–636.

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

[31] Suryakanthi Tangirala. "Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm". In: *International Journal of Advanced Computer Science and Applications* 11.2 (2020), pp. 612–619.

[32] Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A. Scott Patterson. "Privacy-preserving decision trees over vertically partitioned data". In: *ACM Trans. Knowl. Discov. Data* 2.3 (2008), 14:1–14:27. DOI: 10.

1145/1409620.1409624. URL: https://doi.org/10.1145/1409620.1409624.

Appendix

A A brief introduction to the UC framework

In the following, we give a brief introduction to into the Universal Composability framework by Canetti [7], tailored to our usecase. As the framework is quite complex, we omit any details that are not relevant for our work.

The UC model extends the notion of the real-ideal paradigm, where the security of a protocol is defined through some ideal functionality, that captures the computation to be done and is secure by definition.

All parties are modeled as an interactive PPT machines. In addition to parties existing in the protocol, UC execution is defined with two additional entities, namely the environment and the adversary, which are modeled in the same way.

The adversary can corrupt any subset of parties. Considering passive security, the adversary can see the view of parties it corrupts (including all internal state, randomness, incoming and outgoing messages), but it cannot make corrupted parties deviate from the protocol. If it accesses variables from the internal state of a corrupted party, we say it *extracts* this information.

The environment selects inputs for honest parties and receives their outputs. Additionally, it can freely interact with the adversary, sending and receiving arbitrary messages.

To prove the security of a protocol, UC uses the notion of protocol emulation. We say a protocol π in the real world securely realizes an ideal functionality $\mathcal{F}$ in the ideal world, if for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that no environment can distinguish between an interaction with π and $\mathcal{A}$ in the real world and an interaction with $\mathcal{F}$ and $\mathcal{S}$ in the ideal world. This can be done by constructing a simulator for each $\mathcal{A}$, which internally runs $\mathcal{A}$ and translates ideal messages from/to the ideal functionality and protocol messages from/to corrupted parties. Additionally, it is sufficient to only consider the dummy adversary, that sends all protocol messages it receives to the environment and sends any messages it receives from the environment as protocol messages. In the real world, the honest parties execute the protocol and the environment can interact with them using the real adversary. In the ideal world, the input of honest parties is directly sent to the ideal functionality and the output of the ideal functionality to the honest parties is directly outputted by them.

If a protocol π is proven to realize an ideal functionality $\mathcal{F}$, all security properties from $\mathcal{F}$ carry over to π, as this could otherwise be used to distinguish the real and ideal execution.

In UC, the universal composition theorem says that if a protocol is proven to realize an ideal functionality, it remains secure under universal composition. Therefore, it can for example be run in parallel, concurrently or as a subprotocol

to other protocols without becoming insecure. If a protocol π' realizes a functionality $\mathcal{F}'$ using $\mathcal{F}$ as a building block, we say π' realizes $\mathcal{F}'$ in the $\mathcal{F}$-hybrid model. Due to the universal composition theorem, π' still realizes $\mathcal{F}'$, even after $\mathcal{F}$ is instantiated with a protocol that securely realizes $\mathcal{F}$.