PROBONITE : PRivate One-Branch-Only Non-Interactive decision Tree Evaluation

Sofiane Azogagh, Victor Delfour, Sébastien Gambs, Marc-Olivier Killijian
{azogagh.sofiane,delfour.victor}@courrier.uqam.ca
{gambs.sebastien,killijian.marc-olivier2}@uqam.ca
Université du Québec à Montréal (UQAM)
Montréal, Canada

ABSTRACT

Decision trees are among the most widespread machine learning model used for data classification, in particular due to their interpretability that makes it easy to explain their prediction. In this paper, we propose a novel solution for the private classification of a client request in a non-interactive manner. In contrast to existing solutions to this problem, which are either interactive or require evaluating all the branches of the decision tree, our approach only evaluates a single branch of the tree. Our protocol is based on two primitives that we also introduce in this paper and that maybe of independent interest : *Blind Node Selection* and *Blind Array Access*. Those contributions are based on recent advances in homomorphic cryptography, such as the functional bootstrapping mechanism recently proposed for the Fully Homomorphic Encryption over the Torus scheme TFHE. Our private decision tree evaluation algorithm is highly efficient as it requires only one round of communication and d comparisons, with d being the depth of the tree, while other state-of-the-art non-interactive protocols need 2^d comparisons.

KEYWORDS

Private Decision Tree Evaluation, Homomorphic Encryption, Functional Bootstrapping, Blind Array Access, Machine Learning Security and Privacy.

1 INTRODUCTION

Machine learning (ML) is becoming ubiquitous as it is used in many domains of our daily lives for tasks such as the attribution of a loan, recommendation of a restaurant or a place to visit matching the interest of a user, predicting the risk of recidivism if a conditional release from a prison sentence is granted, etc. Most of these applications of machine learning involve the use of private or personal information, both during the training phase and at inference time. Thus, the security and privacy of machine learning has grown as a major research domain that encompasses multiple issues ranging from the confidentiality of the inference data to integrity of the classification, through the protection of privacy of the model or of the data used to train it [30].

Decision trees are a specific type of machine learning model used in a wide range of applications such as medical diagnostics, insurance ruling, credit attributions, etc. On one hand, those applications typically imply the use of sensitive data that users would prefer to keep private rather than sharing it with an untrusted classification server. On the other hand, the server might not want to share its model with the clients for reasons such as protecting this model against privacy attacks such as membership inference or model inversion attacks, or simply to preserve its intellectual property. Despite those privacy and confidentiality issues, both the clients and the server might wish to cooperate in order for the clients to get access to a valuable inference service provided by the server. A typical application is the well-known Machine Learning as a Service (MLaaS) which allows a user to pays for performing Machine Learning models without dealing with complicated training or inference procedures.

Compare to so-called ”black boxes models”, such as deep neural networks, decision trees have several advantages. First, they are simpler to train and their predictions can easily be explained to a human. This makes them a good candidate for the field of explainable AI. Indeed, predictions made by ML algorithms can be of uttermost importance in the life of some individuals if they are put into action in decision-making systems impacting them. They can also be used as a surrogate model in post hoc explanation techniques to explain a more complicated model, such as a neural network for example. Decision trees are also a building block for random forest [5], which is a classification model composed of several decision trees trained on different subsets of the training data. The result of the classification by the random forest consists in an aggregation of the predictions of the different trees. This approach can be used to mitigate the tendency of decision trees to overfit while boosting the accuracy. Private random forest evaluation naturally requires the capacity to privately evaluate each decision tree independently [36].

In the literature, the issue of evaluating a decision tree on private data is addressed via different approaches. The first approach, which is also the most trivial, consists in the client downloading the decision tree from the server and performing the evaluation locally on his own data. While this solution perfectly preserve the privacy of the clients’ data, this comes at the cost of the confidentiality of the server’s model. The second approach requires the client and the server to cooperate through an interactive protocol to securely evaluate the tree. This type of approach is based on secure two-party computing and advanced cryptographic primitives, such as oblivious transfer [31], secret sharing [32], oblivious RAM [20], etc. With this approach, the privacy of the client’s data and the confidentiality of the server’s model are guaranteed at the cost of multiple rounds of communication, usually linear in the depth of the tree. This level of interactivity is not realistic for some application scenarios, such as when the client has very little or no connectivity after submitting its request. In the third approach, the server performs a homomorphic evaluation of the decision tree on ciphered data. The confidentiality of both the client and the server is ensured due to the properties of homomorphic cryptography and this type of approach typically implies the evaluation of all the branches of the decision tree, which computing cost is exponential in the depth of the tree. While this can be circumvented by evaluating several clients requests in parallel with ciphertext packing techniques, this is not always possible for all application settings.

To address this issue, in this paper we propose PROBONITE, which stands for PRivate One-Branch-Only (OBO) Non-Interactive decision Tree Evaluation. Evaluating privately a tree inherently implies that the server has to consider all the nodes, or else he would trivially learn which nodes were not used and so learn information on the client's attributes. All the previous non-interactive PDTE schemes have dealt with this issue by performing the comparisons between the thresholds and the attributes of profile considered for all the nodes. Our main contribution consists in reducing the number of comparisons to its bare minimum and managing the integration of all nodes in our evaluation differently. PROBONITE is, to the best of our knowledge, the first protocol that performs a private inference on a complete binary decision tree with both a single-round of communication between the client and the server and the evaluation of a single branch of the tree. To realize this, PROBONITE is based on homomorphic encryption and specifically on functional bootstrapping, a technique recently introduced in the Fully Homomorphic Encryption over the Torus scheme TFHE [9] to implement non-linear functions or lookup-tables.

The outline of this paper is as follows. First in Section 2, we review the related work, which is either based on secure multi-party computation, thus requiring several rounds of communication, or based on homomorphic encryption, which implied, up to now, the evaluation of all the branches of the decision tree. Afterwards in Section 3, we introduce the notations, the fully homomorphic encryption scheme we use and its building blocks as well as some higher level primitives that we are going to use in PROBONITE protocol : the functional bootstrapping, private information retrieval and private comparison. Then, in Section 4 we detail the main contributions of this article : namely the two primitives for private computing, *Blind Node Selection* and *Blind Array Access* and the PROBONITE protocol, whose performance is evaluated in Section 5. Finally, we conclude by discussing future work in Section 6.

2 RELATED WORK

The problem of protecting the privacy of both inference data and model while performing a classification task using a decision tree is coined Private Decision Tree Evaluation (PDTE) and is addressed by several recent related work, such as [1, 3, 25, 34–36]. Among these different works, we can distinguish interactive approaches from non-interactive ones. Typically, an interactive approach is based on secure multiparty computation techniques, such as secret-sharing, garbled circuits or oblivious transfer, to compare the attributes of the profile considered to the thresholds of the nodes and evaluate the appropriate branch of the tree.

Non-interactive approaches are usually based on homomorphic encryption and they require the server to obliviously and simultaneously evaluates all the branches of the decision tree to construct a private reply containing the inferred class. Interactive approaches are usually more computationally efficient than non-interactive ones, but imply much more communication. Hereafter, we review first the major interactive solutions of the state-of-the-art in PDTE and then the non-interactive ones.

In [3], the decision tree is represented as a multivariate polynomial, in which the variables are the decision bits resulting of the comparison, at each node, of the attribute of the client and the threshold of the server. Henceforth, for each node of the tree, the client and the server engage in an interactive comparison protocol to obtain the corresponding comparison bit. Afterwards, the multivariate polynomial is then homomorphically evaluated on the vector of the comparison bits to produce the model's prediction.

Wu and co-authors [36] have designed an interactive protocol with a constant number of rounds in contrast to the previous works that were in $O(d)$, with d the depth of the tree. This protocol only uses additive homomorphic operations. In a nutshell, the client first gets the class index and in a last round of communication performs an Oblivious Transfer (OT) to obtain the predicted class.

In [35], the server interactively evaluates the client's input using d comparisons. Each comparison enables both the client and the server to obliviously select the next node to evaluate without any party gaining information on the selected node. This is done using either OT for small trees or Oblivious RAM (ORAM) for larger trees. The proposed solution is computationally efficient but requires d rounds of communication, which might be impractical in some situations, especially for disconnected clients or those with limited bandwidth.

The first single-round protocol for evaluating a decision tree was designed by Lu and collaborators in [25] and is related to a non-interactive comparison protocol called XCMP [26]. This protocol is efficient for small inputs thanks to a small multiplicative depth. However, it suffers from several drawbacks. Indeed, the schemes cannot be expanded for larger inputs, the output length is exponential in the tree's depth, do not support Single Instruction Multiple Data (SIMD) operations. Another proposal [34] evaluates a complete decision tree using fully homomorphic encryption with packing techniques. This enables to simultaneously evaluate the decision tree for several attribute vectors, resulting in a lower amortized time.

Akavia and collaborators in [1] also proposed a protocol tailored for training and classifying using a decision tree over homomorphic encryption (using the CKKS scheme) by introducing a polynomial dedicated to the approximation of the step function. Their work result in an efficient non-interactive prediction scheme and also a training phase with a small use of the client computational resources. Nonetheless, this method requires evaluating all the nodes of the tree and to build a reply in the form of an encrypted vector of the classes' scores. This vector has to be decrypted by the client before he selects the class with the highest score.

More recently, Cong and co-authors [15] have proposed a clever approach to reduce the cost of comparisons. More precisely, their protocol exploit the observation that the server knows the threshold values in the clear, and the ciphered values of attributes and that homomorphic comparisons between ciphertexts and plaintexts are cheaper than ciphertext-to-ciphertext comparisons. However, this protocol also requires evaluating all the nodes of the tree and henceforth leads to more comparisons than interactive solutions that evaluate only one branch of the tree.

In this work, we also propose to use homomorphic encryption, and more specifically we leverage a new mechanism introduced in TFHE and called *Functional Bootstrapping*, to provide a single-round protocol that evaluates a single branch of the decision tree, similarly to what would be done by a server accessing to the data in the clear. Private comparisons are usually the most expensive operation in oblivious computing, henceforth evaluating a single branch of the tree provides great benefits in terms of computation requirements. Interactive protocols imply great communication costs that PROBONITE avoids by not requiring communication between the client and the server beyond the client sending its request composed of a vector of ciphered attributes and the server sending the ciphered inferred class.

3 PRELIMINARIES

In this section, we review the background notions and building blocks necessary to understand PROBONITE. Due to space constraint, our objective is not to dive deep into all the details for each building block but references are provided for the interested reader.

3.1 Notations

Let $\mathcal{M}$ be the set of messages, we denote the encryption in TFHE of a message $m \in \mathcal{M}$ by $[\![m]\!]$ and the size of $\mathcal{M}$ by p. For N a power of 2, $\mathcal{R}$ is the quotient ring $\mathbb{Z}[X]/(X^N+1)$ and $\mathcal{R}_q$ is the same ring but modulo q, *i.e* $\mathbb{Z}_q[X]/(X^N+1)$. The set $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ is denoted as the Real Torus and it represents the set of real numbers modulo 1. In the remainder of this paper, the operations are performed in the ring $\mathcal{R}_q$ unless otherwise specified.

3.2 The TFHE Encryption Scheme

The TFHE encryption scheme was proposed in 2016 [9, 10] and its security is based on the so-called Learning With Errors (LWE) problem on lattice and its ring version, the Ring-LWE (RLWE) problem. In TFHE, there exists three types of ciphertexts : LWE, RLWE and RGSW, which are defined hereafter.

Definition (General LWE -GLWE- ciphertexts)**.** A message $m \in \mathcal{M}$ can be encrypted under the secret key $s = (s_0, \ldots, s_{k-1}) \leftarrow_\$ \mathcal{R}_q^k$ as a GLWE ciphertext (a, b) if $a = (a_0, \ldots, a_{k-1}) \leftarrow_\$ \mathcal{R}_q^k$ and $b = \sum_{i=0}^{k-1} a_i \cdot s_i + \Delta m + e$, in which $\Delta = q/p$ and e is sampled from a Gaussian Distribution. More specifically, when $N = 1$ the ciphertext is called a LWE ciphertext and when $k = 1$ and $N > 1$ it is called a RLWE ciphertext. In other words, a LWE ciphertext encrypts a message in $\mathbb{Z}_q$ while a RLWE ciphertext encrypts a polynomial in $\mathbb{Z}_q[X]$ modulo $X^N + 1$.

Definition (General GSW -GGSW- ciphertexts)**.** A message $m \in \mathcal{M}$ can be encrypted under the secret key $s = (s_0, \ldots, s_{k-1}) \leftarrow_\$ \mathcal{R}_q^k$ as a GGSW [18] ciphertext $C = (C_1, \ldots, C_{k-1}, C_k)$. In C each C_i for $i \in \{0, \ldots, k-1\}$ has to be a decomposition vector of GLWE ciphertexts encrypting $-s_i m$ in a certain base $(\Delta_1, \ldots, \Delta_l)$ in which $l = O(log(q))$, while C_k is a decomposition vector of GLWE ciphertexts encrypting m in the same base. Similarly, as the above GLWE definition, when $N = 1$ the ciphertext is named a GSW ciphertext and when $k = 1$ and $N > 1$ it is called a Ring GSW (RGSW) ciphertext.

The TFHE cryptosystem uses several algorithms that we will briefly define here:

- **Modulus Switching** is used to convert a LWE ciphertext $(a, b) \in \mathcal{R}_q^{k+1}$ into a LWE ciphertext $(a', b') \in \mathcal{R}_{2N}^{k+1}$ such that $b' = \sum_{i=0}^{k-1} a'_i \cdot s_i + \Delta' m + e'$, in which $\Delta' = 2N/p$.
- **CMux Gate** is an operation used to select between two GLWE ciphertexts $[\![d_0]\!]$ and $[\![d_1]\!]$ depending on a GGSW ciphertext $[\![b]\!]$, which is a bit selector. The output of the CMux operation is a GLWE ciphertext $[\![d_b]\!]$.
- **Blind Rotation** is used to privately rotate by $[\![i]\!]$ the coefficients in the polynomial $M(X)$ encrypted as a RLWE ciphertext. This is done by using a loop of CMux operations in which the bit selector at each step is a bit from the binary decomposition of i encrypted as a GGSW ciphertext (*cf.*, Algorithm 4 in [11]).
- **Sample Extraction** is used to extract a coefficient of a polynomial $M(X) = \sum_{i=0}^{N-1} m_i X^i$ encrypted as a RLWE ciphertext. The output of this operation is a LWE ciphertext $[\![m_j]\!]$. This operation consists in building the LWE ciphertext output by copying some coefficients of the RLWE ciphertext inputs.
- **Key switching** is used to blindly switch the current secret key to a different secret key. Intuitively, this is done by cancelling the secret key and homomorphically re-encrypting the result with another one. This operation can also be used to switch between different types of ciphertexts, for instance from LWE to RLWE or from many-LWE to one RLWE (*cf.* Algorithm 2 in [11] for more details).

All of these algorithms are used in the operation called *bootstrapping*. In a nutshell, bootstrapping enables to reduce the noise in a ciphertext. Indeed, after several operations (additions or multiplications) are performed on a ciphertext, the noise e, defined in the GLWE ciphertext definition, grows and may overlap with the encrypted message, which would then result in an incorrect decryption later. To remediate to this issue, bootstrapping is applied before the noise of a ciphertext grows too much, to be able to continue to compute on this refreshed ciphertext.

3.3 TFHE's Functional Bootstrapping

The TFHE bootstrapping operation is different from other bootstrapping existing in the literature on at least two main points: first it is fast, taking approximately 18 ms in the first release of the TFHE library [12], but most importantly for our concerns, it allows evaluating a function at the same time as it performs its initial role of noise reduction. This latter property is called *functional bootstrapping*, and the idea behind this technique is to rely on a Look Up Table (LUT).

Definition (Look Up Table)**.** Let f be a function from $\mathbb{Z}_p$ to $\mathbb{Z}_p$, the Look Up Table of f is an array defined as $LUT(f) = (i, f(i))_{i \in [0,N-1]}$. Moreover, a LUT may be encoded into a polynomial $L(X)$ by mapping each element of the LUT to a sequence of many consecutive coefficients of L, which is called the redundancy of the LUT.

The bootstrapping operation in TFHE takes as input a LWE ciphertext $[\![m]\!]$ under a secret key s, a polynomial LUT $L(X)$ and a bootstrapping key BK, which is a GGSW ciphertext of the secret key s. The first step of the bootstrapping operation consists in performing a Modulus Switching on the LWE ciphertext $[\![m]\!]$ to switch $[\![m]\!]$ from modulo q to modulo $2N$, which produces a new LWE ciphertext $[\![m']\!] = (a', b')$. Afterwards, a Blind Rotation is applied on the polynomial L by using the LWE ciphertext $[\![m']\!]$ and the elements of the bootstrapping key BK as bit selectors of the loop of CMux gate. This operation outputs a GLWE ciphertext of $L(X) \cdot X^{-\Delta' m + e'}$ under a new secret key s' (actually s' is the secret key encrypting s into BK). Then, a Sample Extraction of the constant coefficient of $[\![L(X) \cdot X^{-\Delta' m+e'}]\!]$ gives a LWE ciphertext of $f(m)$ under a part of the secret key s'. Finally, the Key Switching operation is used to get back to the original LWE secret key s. Note that in the basic bootstrapping case, whose goal is only the noise reduction of $[\![m]\!]$, then the polynomial LUT $L(X)$ encodes the identity function (*i.e*, $f = Id_{\mathcal{R}}$). The complete functional bootstrapping procedure is summarized in Algorithm 1.

Algorithm 1 Functional Bootstrapping - FB

```
Input : A LWE ciphertext $[\![m]\!] = (a_0, \dots, a_{k-1}, b) \in \mathcal{R}_q^k + 1$ with
    $m \in \mathcal{R}_p$.
1: A LUT of $f$ encoded into a polynomial $L(X)$.
2: A bootstrapping key $BK = (BK_i)_{i \in [0,k-1]}$ in which each $BK_i$ is
    a GGSW ciphertext encrypting the secret key $s_i$ under a secret
    key $s'$.
Output : A LWE ciphertext $[\![f(m)]\!]$
3: function FB($[\![m]\!]$, $(0, L(X))$, $BK$)
4:     $[\![m]\!] \leftarrow$ ModulusSwitching($[\![m]\!]$)
5:     $acc \leftarrow$ BlindRotation($(0, L(X))$, $[\![m]\!]$, $BK$)
6:     $c' \leftarrow$ SampleExtraction($acc$)            ▷ Encrypted under $s'$
7:     $c \leftarrow$ KeySwitching$_{s' \to s}$($[\![f(m)]\!]$)         ▷ To go back to $s$
8:     return $c$
9: end function
```

Furthermore, the polynomial $L(X)$ is given unencrypted as a trivial GLWE ciphertext (*i.e*, $(0, \dots, 0, L(X))$), but we can encrypt it into a RLWE ciphertext to compute a private LUT with a minimal impact on the noise. We denote this operation as the *private functional bootstrapping.*

3.4 Private Information Retrieval

Private Information Retrieval (PIR) is a protocol that allows a client to retrieve a record from a database without the server knowing which record is concerned [14]. Several approaches can be used to implement a PIR such as sharing the database among mutually distrustful servers to achieve information-theoretic PIRs [29], using trusted hardware [33] or based on homomorphic cryptography [27]. This latter category is referred to as *computational PIR* (cPIR). Typically, cPIR protocols use homomorphic encryption and require the server to process the entire database according to the client request, so that the server cannot learn a single bit of information on the request from this process. In a nutshell, the client sends a request composed of N ciphertexts, all encoding 0 except for one ciphertext encoding a 1 at position i to get the i^{th} record. The server absorbs each record of the database into the client's request using a multiplication between a cipher and the record. This results in a new series of ciphertexts, all encoding 0 but one encoding the i^{th} element. Then these ciphertexts are summed into what constitutes the reply sent to the client, who can then decrypt it and obtain the requested record.

3.5 Homomorphic Comparison Function

In the evaluation of a decision tree, the most important operation is the comparison function used to compare the value of an attribute of the client and the threshold. More precisely, since this information needs to be kept private, the comparison function has to be performed over encrypted data. In PDTE, this comparison is usually the most expansive homomorphic operation, as shown in several recent papers such as [7, 15, 25].

In [25], in order to compare two elements $x, y \in \mathbb{Z}_N$, those have to be encoded in the form of two polynomials, X^x and X^y, encrypted as RLWE ciphertexts. Afterwards, the computation to be performed is the following, $[\![X^x]\!] \cdot [\![X^{-y}]\!] \cdot (-\frac{T(X)}{2}) + \frac{1}{2}$in which $T(X) = \sum_{i=0}^{N-1} X^i$. The constant coefficient is the result of the comparison $y < x$, that is 1 for *true* and 0 for $false$.

The approach proposed in [15] is similar to [25], with the difference that the assumption is made that one input is encrypted while the other is a plaintext. This results in a computationally less demanding algorithm, since the multiplication (ciphertext-ciphertext) is replaced with an absorption (ciphertext-plaintext).

Recently, Charkraborty and Zuber [7] have proposed two approaches to compute the min/argmin of two or more integers. One of these methods relies on the application of two consecutive functional bootstrapping. More precisely, one is used to compute the sign of the difference between the two integers, which is called *Sign Bootstrapping*[1] while the second functional bootstrapping takes as input the output of the previous one plus some constant to compute the min/argmin. The authors claim their method to be the fastest and have compared it to other state-of-the-art techniques [22].

The method of Lu, Zhou and Sakuma [25] requires the client to use a specific encoding of its attributes into the degrees of X, as opposed as in the coefficients for other methods. This makes this method impractical for our setting in which the value of the attributes have to lie in $\mathbb{Z}_N$ as we put them in a LUT to apply our `BlindArrayAccess` algorithm (see Algorithm 3). Moreover, even if the server knows the thresholds used in the different nodes of his decision tree, we need to hide the threshold that is compared with the encrypted attribute. Otherwise, the server could learn which node is currently being processed by observing this threshold. To realize this, the server gets the threshold value from a call to `BlindNodeSelection` algorithm (see Algorithm 5) which returns a ciphered value of the node attributes, including the threshold, similarly to PIR. Thus, we need a ciphertext-to-ciphertext comparison function and the optimization proposed by [15] is not adequate for our setting.

For all these reasons, we use the Sign Bootstrapping algorithm from [4]. To apply it, both parties must first encode the integers that will be compared in $\mathbb{T}$. Indeed, since the Sign Bootstrapping operation works with input in $\mathbb{T}$ and since the threshold and the value of the client's attribute belong to $\mathbb{Z}_N$, we can represent them in the Real Torus $\mathbb{T}$ by partitioning its positive half in N slices.

[1]The Sign Bootstrapping operation was originally introduced in [4] to replace the activation function of a Deep Neural Network. Zuber and Sirdey have also used it in [37] to build a homomorphic k-nearest neighbours classifier.

These slices are separated by the distance between two consecutive integers in the Torus. This approach has been widely used in [4, 6, 23] to represent bounded integers in the Torus. Finally, the complete homomorphic comparison that we use is summarized in Algorithm 2.

Algorithm 2 Homomorphic comparison

```
Input : Two LWE ciphertext $[\![x]\!]$,$[\![y]\!]$.
1: A bootstrapping key $BK$
Output : A LWE ciphertext $[\![b]\!]$ such that $b = 1$ if $x < y$ and 0
   otherwise
2: function CMP($[\![x]\!]$,$[\![y]\!]$,$BK$)
3:     $L(X) \leftarrow \sum_{i=0}^{N} X^i$
4:     $d \leftarrow [\![y]\!] - [\![x]\!]$
5:     $c \leftarrow \mathsf{FB}(d, (0, -L(X)/2), BK)$
6:     return $c + (0, \frac{1}{2})$
7: end function
```

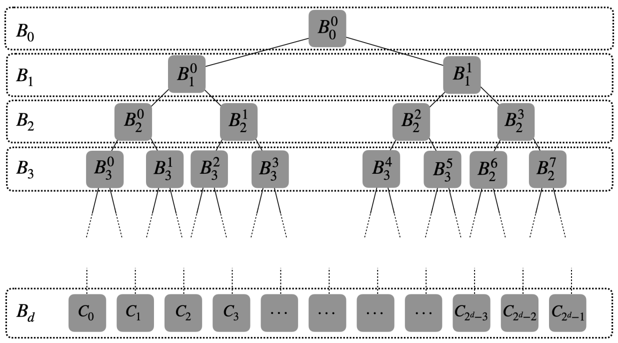

Fig. 1. Representation used for a decision tree of depth d. The level j is named B_j and it contains 2^j elements since the decision tree is complete. Each element B_j^ℓ is a data structure containing 3 elements : the threshold θ_j^ℓ, the index of the attribute to be processed i_j^ℓ and a bit accumulator acc_j^ℓ used to select the next node. The last level B_d (*i.e.*, the leaves of the tree) contains the different classes.

4 PRIVATE DECISION TREE EVALUATION

In the following section, we present the different algorithms that allow our Private Decision Tree Evaluation to have the One Branch Only (OBO) property. First, we will present the different challenges that this property meet and the data structure it implies. Then we describe the two main algorithms, namely Blind Node Selection and Blind Array Access, that are needed to satisfy the OBO property. Finally, we put things together by detailing our PDTE protocol.

4.1 OBO challenges

The main contribution of our protocol consists in reducing the number of comparisons to its minimum, i.e. only evaluating one branch. However, doing the minimum of comparisons requires solving two problems that have not been addressed by previous works. First, we have to select the node to evaluate while being oblivious to the server. To realize this, we have designed a Blind Node Selection based on a PIR construction. This new primitive described in Algorithm 5 enable us to interrogate the required node obliviously thanks to the properties of homomorphic encryption and also because we use all the nodes of this level as illustrated in Figure 3.

The second challenge consists in selecting the attribute without giving the server any some knowledge about it. Except for the situation in which the attribute considered is the same for all the nodes at a particular level, in which case revealing this information enables the server to learn information on the node being currently evaluating by eliminating those that do not treat this particular feature. We address this issue though our Blind Array Access primitive, described in Algorithm 3. In the next subsections, we will these two primitives, which might be of independent interest on their own.

4.2 Impact of OBO on data structures

Given the attribute's vector of the client F whose components are encrypted, evaluating the tree privately while taking only one path through the tree requires that the tree follows a specific structure, which we define in the following. Let view the tree of depth d as a database T in which each level of T is a sub-database B_j as illustrated in Figure 1. We assume that the tree is complete, as otherwise we can always add some dummy internal nodes to the tree to make it complete. Each level B_j contain 2^j elements corresponding to the nodes of the tree, which we denote by B_j^ℓ with $\ell \in \{0, \dots, 2^j - 1\}$. An element of B_j^ℓ is a data structure containing 3 elements $(\theta_j^\ell, i_j^\ell, acc_j^\ell)$, in which θ_j^ℓ is the threshold, i_j^ℓ is the index of the attribute value in the client's attribute vector that will be compared with the threshold and acc_j^ℓ is a bit accumulator determining the next node to select during the evaluation. For the sake of simplicity, we sometimes denote those three elements θ_j^ℓ, i_j^ℓ and acc_j^ℓ respectively as $B_j^\ell.\theta$, $B_j^\ell.index$ and $B_j^\ell.acc$. This tree structure can be implemented by using an array of d elements, in which the j^th element is B_j instantiated as a vector of 2^j nodes. Each node is then linked to their children.

4.3 Blind Array Access

Algorithm 3 Blind Array Access

```
Input : An encrypted array $A = [[\![a_0]\!], ..., [\![a_n]\!]]$.
1: An encrypted index $[\![i]\!]$
2: A bootstrapping key $BK$
Output : The LWE ciphertext $[\![a_i]\!]$
3: function BlindArrayAccess($A$,$[\![i]\!]$, $BK$)
4:     $[\![L(X)]\!] \leftarrow [\![\sum_{j=0}^{n} a_j X^j]\!]$        ▷ many-LWE to RLWE
5:     $c \leftarrow \mathsf{FB}([\![i]\!], [\![L(X)]\!], BK)$
6:     return $c$
7: end function
```

Given an array $A = [[\![a_0]\!], ..., [\![a_p]\!]]$ in which all the elements are encrypted as LWE ciphertexts, we can use the private functional bootstrapping previously defined to privately get the $[\![i]\!]$-th element of A. This can be done by encoding all the elements of the array

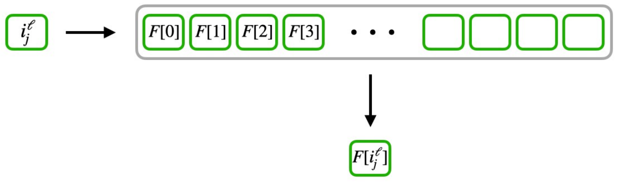

Fig. 2. An illustration of the `BlindArrayAccess` algorithm applied to the ciphered vector of attributes to privately retrieve the value of the attribute at index i_j^ℓ. The components of the vector are rotated to the left until the first component is the encryption of $F[i_j^\ell]$, then a sample extraction is performed to obtain this element.

into a polynomial $L(X)$ encrypted as a RLWE ciphertext by using the Public Functional Key Switching presented in Algorithm 2 of [10]. In a nutshell, this algorithm takes the p LWE ciphertext and fills a polynomial of degree N with $\frac{N}{p}$ of redundancy of each p element. More formally, this algorithm homomorphically computes the following function :

$$\text{many-LWE to RLWE} : (a_0, \ldots, a_p) \rightarrow \sum_{j=0}^{p-1} X^{j\frac{N}{p}} \cdot \sum_{k=0}^{\frac{N}{p}-1} a_j X^k.$$

The problem with this function is that the number of elements is limited because of the redundancy of the LUT. To avoid this, an improvement of `BlindArrayAccess` can be implemented by encoding the binary decomposition of the encrypted index in several RGSW ciphertexts, *i.e.* $[\![i]\!] \rightarrow ([\![i_0]\!], \ldots, [\![i_{log(N)}]\!])_2$ instead of the bootstrapping key BK. Then, during the bootstrapping operation, when the CMux loop is performed, it will take as selector those encrypted bits instead of the element of BK. To realize this, the index i is encoded as a polynomial $I(X) = \sum_{j=0}^{N} i_j X^j$, which is encrypted as a RLWE ciphertext $[\![I(X)]\!]$. Then, the homomorphic expansion algorithm `homExpand` presented in [8] (Algorithm 4) is applied to extract $log(N)$ RGSW ciphertexts encrypting each bit i_j. This improvement is summarized in Algorithm 4.

Algorithm 4 An improvement of Blind Array Access

```
Input : An encrypted array $A = [[\![a_0]\!], ..., [\![a_N]\!]]$.
  1: A RLWE ciphertext encrypting the index $i$ as a polynomial
     $I(X) = \sum_{j=0}^{N} i_j X^j$ in which $i = (i_0, \ldots, i_{log(N)})_2$
  2: A bootstrapping key $BK$
Output : The LWE ciphertext $[\![a_i]\!]$
  3: function BlindArrayAccess2($A$,$[\![I(X)]\!]$, $BK$)
  4:     $[\![L(X)]\!] \leftarrow [\![\sum_{j=0}^{n} a_j X^j]\!]$          ▹ many-LWE to RLWE
  5:     $([\![i_j]\!])_{j\in[0,N]} \leftarrow$ homExpand($I(X)$, $BK$)      ▹ Alg.4 of [8]
  6:     ACC $\leftarrow [\![L(X)]\!]$
  7:     for $j = 0$ to $log(N)$ do                    ▹ Blind rotation adapted
  8:         ACC $\leftarrow$ CMux($[\![i_j]\!]$,ACC,ACC $\cdot X^{-i_j 2^j}$)
  9:     end for
 10:     $c \leftarrow$ SampleExtraction(ACC)
 11:     return $c$
 12: end function
```

4.4 Blind Node Selection

In PROBONITE, we use a cPIR construction in order for the server to blindly select the next node of the decision tree based on the results of previous comparisons. By doing so, the server has no better option than randomly guessing with probability of $\frac{1}{2^j}$ which node is selected at stage j, which is the best we can hope for.

Algorithm 5 Blind Node Selection

```
Input : The array of nodes at level $j$ called $B_j$.
  1: The array of nodes at level $j-1$ called $B_{j-1}$.
  2: The comparison bit $b_{j-1}$.
Output : The $j$-th node as three encrypted ciphertexts
     $(Node.\theta_j, Node.index_j, Node.bit_j)$
  3: function BlindNodeSelection($b_j, B_{j-1}, B_j$)
  4:     $Node_j.\theta = 0$
  5:     $Node_j.index = 0$
  6:     for $i = 0$ to $2^j - 1$ do
  7:         $[\![parent]\!] = [\![B_{j-1}^{\lfloor \frac{i}{2} \rfloor}.acc]\!]$
  8:         if $i\%2 == 0$ then
  9:             $[\![acc]\!] = [\![b_{j-1}]\!] \wedge [\![parent]\!]$
 10:         else
 11:             $[\![acc]\!] = [\![\overline{b_{j-1}}]\!] \wedge [\![parent]\!]$
 12:         end if
 13:         $[\![B_j^i.acc]\!] = [\![acc]\!]$
 14:         $[\![Node_j.\theta]\!] += B_j^i.\theta \cdot [\![acc]\!]$
 15:         $[\![Node_j.index]\!] += B_j^i.index \cdot [\![acc]\!]$
 16:     end for
 17:     return $Node_j$
 18: end function
```

This algorithm is very similar to the baseline solution proposed to retrieve the label (*cf.* Section 5.2). Indeed, at every level of the decision tree, except for the root, we assign a bit to all the nodes at the level currently considered similarly to what the baseline does for the last (*i.e.*, leaves) level. However, the bit associated to the node at level $j-1$ is used to compute the bit assigned to nodes of level j, decreasing the cost of this association to 1 homomorphic AND per node. Afterwards, the nodes are absorbed with their associated bits (recall that only one bit is a 1 for the whole level). Finally, the absorbed nodes are summed up to obtain the node needed to evaluate the tree.

If $Node_j$ represents the node at level j, $b_{0,\ldots,j-1}$ are the resulting bits of previous comparisons, which results in :

- At level 1 : $Node_1 = (b_0)B_1^0 + (\overline{b_0})B_1^1$.
- At level 2 : $Node_2 = (b_0)(b_1)B_2^0 + (b_0)(\overline{b_1})B_2^1 + (\overline{b_0})(b_1)B_2^2 + (\overline{b_0})(\overline{b_1})B_2^3$.
- In general, at level j : $Node_j = \sum_{i=0}^{i=2^j-1} (B_j^i \prod_{k=0}^{k=j-1} b_k^*)$

  in which b_k^* is equal to $\begin{cases} b_k \text{ if the k-th bit of the binary decomposition of i is equal to 0} \\ \overline{b_k} \text{ else} \end{cases}$

This is the general formula of the bits associated to the nodes but, by doing it recursively in Algorithm 5, a lot of AND computations

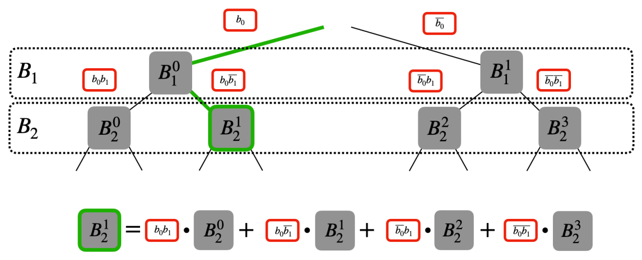
b_0
$\overline{b_0}$
B_1
b_0b_1
B_1^0
$b_0\overline{b_1}$
$\overline{b_0}b_1$
B_1^1
$\overline{b_0}\overline{b_1}$
B_2
B_2^0
B_2^1
B_2^2
B_2^3
$B_2^1 = b_0b_1 \cdot B_2^0 + b_0\overline{b_1} \cdot B_2^1 + \overline{b_0}b_1 \cdot B_2^2 + \overline{b_0}\overline{b_1} \cdot B_2^3$

Fig. 3. An example of our `BlindNodeSelection` applied to level B_2. In this scenario, the node B_2^1 is selected as the result of the comparisons b_0 is True and b_1 is False, so the AND operation between b_0 and $\overline{b_1}$ is True.

are saved, thus gaining in computational time and in memory cost. To further reduce the use of memory, an optimization trick would be to erase the bit information for nodes two levels above the one we are considering. More precisely, as those bits are homomorphic ciphertexts, the memory gain would be non-negligible, at the level j it would save $2^j - 1$ bits compared to an evaluation without this enhancement.

4.5 The PROBONITE protocol

To evaluate the decision tree, the server is assumed to know the attribute it has to process at the root and we additionally assume without loss of generality that it is the first element of the attributes' vector sent by the client (*i.e.*, formally it implies $i_0^0 = 0$). The server starts by homomorphically computing the bit comparison $[\![b_0]\!]$ between $[\![F[i_0^0]]\!]$ and θ_0^0 (the latter is encrypted as a trivial LWE ciphertext, *i.e*, $(0, \theta_0^0)$). Afterwards, he gets the appropriate node of the next level B_1 by running the subroutine `BlindNodeSelection` described in Section 4.4 with the bit selector b_0. This encrypted bit is kept to generate the next bit selector of the following level. Indeed, each internal node is associated with a bit accumulator *acc*, which is an aggregation of the current bit comparison and the previous ones. To realize this, the server has to run an AND gate bootstrapping between the current bit comparison (or its negation depending on the child node) and the accumulator of the parent. This updating of the accumulator is performed after the node selection. More formally, when being at the level B_j for the selection of the next node, each node B_j^ℓ contains the encrypted accumulator acc_ℓ that will be used for the children of the node B_j^ℓ. Let b_j be the result of the comparison of the current node. Then, the selection of the next node is done by computing the following formula:

$$\underbrace{b_j \wedge acc_j^0}_{acc_{j+1}^0} B_{j+1}^0 + \ \dots \ + \underbrace{\overline{b_j} \wedge acc_j^{2^j-1}}_{acc_{j+1}^{2^{j+1}-1}} B_{j+1}^{2^j-1}$$

This algorithm works in the following manner. For the root, as the server knows the decision tree and which attribute shall be compared to a fixed threshold, the server can do this comparison using a plaintext-ciphertext comparison that will generate the bit $[\![b_0]\!]$. Thanks to this bit, the server obtain B_0^0 obliviously using the Blind Node Selection primitive (Algorithm 5).

Algorithm 6 Private Decision Tree Evaluation - PDTE

```
Input : An encrypted attribute vector $F = (F[i])_{i\in[0,N]}$.
 1: A bootstrapping key $BK = [\![s]\!]$. A complete decision tree
    $T = \{B_0, ..., B_d\}$ such that $B_j$ is the $j$-th level of $T$. The lev-
    els $\{B_j\}_{j\in[0,d-1]}$ are each composed by $2^j$ internal nodes
    $B_j = \{B_j^\ell\}_{\ell\in[0,2^j-1]}$ and the last level $B_d$ is composed by $2^d$
    leaves $B_d = \{C_i\}_{i\in[0,2^d-1]}$. Each internal node $B_j^\ell$ is a data
    structure containing the threshold $\theta_j^\ell$ and the associated at-
    tribute index $i_j^\ell$.
Output : A GLWE ciphertext $[\![C_p]\!]$ corresponding to the predicted
    class
 2: function PDTE($F, T, BK$)
 3:     $[\![b_0]\!] \leftarrow$ CMP($\theta_0^0, F[i_0^0]$)
 4:     $Next\_node \leftarrow$ BlindNodeSelection($b_0, B_0, B_1$)
 5:     for $j = 1$ to $d - 1$ do
 6:         $[\![F[i_j]]\!] \leftarrow$ BlindArrayAccess($F, Next\_node.i, BK$)
 7:         $[\![b_j]\!] \leftarrow$ CMP($Next\_node.\theta, [\![F[i_j]]\!], BK$)
 8:         $Next\_node \leftarrow$ BlindNodeSelection($b_j, B_j, B_{j+1}$)
 9:     end for
10:     return $Next\_node$
11: end function
```

For all the other nodes, the server will select the next node up to the label by repeating the following three steps. First, the server wants to get the attribute evaluated at the node value. To realize this, the server only has in his hand the attribute's encrypted index. This information is enough to apply the Blind Array Access primitive (Algorithm 4) and retrieve obliviously the encrypted attribute value. Then, the comparison between the encrypted threshold stored in *current_Node.θ* and the attribute resulting from the previous step can be performed. The output of this comparison is the encrypted bit b_j, which can be used to get the next node from the Blind Node Selection primitive (Algorithm 5). If the next node is the label (*i.e.*, leaf node), the server sends it to the client. Otherwise, it means that it is a regular node and the server repeats the previous steps to get the next node.

5 SECURITY MODEL AND PERFORMANCES COMPARISON

5.1 Security model

The adversary model we consider is the honest-but-curious [28] model in which both the client and the server follow the protocol but try to gain as much information as possible from the observations they gather during the execution of the protocol. On the one hand, the client might be interested to learn information about the model, its parameters, its size or its thresholds. On the other hand, the server might want to obtain information about the client, which in this case would be the description of the attributes' vector. The honest-but-curious model is the one mainly used for all the previous PDTE protocols and to the best of our knowledge, there is no 2-party PDTE scheme that has been designed and proven to be secure against a malicious adversary without the use of universally composable secure protocols against malicious adversaries

[24]. Note that for all PDTE schemes that are non-interactive, including ours, a malicious server cannot learn anything about the client's input. However, he could compromise the integrity of the output computed by sending back a random class (*e.g.*, to decrease the use of his computational resources by not performing some computations or to mislead the client into a wrong inference).

Even an honest-but-curious client learns two information during a PDTE : (1) the tree depth (or at least an upper bound of it) as the computational time is directly related to it and (2) the output corresponding to the evaluation of the decision tree over his input, which precisely corresponds to the functionality provided by the PDTE. Note that possibly, the same client could reconstruct the decision tree by sending a lot of requests to the server, which would enable it to reverse-engineer the decision tree or to retrain it with the data acquired. Note that this leakage cannot be prevented and is inherent to the repeated use of PDTE unless there is a way to limit the number of queries that a particular client can perform.

For instance, PDTE ensuring security against malicious adversaries can be achieved by doing verifiable computation [17], which would put an additional stress on the decision tree security. Indeed, the verification phase must not reveal information on the decision tree to the client yet the client need to know the model to do verifiable computation, which are two antagonist yet not irreconcilable objectives.

5.2 Baseline : Full Tree Evaluation

In this subsection, we will describe one of the simplest and most intuitive method to perform a non-interactive full-tree PDTE by relying on homomorphic operation. In particular, this method has been used previously by Tueno, Boev and Kerschbaum [34] to evaluate decision trees associated to a data structure adapted for ciphertext packing and SIMD evaluation. First, the client sends his attributes' vector to the server. Then, the server do the comparisons for all the nodes in the tree, obtaining an encrypted bit associated to each branch. Then, he computes the bits associated to each leaf by aggregating the bits along the path of the leaf (with bitwise AND). Finally, the server absorbs all the leaves containing the class with their associated bits and sum all of them into one ciphertext whose value is equal to the predicted class, since the others class have been absorbed by 0. He can then send the result back to the client, who will decrypt it and get the prediction wanted. The complexity of this protocol is $O(2^d)$.

5.3 Performances

The following table compare several modern PDTE schemes and their performance. The first column shows the number of rounds used during the protocol, a protocol being non-interactive if and only if the number of rounds is 1. The second and third column are related to the computational complexity of the protocol, the third column shows the global computational complexity while the second only displays the number of comparisons, *i.e.* the most expensive operation. The fourth column represents the communication complexity, while the fifth gives the leakage from the time of execution of the protocol.

As shown in Table 1, PROBONITE achieves the lower bound of comparisons, and it is the first non-interactive PDTE protocol to achieve this. Nonetheless, all the PDTE schemes still have a computational cost of complexity $O(2^d)$, which is far from the complexity of a public decision tree evaluation (*i.e.*, $O(d)$). Unsurprisingly, communication costs are higher for interactive schemes than for non-interactive ones, but the latter usually have lower practical execution speed compared to their non-interactive counterparts. Most of the schemes leak the numbers of nodes $|T|$, while ours leak the depth d of the tree. In all cases, the computational time is related to at least one of those parameters. In particular, the knowledge of $|T|$ reveals more information on the tree structure than the one of d because it leaks the exact number of nodes while letting a user get a good approximation of d. However, the knowledge of d also provides an upper bound on the maximum amount of nodes the tree have. Most previous non-interactive PDTE schemes can be adapted to an incomplete binary tree, which is the reason they leak the number of nodes of the tree instead of the depth. Our scheme has a number of nodes only related to d with as many dummy nodes as necessary to fill the tree to make it complete.

Scheme	Rounds	Comparisons	Computation	Communications	Leakage
[35]	$O(d)$	d	$O(2^d)$	$O(2^d)$	$\|T\|, d$
[3]	$O(d)$	$O(2^d)$	$O(2^d)$	$O(2^d)$	$\|T\|$
[36]	6	$O(2^d)$	$O(2^d)$	$O(2^d)$	$\|T\|$
[25]	1	$O(2^d)$	$O(2^d)$	$O(2^d)$	$\|T\|$
[34]	1	$O(2^d)$	$O(2^d)$	$O(1)$	$\|T\|$
[1]	1	$O(2^d)$	$O(2^d)$	$O(\|L\|)$	$\|T\|$
Ours	1	d	$O(2^d)$	$O(1)$	d

Table 1: Comparison of PDTE protocols where d is the depth,$|T|$ is the number of nodes and $|L|$ is the number of labels.

Scheme	Comparisons	Absorptions	Bitwise AND	Additions	FB
[34]	2^d	2^d	2^d	2^d	0
Ours	d	2^d	2^d	2^d	$d-1$

Table 2: Comparison of computational cost between the baseline non-interactive PDTE used in [34] and our protocol.

Table 2 illustrates the computational trade-off between the baseline non-interactive PDTE used in [34] and our protocol. More precisely, PROBONITE only do d comparisons and $d-1$ functional bootstrapping instead of 2^d comparisons. In practice, this translates in a huge improvement, as comparisons are the most expensive operations used in both protocol. Indeed, in the context of non-leveled TFHE, comparisons are $10n$ times more time-consuming than a bootstrap, for n the number of bits to compare (see [22] for more details). This trade-off leads to an overall lower computational cost while not increasing the communication one. Overall, it decreases the computational cost of comparisons to a linear complexity, only leaving additions, bitwise AND and absorption to an exponential complexity with respect to the tree depth. However, all of these operations are considered much cheaper than comparisons or even bootstrapping.

6 CONCLUSION AND FUTURE WORKS

In this paper, our first main contribution is the proposition of PROBONITE, which to the best of our knowledge, is the first PDTE protocol, which is both non-interactive and that computes one branch only of the decision tree. More precisely, using PROBONITE, a server can classify the client's private data without learning anything about and also without the client gaining information about the decision tree. To build this protocol, our second main contribution is the introduction of two new primitives based on recent homomorphic encryption advances : (1) Blind Array Access, which enables a server to access an array of encrypted data using a private index, and (2) Blind Node Selection, which allows a server to get a ciphered element with a private index from a public database, similarly to PIR. We believe that these two new primitives can be of independent interest to construct other privacy protocols and applications as they enable a server to read arrays or other data structures privately, which make them quite generic. Furthermore, these primitives are much cheaper than their secure multiparty counterpart, namely ORAM protocols, that offer read-write access at the cost of being interactive and more intensive in terms of computation [19, 21].

The comparison of our protocol with other previous works, demonstrates that in terms of rounds, communication and computational complexity, PROBONITE is more efficient than other state-of-the-art protocols, both interactive and non-interactive. This is mostly due to the fact that, by evaluating only one branch of the decision, our protocol only requires d comparisons, whereas other non-interactive protocols require 2^d comparisons. As future works, we plan to implement PROBONITE to experimentally confirm those results. More precisely, we plan to perform this performance evaluation on real datasets from the UCI repository [16], namely heart-disease, housing, spambase and artificial. To benefit from the functional bootstrapping capacities of the TFHE scheme [10], the implementation will be done in TFHE library [12] or on the recent libraries [2, 13].

PROBONITE could benefit from several improvements, that we plan to explore in the near future. For instance, batching could be used in several places of the protocol to boost the performance such as in the Blind Node Selection, to perform several absorptions or additions at the same time, or in the Blind Array Access to select several attributes, for multiples clients, in parallel. In the current version of the protocol, the attributes are encoded in the coefficient of the polynomials, but it would be possible also to encode them in the exponents, which would open the way to the use of more efficient homomorphic comparison algorithms such as XCMP [25]. This encoding could also be beneficial more generally for the efficiency of the Blind Array Access primitive by increasing the maximum size of the arrays (*i.e.*, the number of attributes), it can handle.

ACKNOWLEDGMENTS

REFERENCES

[1] Adi Akavia, Max Leibovich, Yehezkel S. Resheff, Roey Ron, Moni Shahar, and Margarita Vald. 2022. Privacy-Preserving Decision Trees Training and Prediction. *ACM Trans. Priv. Secur.* 25, 3, Article 24 (may 2022), 30 pages. https://doi.org/10.1145/3517197

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. (2022). https://eprint.iacr.org/2022/915 https://eprint.iacr.org/2022/915.

[3] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015.* The Internet Society. https://www.ndss-symposium.org/ndss2015/machine-learning-classification-over-encrypted-data

[4] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III (Lecture Notes in Computer Science)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10993. Springer, 483–512. https://doi.org/10.1007/978-3-319-96878-0_17

[5] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[6] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2018. New techniques for Multi-value input Homomorphic Evaluation and Applications. Cryptology ePrint Archive, Paper 2018/622. (2018). https://eprint.iacr.org/2018/622 https://eprint.iacr.org/2018/622.

[7] Olive Chakraborty and Martin Zuber. 2022. Efficient and Accurate homomorphic comparisons. Cryptology ePrint Archive, Paper 2022/622. (2022). https://eprint.iacr.org/2022/622 https://eprint.iacr.org/2022/622.

[8] Hao Chen, Ilaria Chillotti, and Ling Ren. 2019. Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE. Cryptology ePrint Archive, Paper 2019/736. (2019). https://eprint.iacr.org/2019/736 https://eprint.iacr.org/2019/736.

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I (Lecture Notes in Computer Science)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10031. 3–33. https://doi.org/10.1007/978-3-662-53887-6_1

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2018. TFHE: Fast Fully Homomorphic Encryption over the Torus. *IACR Cryptol. ePrint Arch.* (2018), 421. https://eprint.iacr.org/2018/421

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91.

[12] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. August 2016. TFHE: Fast Fully Homomorphic Encryption Library. (August 2016). https://tfhe.github.io/tfhe/.

[13] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2020. CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE. In *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, Vol. 15.

[14] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *J. ACM* 45, 6 (1998), 965–981. https://doi.org/10.1145/293347.293350

[15] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder Vitor Lima Pereira. Efficient Privacy Preserving Tools based on Homomorphic Encryption. (????). Poster presented at FHE.org Conference 2022, Trondheim, Norway.

[16] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. (2017). http://archive.ics.uci.edu/ml

[17] Dario Fiore, Anca Nitulescu, and David Pointcheval. 2020. Boosting Verifiable Computation on Encrypted Data. In *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.), Vol. 12111. Springer, 124–154. https://doi.org/10.1007/978-3-030-45388-6_5

[18] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part*

I (Lecture Notes in Computer Science), Ran Canetti and Juan A. Garay (Eds.), Vol. 8042. Springer, 75–92. https://doi.org/10.1007/978-3-642-40041-4_5

[19] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 182–194.

[20] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (may 1996), 431–473. https://doi.org/10.1145/233551.233553

[21] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[22] Ilia Iliashenko and Vincent Zucca. 2021. Faster homomorphic comparison operations for BGV and BFV. *Proc. Priv. Enhancing Technol.* 2021, 3 (2021), 246–264. https://doi.org/10.2478/popets-2021-0046

[23] Malika Izabachène, Renaud Sirdey, and Martin Zuber. 2019. Practical Fully Homomorphic Encryption for Fully Masked Neural Networks. In *Cryptology and Network Security - 18th International Conference, CANS 2019, Fuzhou, China, October 25-27, 2019, Proceedings (Lecture Notes in Computer Science)*, Yi Mu, Robert H. Deng, and Xinyi Huang (Eds.), Vol. 11829. Springer, 24–36. https://doi.org/10.1007/978-3-030-31578-8_2

[24] Stanisław Jarecki and Vitaly Shmatikov. 2007. Efficient two-party secure computation on committed inputs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 97–114.

[25] Wenjie Lu, Jun-Jie Zhou, and Jun Sakuma. 2018. Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM, 67–74. https://doi.org/10.1145/3196494.3196503

[26] Wen-jie Lu, Jun-Jie Zhou, and Jun Sakuma. 2018. Non-interactive and output expressive private comparison from homomorphic encryption. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 67–74.

[27] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 155–174. https://doi.org/10.1515/popets-2016-0010

[28] Goldreich Oded. 2009. *Foundations of Cryptography: Volume 2, Basic Applications* (1st ed.). Cambridge University Press, USA.

[29] Femi G. Olumofin and Ian Goldberg. 2011. Revisiting the Computational Practicality of Private Information Retrieval. In *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers (Lecture Notes in Computer Science)*, George Danezis (Ed.), Vol. 7035. Springer, 158–172. https://doi.org/10.1007/978-3-642-27576-0_13

[30] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. 2018. SoK: Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*. 399–414. https://doi.org/10.1109/EuroSP.2018.00035

[31] Michael O. Rabin. 2005. How To Exchange Secrets with Oblivious Transfer. *IACR Cryptol. ePrint Arch.* (2005), 187. http://eprint.iacr.org/2005/187

[32] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. https://doi.org/10.1145/359168.359176

[33] Sean W. Smith and David Safford. 2001. Practical server privacy with secure coprocessors. *IBM Syst. J.* 40, 3 (2001), 683–695. https://doi.org/10.1147/sj.403.0683

[34] Anselme Tueno, Yordan Boev, and Florian Kerschbaum. 2020. Non-interactive Private Decision Tree Evaluation. In *Data and Applications Security and Privacy XXXIV - 34th Annual IFIP WG 11.3 Conference, DBSec 2020, Regensburg, Germany, June 25-26, 2020, Proceedings (Lecture Notes in Computer Science)*, Anoop Singhal and Jaideep Vaidya (Eds.), Vol. 12122. Springer, 174–194. https://doi.org/10.1007/978-3-030-49669-2_10

[35] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. 2019. Private Evaluation of Decision Trees using Sublinear Cost. *Proc. Priv. Enhancing Technol.* 2019, 1 (2019), 266–286. https://doi.org/10.2478/popets-2019-0015

[36] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. 2016. Privately Evaluating Decision Trees and Random Forests. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 335–355. https://doi.org/10.1515/popets-2016-0043

[37] Martin Zuber and Renaud Sirdey. 2021. Efficient homomorphic evaluation of k-NN classifiers. *Proc. Priv. Enhancing Technol.* 2021, 2 (2021), 111–129.