RESEARCH-ARTICLE

Level Up: Private Non-Interactive Decision Tree Evaluation using Levelled Homomorphic Encryption

RASOUL AKHAVAN MAHDAVI, University of Waterloo, Waterloo, ON, Canada

HAOYAN NI, University of Waterloo, Waterloo, ON, Canada

DIMITRY LINKOV, University of Waterloo, Waterloo, ON, Canada

FLORIAN KERSCHBAUM, University of Waterloo, Waterloo, ON, Canada

Open Access Support provided by:

University of Waterloo

Published: 15 November 2023

CCS '23: ACM SIGSAC Conference on Computer and Communications Security
November 26 - 30, 2023
Copenhagen, Denmark

Conference Sponsors:
SIGSAC

https://doi.org/10.1145/3576915.3623095
ISBN: 9798400700507

Level Up: Private Non-Interactive Decision Tree Evaluation using Levelled Homomorphic Encryption

Rasoul Akhavan Mahdavi
rasoul.akhavan.mahdavi@uwaterloo.ca
University of Waterloo
Waterloo, Canada

Haoyan Ni
h27ni@uwaterloo.ca
University of Waterloo
Waterloo, Canada

Dimitry Linkov
dimitry.linkov@uwaterloo.ca
University of Waterloo
Waterloo, Canada

Florian Kerschbaum
florian.kerschbaum@uwaterloo.ca
University of Waterloo
Waterloo, Canada

ABSTRACT

As machine learning as a service continues gaining popularity, concerns about privacy and intellectual property arise. Users often hesitate to disclose their private information to obtain a service, while service providers aim to protect their proprietary models. Decision trees, a widely used machine learning model, are favoured for their simplicity, interpretability, and ease of training. In this context, Private Decision Tree Evaluation (PDTE) enables a server holding a private decision tree to provide predictions based on a client's private attributes. The protocol is such that the server learns nothing about the client's private attributes. Similarly, the client learns nothing about the server's model besides the prediction and some hyperparameters.

In this paper, we propose two novel non-interactive PDTE protocols, XXCMP-PDTE and RCC-PDTE, based on two new non-interactive comparison protocols, XXCMP and RCC. Our evaluation of these comparison operators demonstrates that our proposed constructions can efficiently evaluate high-precision numbers. Specifically, RCC can compare 32-bit numbers in under 10 milliseconds.

We assess our proposed PDTE protocols on decision trees trained over UCI datasets and compare our results with existing work in the field. Moreover, we evaluate synthetic decision trees to showcase scalability, revealing that RCC-PDTE can evaluate a decision tree with over 1000 nodes and 16 bits of precision in under 2 seconds. In contrast, the current state-of-the-art requires over 10 seconds to evaluate such a tree with only 11 bits of precision.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Cryptography**.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623095

KEYWORDS

Homomorphic Encryption, Decision Tree, Private Decision Tree Evaluation

ACM Reference Format:
Rasoul Akhavan Mahdavi, Haoyan Ni, Dimitry Linkov, and Florian Kerschbaum. 2023. Level Up: Private Non-Interactive Decision Tree Evaluation using Levelled Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3576915.3623095

1 INTRODUCTION

With the widespread adoption of machine learning (ML) in many industries, there is a growing interest to offer cloud-based machine learning services [1–4]. However, using cloud-based ML services necessitates clients to share their confidential data with providers to benefit from these services. For many users, this prerequisite raises serious concerns about the potential loss of privacy. Additionally, companies which wish to collaborate and use each other's services cannot risk exposing their customers' and employees' data. This creates a barrier to potential business collaborations. Moreover, service providers are unwilling to relinquish classification models to users, which could eliminate their competitive advantage and put the users in the training data at risk.

Decision trees are a well-known ML algorithm which are still used widely in many tasks due to their simplicity, interpretability, and ease of training [20, 33]. Private Decision Tree Evaluation (PDTE) is a protocol for providing a prediction using a private decision tree hosted by a server on a private input provided by a client. At the end of the protocol, the server learns nothing about the client's input (input privacy), and the client learns nothing about the server's decision tree (model privacy) other than the result of the inference and some hyperparameters.

One set of solutions is interactive, where the client and server exchange messages [9, 10, 37, 40] in multiple rounds. These solutions are based on tools such as multi-party computation, secret-sharing and garbled circuits [25]. Another category of solutions is non-interactive approaches, where the client can submit the query and go offline until the response is ready. This is great for the setting where the client lacks computational power or suffers from limited bandwidth. All existing solutions to non-interactive PDTE use levelled or fully homomorphic encryption [6, 15, 29, 39]. Solutions

requiring levelled homomorphic encryption select parameters that depend on the comparison's precision. This limits the scalability of the solution significantly [29, 40]. Solutions using fully homomorphic encryption do not suffer the same issue, but individual operations are inefficient due to the expensive bootstrapping procedure. The efficiency problem is exacerbated because some fully homomorphic schemes do not support SIMD operations, so inferring multiple samples in parallel is not possible.

This work proposes two non-interactive PDTE protocols using levelled homomorphic encryption schemes. We propose a protocol where the multiplicative depth of the entire PDTE protocol does not depend on the tree's structure or the number of client attributes, making it more scalable and practical. Our solution can scale to trees with over 1000 decision nodes and 100 client attributes. At the heart of our proposed protocols are efficient non-interactive comparison operators. First, an extension of XCMP by Lu et al. [29], which we denote as Extended XCMP (XXCMP). This extension supports arbitrary precision and is implemented using automorphisms in the SEAL library. Second, is a comparison operator based on the concept of range covers [24, 35], combined with constant-weight equality operators proposed by Mahdavi and Kerschbaum [30]. We denote this comparison operator as Range Cover Comparison (RCC). Both operators are implemented with FV, an RLWE-based cryptosystem, but in different modes of operation [18]. Comparison operators are the core building block in all non-interactive PDTE protocols [6, 8, 15, 29, 40]. Our proposed comparison operators can efficiently compare numbers with arbitrary precision. In contrast, previous work is limited in precision due to efficiency. XCMP is limited to 13 bits, while the operator used by Cong et al. [15] can only compare 11-bit numbers. The comparison operator of Tueno et al. [39] can compare numbers with arbitrary precision. However, the parameters of the levelled FHE scheme grow with bit precision, limiting the solution's efficiency for high-precision inputs. Our evaluation shows that XXCMP and RCC are up to 100 times faster than the operators proposed by Tueno et al. Since our comparison operators perform comparisons with arbitrary precision, models such as decision trees need not be retrained with low precision to enable private inference.

We use the SumPath algorithm described in Section 2.4 to evaluate the paths in the decision tree. SumPath requires no multiplications and hence does not increase the multiplicative depth of the circuit. By combining SumPath with XXCMP and RCC, we propose two new PDTE protocols, XXCMP-PDTE and RCC-PDTE, which use RLWE-based cryptosystems in two different modes of operation. While using RLWE-based cryptosystems with SIMD support has previously been proposed [39], it was only to infer multiple samples in parallel and reduce amortized time. In contrast, our work uses SIMD operations to not only perform multiple inferences but also to speed up even a single inference, which reduces client latency. This approach also allows the client to pack more information into fewer ciphertexts, reducing the overall communication between the client and server when only one inference is performed.

In our evaluation, we train decision trees with different bit precision over commonly used UCI datasets [16]. Our evaluation shows that XXCMP-PDTE and RCC-PDTE are up to 5 times faster than SortingHats [15], which is a state-of-the-art solution for PDTE. This advantage increases when inferring many samples in parallel. The performance of PDTE, i.e., the communication and computation cost, depends mainly on three factors: precision, the number of decision nodes, and the number of client attributes. We perform an ablation study over these parameters using synthetic decision trees to demonstrate the dependency between these parameters and the performance. Our experiments show that the number of client attributes affects communication and the number of decision nodes affects computation, while bit precision influences both metrics. Moreover, our experiments show that RCC-PDTE is the most scalable solution. It outperforms all other solutions when the number of decision nodes grows, and the number of client attributes increases. Specifically, it can infer decision trees with over 1000 decision nodes and 16-bit precision in under 2 seconds. SortingHats requires more than 10 seconds to evaluate such a decision tree with only 11 bits of precision.

In summary, our contributions are as follows:

- Two non-interactive comparison protocols which we denote as XXCMP and RCC, that can compare numbers with arbitrary precision using levelled homomorphic encryption.
- Two non-interactive PDTE protocols XXCMP-PDTE and RCC-PDTE.
- Evaluation of proposed comparison operators with state-of-the-art protocols.
- Ablation of PDTE over the number of client attributes and the size of the decision tree, which shows RCC-PDTE to be the most scalable solution. Our experiments show that RCC-PDTE can evaluate decision trees with up to 1000 nodes in less than 2 seconds.

In Section 2, we review necessary background material such as homomorphic encryption, decision trees and private decision tree evaluation, range covers and tree traversal algorithms. We also compare the properties of related work on non-interactive comparison and non-interactive PDTE in this section. We describe our constructions, XXCMP and RCC, two non-interactive private comparison protocols in Section 6.1. In Section 5, we describe our PDTE protocols, XXCMP-PDTE and RCC-PDTE. In Section 6.2, we compare XXCMP and RCC with other non-interactive comparison protocols and then compare PDTE protocols with our constructions. We conclude in Section 7 with a discussion on limitations and future work.

2 BACKGROUND & RELATED WORK

Table 2 summarizes the notation used in the background section and throughout the paper. Throughout the paper, we use $[n]$ to refer to the set $\{0, 1, \cdots, n-1\}$, for $n \in \mathbb{N}$.

2.1 Homomorphic Encryption

Homomorphic Encryption (HE) is a form of encryption that permits computation on the data while in encrypted form. Levelled homomorphic encryption (LHE) schemes permit computation of circuits with a limited multiplicative depth [11, 18]. The parameters of the cryptosystem are chosen based on the multiplicative depth. Hence, we try to design algorithms with a lower multiplicative depth to enhance performance. A fully homomorphic encryption (FHE) scheme permits an unlimited amount of operations with the

Encoding	Plaintext Space	Operation		
Polynomial	$R_p = \frac{\mathbb{Z}_p[X]}{X^N+1}$	Polynomial Add	$c_1(x), c_2(x) \in C$	$m_1(x) + m_2(x)$
		Plain Polynomial Mult.	$c_1(x) \in C, m_2(x) \in R_p$	$m_1(x)m_2(x)$
		Polynomial Mult.	$c_1(x), c_2(x) \in C$	$m_1(x)m_2(x)$
		Oblivion Expansion [7, 13]	$c_1(x) \in C, k \in \mathbb{N} \cup \{0\}$	Coefficient of x^k in $m_1(x)$
Batched	$\mathbb{Z}_p^{N/2}$	SIMD Add	$c_1(x), c_2(x) \in C$	$m_1(x) \oplus m_2(x)$
		Plain SIMD Mult.	$c_1(x) \in C, m_2(x) \in \mathbb{Z}_p^{N/2}$	$m_1(x) \otimes m_2(x)$
		SIMD Mult.	$c_1(x), c_2(x) \in C$	$m_1(x) \otimes m_2(x)$
		Circular (Right) Rotation	$c_1(x) \in C, k \in \mathbb{N}$	$\text{Rotate}_k(m_1(x))$

Table 1: Different encodings for the FV cryptosystem. In all operations, we have $c_1(x), c_2(x) \in C$ that encrypt $m_1(x), m_2(x)$, respectively. Operations over plaintext polynomials happen in R_p. $\oplus$ and $\otimes$ denote element-wise addition and multiplication modulo p between two vectors. Rotate_k denotes the circular right rotation of a vector by k slots.

help of bootstrapping [14, 17]. However, FHE is more computationally expensive and requires large cryptographic keys for setup.

Fan–Vercauteren (FV) Cryptosystem. The Fan–Vercauteren (FV) cryptosystem [18] is a lattice-based homomorphic cryptosystem. An FV ciphertext is an array of polynomials, each from $R_q = \mathbb{Z}_q[X]/(X^N + 1)$, where q is called the *coefficient modulus*. In the simplest case, the ciphertext is only two polynomials. Let C denote the ciphertext space. N and q determine both the security parameter and how many homomorphic operations can be performed on ciphertexts before decryption is necessary. Inputs in this cryptosystem can be encoded in two formats. Table 1 shows the two encoding types and the corresponding operations that can be performed.

Symbol	Description
λ	Security parameter
N	Polynomial modulus degree
p	Plaintext modulus
$\llbracket x \rrbracket$	Encryption of x
$\mathcal{M}$	$(\mathcal{T}, \mathbf{a}, \mathbf{t}, \mathbf{v})$
$\mathcal{T}$	Decision tree nodes
$\mathcal{D}$	Set of internal decision nodes in $\mathcal{T}$
$\mathcal{L}$	Set of leaf nodes in $\mathcal{T}$
m	Number of decision nodes ($\lvert\mathcal{D}\rvert$)
$\mathbf{x}$	Client attribute vector
$\mathbf{a}$	Node to attribute mapping
$\mathbf{t}$	Threshold value function
$\mathbf{v}$	Leaf value function
d	Depth of the decision tree
k	Number of classification labels
h	Hamming weight
ℓ	Constant-weight code length
n	Bit Precision
$[n]$	$\{0, 1, \cdots, n-1\}$
$\mathbb{B}$	$\{0, 1\}$

Table 2: Summary of notation

Microsoft SEAL [34] implements the FV cryptosystem and supports all the operations mentioned in Table 1. We use the SEAL library for our implementations in this work.

2.2 Non-interactive Private Comparison

A comparison operator is a function $f : D \mapsto \{0, 1\}$ such that for $x, y \in D$,

$$f(x, y) = \mathbb{I}[x \leq y] \tag{1}$$

A private comparison is a protocol where two inputs, x and y, are provided, such that one or both of them are encrypted. The output of the protocol is $f(x, y)$ in encrypted form.

In this work, we are particularly interested in non-interactive solutions to this problem. This is useful in protocols where the encrypted input is provided by a lightweight client which may go offline after providing the input. Below we describe three non-interactive private comparison protocols from the literature.

Folklore Private Comparison. The folklore comparison algorithm compares two n-bit numbers in binary format. This is identical to how binary numbers are compared in the clear, with some adaptations to make it easier to compute using HE. The multiplicative depth of the algorithm is $1 + \log_2(n + 1)$, which poses a burden to compute efficiently using HE. Algorithm 1 shows this algorithm. The inputs are binary vectors and all operations are element-wise. We also use the RightShift_k function, which logically shifts the contents of any vector or bitstring by k positions. Additions and multiplications can also be replaced with XOR and AND operations.

Algorithm 1 Folklore Comparison($\mathbb{I}[a \leq b]$)

Input: $a, b \in \{0, 1\}^n$

1: $\theta_{eq} \leftarrow 1 - (a - b)^2$
2: $\theta_{gt} \leftarrow (1 - a) \cdot b$
3: $\theta_{\text{PrefixEq}} = \theta_{gt} \cdot \prod_{k=0}^{n-1} \texttt{RightShift}_k(\theta_{eq})$
4: $\theta \leftarrow \sum_i \theta_{\text{PrefixEq}}[i]$

Output: $\theta \in \{0, 1\}$

Variations of folklore comparison have been implemented in many works using levelled homomorphic encryption [19, 39].

XCMP. Lu et al. [29] proposed a comparison operator called XCMP which compares two encrypted numbers using levelled HE. Cong et al. introduced a variation using TFHE where one operand is unencrypted [15]. This relaxation reduces the runtime of the comparison. Algorithm 2 shows this variant of XCMP which compares an encrypted input, a, provided by the client with an unencrypted b provided by the server. The output is $\mathbb{I}[a \leq b]$. Inputs are encoded as RLWE ciphertexts, i.e., Polynomial encoding from Table 1.

Algorithm 2 XCMP($\mathbb{I}[a \leq b]$) [29]

procedure XCMP(A, b) ▷ $A = X^a, a, b \in [N]$
  $T \leftarrow \frac{1}{2} X^{-b} \cdot (1 + X + \cdots + X^{N-1})$
  $R \xleftarrow{\$} R_p$ and $R[0] = 1/2 \mod p$
  $C = A \cdot T + R$
  return C

The result of the comparison is in the constant term of C.

Iliashenko and Zucca [21]. The comparison function of $x, y \in \mathbb{Z}_p$ can be represented as either a bivariate polynomial of the two inputs or a univariate function of the difference. Iliashenko and Zucca showed how to exploit the structure of these polynomials to efficiently evaluate the comparison function. Their main observation was that these polynomials have many zero coefficients which can be ignored.

Based on this observation, they showed that comparison using FHE schemes that operate over arithmetic circuits can be efficient.

2.3 Decision Trees and PDTE

Decision Trees. A decision tree is a classification algorithm which classifies input data by sequentially checking a series of criteria. The simplest form of a decision tree is represented by a binary tree where each internal node compares an attribute with a threshold. Each leaf is assigned a classification value (or simply a class). To classify a data point, we start at the root of the tree, perform a comparison and move to the right or left child, depending on the result of the comparison. We continue until we reach a leaf and output the class of that leaf.

More formally, a decision tree consists of a set of nodes $\mathcal{T} = \mathcal{D} \cup \mathcal{L}$, where $\mathcal{D}$ and $\mathcal{L}$ are the set of decision nodes and leaf nodes, respectively. There also exists an attribute vector, $\mathbf{x}$, and three functions:

- $\mathbf{a} : \mathcal{D} \mapsto [|\mathbf{x}|]$ maps decision nodes to attribute indices.
- $\mathbf{t} : \mathcal{D} \mapsto \mathbb{Z}$ maps internal nodes to threshold values.
- $\mathbf{v} : \mathcal{L} \mapsto \mathbb{Z}$ maps leaf nodes to classification values.

We denote the tuple $\mathcal{M} = (\mathcal{T}, \mathbf{a}, \mathbf{t}, \mathbf{v})$ as the decision tree model.

Private Decision Tree Evaluation (PDTE). Private Decision Tree Evaluation (PDTE) is a protocol between a server and a client where the server holds the model, $\mathcal{M}$, and the client holds the attribute vector, $\mathbf{x}$. The goal is to infer the tree on the client's attribute vector such that the server does not learn anything about the client's input. Moreover, the client should not learn anything about the server's private decision tree other than the classification result and some hyperparameters.

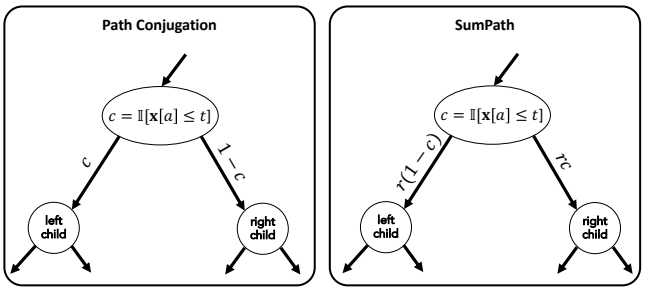


Figure 1: Labelling of edges in two tree traversal methods, Path Conjugation and SumPath. x is the client attribute vector. a, and t are the attribute index and threshold for the parent node, respectively. In SumPath, r is either a random number or 1.

The client may try to steal the model with black-box access to an API through carefully crafted queries to the server [22, 23, 32, 36, 38]. Such an attack is outside the scope of this work, but defences against such attacks is an active area of research [22, 38].

2.4 Tree Traversal

One of the main steps in all private decision tree evaluation protocols is traversing the tree from the root to the leaves, as discussed by Kiss et al. [25]. We denote the leaf holding the result of the prediction as the *result leaf* and the value of that leaf as the *result value*. If all comparisons in the decision nodes are computed as a binary output, there are two methods to aggregate the results and compute the result leaf and value. We denote these two methods as *Path Conjugation* and *SumPath*. For each method, the edges in a tree are annotated as shown in Figure 1.

In Path Conjugation, for each leaf in the tree, all values on the edges between the root and that leaf are conjugated. Consequently, the result leaf will have a value of one, and all other leaves will have a value of zero. This process can be computationally optimized by reusing computations in inner nodes [39]. Some works, such as those by Sortinghats and Tueno, utilize this approach with fully homomorphic encryption and the CMux gate in TFHE [15, 40]. However, this approach is not practical when using levelled homomorphic encryption, given that the multiplicative depth of the circuit will depend on the depth of the tree.

On the other hand, the SumPath method has been proposed previously for use with additive encryption [25, 37] or levelled homomorphic encryption in an arithmetic field [40]. Edges are annotated as indicated in Figure 1. For each leaf in the tree, the sum of all the edges in the tree from the root to that leaf is assigned to that leaf. Only the result leaf will have a sum of zero, and all others will be non-zero. By returning the sum on each leaf, plus the masked value on each leaf, the client can infer the correct result value. The advantage of this approach is that no multiplications or conjugations are required, which makes it computationally inexpensive and does not increase the multiplicative depth.

2.5 Range Covers & Point Encoding

Range covers are a method to represent intervals in a manner that is concise and easy to use in secure computation [24, 35]. Let T be a binary tree of internal nodes of prefixes numbers in $[2^n]$ and leaf nodes of the elements in $[2^n]$. The children of a prefix p are $p|0$ and $p|1$, and the root is the empty prefix. A *range cover* $RC_n(a, b)$ is a set of nodes in T, such that its set of children at the leaf level are all elements in the range $[a, b]$, where $a, b \in [2^n]$. The *best range cover* contains at most $2 \log n$ nodes with at most 2 nodes at each level of T (excluding the root level). A uniform range cover is a modification of the best range cover such that each level of the tree contains exactly 2 nodes [24, 35]. This is achieved by padding the best range cover with dummy nodes at all levels if fewer than two nodes are chosen. The advantage of a uniform range cover is that the size is independent of the interval, which is a requirement in private protocols. For every range $[a, b]$ there exists a best range cover and uniform range cover.

A *point encoding* $PE_n(c)$ of an element $c \in [2^n]$ is the set of nodes from the leaf c to the root (except the root itself). A point encoding consists of n nodes with one node at each level of T (except the root level). We denote as $RC_n(a, b)[i]$ and $PE_n(c)[i]$ the i-th element of a range cover or encoding, respectively.

We can test the relationship $c \in [a, b]$ given $RC_n(a, b)$ and $PE_n(c)$ by checking to see if there are any common prefix nodes between the range cover and point encoding. We give the full algorithm on how to do this check in Algorithm 12 in the appendix.

Figure 2 shows an example: the prefix tree for $[0, 2^3 - 1]$ with the range cover for $[0, 4]$, indicated by the shaded boxes, and point encodings for 2 and 6. We can see that $RC_3(0, 4)$ and $PE_3(2)$ have the prefix node corresponding to 0 in common whereas $RC_3(0, 4)$ and $PE_3(6)$ have no prefix node in common.

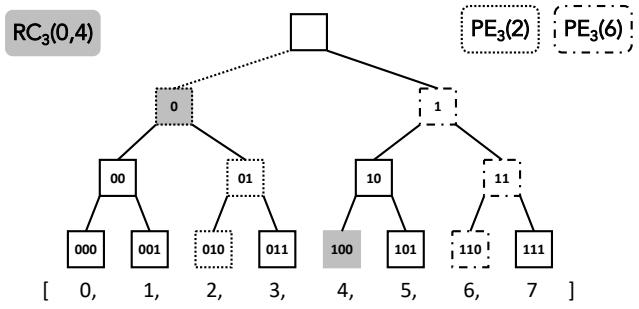


Figure 2: Range cover for $[0, 4]$ (shaded boxes) and point encoding of 2 and 6 in domain $[0, 2^3 - 1]$. In this example, $RC_3(0, 4)$ and $PE_3(2)$ have a prefix node in common, given that $2 \in [0, 4]$ but $RC_3(0, 4)$ and $PE_3(6)$ have no prefix node in common since $6 \notin [0, 4]$.

2.6 Constant-weight Equality Operators

Constant-weight equality operators, proposed by Mahdavi and Kerschbaum, are equality operators with a constant multiplicative depth, independent of the bitlength of the operands [30]. To use the constant-weight equality operator, numbers are represented as constant-weight codes with Hamming weight h. We make an adjustment to the encoding algorithm to encode null values as well. Null values are encoded as the all-zero string. We include the adjusted algorithm from the work of Mahdavi and Kerschbaum in Appendix A. To compare constant-weight codewords, we use the arithmetic constant-weight equality operator in this work, which has a multiplicative depth of $1 + \log_2 h$. We use this operator to be able to compare many pairs of numbers simultaneously, and in a SIMD fashion. This operator is shown in Algorithm 3.

Algorithm 3 Arithmetic Constant-weight Equality Operator [30]

1: **procedure** ARITH-CW-EQ-OP(x, y) ▷ $x, y \in CW(\ell, h) \cup \{0^\ell\}$
2: $h' = \sum_{i \in [\ell]} x[i] \cdot y[i]$
3: $e = 1/h! \cdot \prod_{i \in [h]} (h' - i)$
return e ▷ $e \in \mathbb{B}$

3 RELATED WORK ON PDTE

3.1 Interactive PDTE

Decision trees can be privately evaluated using two-party computation (2PC) or using a combination of 2PC and homomorphic encryption. There are also solutions which use tools such as secret sharing, garbled circuits, and oblivious transfer [9, 12, 25, 28]. Unlike homomorphic encryption, 2PC is usually communication-bound, i.e., reducing the communication complexity or rounds is necessary to increase efficiency. However, since this work focuses on non-interactive decision tree evaluation, we provide only a few selected examples of the work in private decision trees using 2PC.

2PC, e.g., using Yao's garbled circuit, implements a constant-round protocol. Brickell et al. [12] present an improved constant-round protocol using 2PC and homomorphic encryption. Kiss et al. [25] later surveyed this and a number of other protocols and identified several design options and new combinations of these protocols systematizing the protocols.

However, 2PC usually leads to communication complexity which is exponential in the tree depth since all branches need to be evaluated. Using different techniques, one can also reduce the communication complexity. Tueno et al. [40] propose to use oblivious RAM (ORAM). Bai et al. [9] propose to use additive homomorphic encryption or pseudo-random functions.

Given enough network capacity, it is possible to train decision tree classifiers using 2PC or in general multi-party computation (MPC). Lindell and Pinkas [27] present a theoretic design of a 2PC protocol for computing the logarithm in the ID3 training algorithm. Since then, many proposals for practical systems have been made, e.g., by Wu et al. [41], by Zheng et al. [42] and by Lu et al. [28].

3.2 Non-Interactive PDTE

Interactive protocols are great for computational efficiency but usually require high network activity. Moreover, they require several rounds of interaction between the client and server. This is not a good solution if the client is not strong or has a limited network connection. The client may wish to go offline while waiting for the result in many use cases. Non-interactive approaches to private decision tree evaluation primarily use homomorphic encryption. Some

	PROBONITE [8]	PDT-Bin [39]	PDT-Int [39]	SortingHats [15]	XXCMP-PDTE	RCC-PDTE
Supports Unbalanced	×	✓	✓	✓	✓	✓
Attribute Selection	PIR	Clear	Clear	Clear	Clear	Clear
Comparison	PBS (CT-CT)	Folklore	Lin-Tzeng [26]	XCMP-CT-PT	**XXCMP**	**RCC**
Path Evaluation	CMux	AND	SumPath	CMux	SumPath	SumPath
Batchable	×	✓	✓	×	✓	✓
Bit Precision	$< 8^*$	n	n	11	n	n
Levelled or FHE	FHE	LHE or FHE	LHE(BGV)	FHE(TFHE)	LHE(FV)	LHE(FV)
# of Comparisons	$O(d)$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}\rvert$
Query Complexity	$O(\lvert\mathbf{x}\rvert)$	$O(n\lvert\mathbf{x}\rvert)$	$O(n\lvert\mathbf{x}\rvert)$	$O(N\lvert\mathbf{x}\rvert)$	$O(N\lvert\mathbf{x}\rvert)$	$O(\ell n\lvert\mathbf{x}\rvert)$
Mult. Depth	N/A	$\log_2 n$	$\log_2 n$	N/A	$\lceil\log_2(n/\log_2 N)\rceil$	$1+\log_2 h$

Table 3: Properties of Non-interactive Private Decision Tree Evaluation Protocols. a is the number of client attributes, d is the depth of the tree, $\mathcal{D}$ is the set of internal decision nodes, and $\mathbf{x}$ is the client attribute vector. * The precision of PROBONITE depends on the choice of parameters for the LWE scheme but is typically less than 8 bits.

works used levelled homomorphic encryption [6, 29, 40] while others require a fully homomorphic scheme [8, 15, 40]. A decision tree is comprised of many comparisons between client attributes and thresholds, so at the heart of all protocols is an efficient comparison operation. The primary approach in many works is to reduce the precision of the comparison to make it efficient [15, 29]. For this to be possible, the decision tree evaluation has to be fine-tuned for low precision. Quantization-aware training is one approach to this. If a model already exists with high precision, it can not be used as it is and has to be retrained. Table 3 summarizes the properties of related work. We also provide a short description of how each protocol works.

XCMP PDTE [29]. Lu et al. proposed a non-interactive PDTE protocol based on XCMP, described in Section 2.2. All comparisons are performed using XCMP, and all paths are evaluated using the same method as the work of Tai et al. [37]. This protocol's main limitation is that the precision of the comparison which they use is limited to 13 bits, i.e., $\log_2 N$. The protocol's design was such that the model holder could also encrypt the thresholds of the model and delegate the evaluation to a third-party server. In this case, the third party would learn the tree's structure but not the thresholds.

PDT-Bin & PDT-Int [39]. Tueno, Boev, and Kerschbaum proposed two PDTE protocols using a binary and arithmetic circuit. In their first construction, denoted as PDT-Bin, In this construction, numbers are represented using binary encoding and compared using a Folklore comparison. Traversal of the decision tree in PDT-Bin is performed using homomorphic AND operations, ultimately yielding a single result that represents the outcome of the classification.

In addition to PDT-Bin, Tueno, Boev, and Kerschbaum introduced another protocol, PDT-Int, which is based on levelled homomorphic encryption, specifically employing the BGV scheme. For the comparison operator in PDT-Int, they adopted a variation of the Lin-Tzeng [26] protocol, which outputs zero in the case of a match and a random number otherwise. To traverse paths in the decision tree, the SumPath technique is employed, and one value corresponding to each leaf node is returned.

PROBONITE [8]. Azogagh et al. proposed a PDTE protocol named PROBONITE which evaluates only one path of the tree [8]. In contrast to other non-interactive protocols, which perform one comparison for each node in the decision tree and evaluate all paths, PROBONITE only evaluates one path. The protocol starts at the root and traverses one path down the tree. This is done by using two subprocedures: 1) *Blind Array Access*, which is used to select the next attribute with which to compare 2) *Blind Node Selection*, which selects the next node to traverse to based on the result of the comparison. The entire protocol is performed using HE, and the comparison is also performed using the functional bootstrapping capability of TFHE. Given that only one comparison is performed at each level, this greatly improves performance by not performing unnecessary comparisons. The drawback is that since the server does not know which threshold to compare with, it must blindly find the correct threshold with the Blind Array Access subprocedure, which can be computationally expensive. Moreover, the comparison happens between two ciphertexts, as opposed to other approaches which compare encrypted attributes with cleartext thresholds [15, 40]. For privacy, imbalanced trees are padded with redundant nodes to a full, balanced tree so that all queries are equally expensive. The protocol only leaks the depth of the tree and not the number of leaves or any other properties of the tree. There is no public implementation of this work available, so we only resort to the theoretical comparison provided in the Table 3.

SortingHats [15]. Cong et al. expanded on the idea of XCMP-style comparisons [29]. XCMP focused on comparing two encrypted numbers, but in PDTE, one operand is usually in the clear, making the comparison operation simpler. Cong et al. proposed a faster comparison operation based on XCMP where only one operand is encrypted and the other is in the clear [15]. We describe this protocol in Algorithm 2. The operation is done with only two polynomial multiplications, which is much cheaper than comparing two encrypted numbers which requires 16 polynomial multiplications [29]. Using their proposed operand, they designed a non-interactive private decision tree evaluation protocol called SortingHats. The authors used TFHE-based FHE to implement their protocol. They also

proposed using transciphering to reduce communication between the client and server, a method orthogonal to this work to reduce communication costs. The main limitation of SortingHats is the bit precision, which is currently capped at 11 bits.

4 OUR COMPARISON OPERATORS

In this section, we propose two operators for non-interactive private comparison. First is an extension of the XCMP protocol mentioned in Section 2 for larger bit precision. Second, is a novel protocol based on range covers in combination with constant-weight codes.

4.1 XXCMP: Extended XCMP Operator

XCMP only supports the comparison of numbers smaller than N. We propose an extension based on XCMP which can support numbers of arbitrary size. We denote this as *XXCMP*. The idea is to represent large numbers in base N. High-order digits are compared first and if they are equal, the next digits are compared. For example, let $a = a_1N + a_0$ and $b = b_1N + b_0$ where $a_i, b_i \in [N]$. Then we use the following identity:

$$\mathbb{I}[a > b] = \mathbb{I}[a_1 > b_1] + \mathbb{I}[a_1 = b_1] \cdot \mathbb{I}[a_0 > b_0]. \tag{2}$$

For this, we need an equality operator over numbers encoded as XCMP ciphertexts. Within that algorithm, we use another subprotocol which checks the equality of two numbers in the XCMP format. This protocol uses the oblivious expansion technique proposed by Angel et al. [7]. Algorithm 4 shows the extension of XCMP for numbers smaller than N^2. This algorithm requires one homomorphic multiplication. This can be extended to numbers with arbitrary length. We have included the algorithm for numbers of arbitrary size in the appendix. In the general case, to compare numbers smaller than $n = N^k$, the multiplicative depth of the circuit is $\lceil \log_2 k \rceil$ and requires $k(k-1)/2$ homomorphic multiplications.

Algorithm 4 Computing $\mathbb{I}[a > b]$ using Extended XCMP (XXCMP) for $a, b \in [N^2]$ such that $a = a_1N + a_0$ and $b = b_1N + b_0$ where $a_i, b_i \in [N]$

procedure XCMP$_0$(X^a, b) ▷ $a, b \in [N]$
  $T \leftarrow -(1 + X + \cdots + X^{N-b-1})$
  $R \xleftarrow{\$} R_p$ and $R[0] = 0 \mod p$
  $C_0 = X^a \cdot T + R$
  return C_0

procedure XXCMP$_2$(A, b) ▷ $A \in R_p^2, b \in [N^2]$
  $X^{a_1}, X^{a_0} \leftarrow A$
  $gt_0 \leftarrow$ XCMP$_0$(X^{a_0}, b_0)
  $gt_1 \leftarrow$ XCMP$_0$(X^{a_1}, b_1)
  $eq_1 \leftarrow$ Oblivious-Expansion(X^{a_1}, b_1)
  $C = gt_1 + eq_1 \cdot gt_0$
  return C

4.2 Range-Cover Comparison (RCC) Operator

Assume we have $a, b \in [2^n]$ and we want to compute $\mathbb{I}[a \le b]$. At a high level, the idea is to use the following statement:

$$a \le b \iff b \in [a, 2^n - 1] \tag{3}$$

which is similar to the RC/PE inclusion problem. However, one end of the interval is always the maximum value. Using this constraint, we define a restricted version of a range cover called the *One-sided Uniform Range Cover (OURC)*. The main difference between a typical range cover and OURC is that an OURC consists of only one prefix node in each level of the prefix tree (including the root level). Hence, the inclusion check requires only $n + 1$ equality checks instead of $2n$. This results in a major performance improvement when running the circuit using HE. Algorithm 5 shows the procedure for computing the OURC. The modified OURC/PE inclusion procedure is shown in Algorithm 5. We include the procedure for calculating the point encoding as well.

Algorithm 5 Calculating OURC and PE

procedure OURC(x, n) ▷ $x \in [2^n]$
  $\theta_{OURC} \leftarrow$ [Null] $*$ $(n + 1)$
  if $x = 0$ **then**
    $\theta_{OURC}[n] = 0$
    return θ_{OURC}
  $s = 2^n - 1$
  $K \leftarrow \{\}$
  while $s \ge x$ **do**
    Find j such that $2^j \le s < 2^{j+1}$
    $\theta_{OURC}[j] = 2^{n-j} - 1 - \sum_{k \in K} 2^{k-j}$
    $s \leftarrow s - 2^j$
    Add j to K
  return θ_{OURC}

procedure PE(y, n) ▷ $y \in [2^n]$
  return $\left[y, \lfloor y/2 \rfloor, \lfloor y/2^2 \rfloor, \cdots, \lfloor y/2^n \rfloor\right]$

procedure PE-RC-Inclusion($\theta_{OURC}, \theta_{PE}$)
  $\theta_{in} = 0$
  for $i \in [n + 1]$ **do**
    $\theta_{in} = \theta_{in} + \mathbb{I}$ [OURC(a, n)$[i]$ == PE(b, n)$[i]$]
  return θ_{in}

As shown in Algorithm 5, $n + 1$ equality checks are required for one RCC comparison. We replace the equality checks in each iteration of the for loop with a constant-weight equality operator of Hamming weight h. All other operations are additions so the total multiplicative depth of the inequality operator only depends on h. Note that the multiplicative depth does not depend on n, the bit precision of the numbers.

OURC Inclusion using Homomorphic Encryption. In our private comparison protocol, the client, which holds a, sends an encryption of OURC(a, n) to the server. The input is encoded and encrypted such that the comparison is performed efficiently. More specifically, if OURC$(a, n) = \{a_0, a_1, \cdots, a_n\}$, then constant-weight encoding of a_i is spread across ℓ ciphertexts. Figure 3 shows a visualization

of how the packing happens. Each blue box contains all the bits of information about OURC(a, n). Using this encoding, each prefix node, a_i occupies exactly n slots in each ciphertext. The number of occupied slots does not depend on the parameters of the constant-weight code, i.e., the Hamming weight. Multiple pairs of numbers from the client and server can be compared in parallel.

Algorithm 6 details the procedure to encode OURC(a, n) across multiple FV ciphertexts in batched mode. We provide the packing method for the point encoding as well, which is used in the comparison protocol. We note that this is not the only method to encode values in FV ciphertexts. We elaborate on other packing methods and why we chose this method in Section 7.

Algorithm 6 OURC and PE Encoding

procedure OURC-Encode(a, h, ℓ, n) ▷ $a \in [2^n]$
  $[a_0, a_1, \cdots, a_n] \leftarrow$ OURC(a, n)
  for $i \in [n+1]$ **do**
    $a'_i = \text{CWEncode}(a_i, h, \ell)$ ▷ $a'_i \in \mathbb{B}^\ell$
  for $i \in [\ell]$ **do**
    $pt_{OURC}[i] = \left[a'_0[i], a'_1[i], ..., a'_n[i]\right]$
  return pt_{OURC}

procedure PE-Encode(b, h, ℓ, n) ▷ $b \in [2^n]$
  $[b_0, b_1, \cdots, b_n] \leftarrow$ PE(b, n)
  for $i \in [n+1]$ **do**
    $b'_i = \text{CWEncode}(b_i, h, \ell)$ ▷ $b'_i \in \mathbb{B}^\ell$
  for $i \in [\ell]$ **do**
    $pt[i] = \left[b'_0[i], b'_1[i], ..., b'_n[i]\right]$
  return pt_{PE}

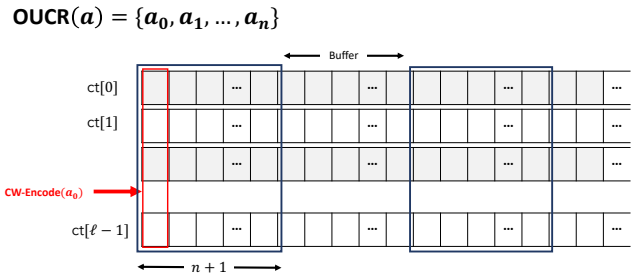


Figure 3: Packing OURC in FV ciphertexts in batched mode.

Given the encryption of OURC(a, n), we can now outline the procedure for comparison. Algorithm 7 shows this procedure. The red symbols show the values that are encrypted when the procedure is performed between a client and server.

5 PDTE USING LEVELED HE

This section describes our two proposed protocols, XXCMP-PDTE and RCC-PDTE. We describe the setup, security model, and details about the two protocols.

Algorithm 7 RCC Comparison

procedure RCC-Compare(a, b)
  $a_{ct} \leftarrow$ OURC-Encode(a, h, ℓ, n) ▷ Done by client
  $b_{pt} \leftarrow$ PE-Encode(b, h, ℓ, n)
  $\theta = \text{Arith-CW-Eq-Op}(a_{ct}, b_{pt})$
  $\theta_{\text{sum}} \leftarrow \sum_{i=0}^{n} \text{Rotate}_i(\theta)$
  $M \leftarrow 0^N, M[0] = 1$ ▷ Mask
  $\theta_{\text{cmp}} \leftarrow \theta_{\text{sum}} \otimes M$
  return θ_{cmp}

5.1 Setup and Security Model

Our PDTE protocols work in the client/server model. The server holds a decision tree model, and the client holds a vector of attributes. The goal is for the client to learn nothing about the server's model other than the inference result. The server should learn nothing from the protocol. Our protocol is non-interactive, i.e., the client uploads its query to the server and waits for a response. The client need not be online while the server processes the query.

Similar to prior work, we work in the semi-honest model, where both the client and server follow the protocol but may try to infer extra information.

Both protocols use homomorphic encryption and assume cryptographic keys such as public keys, evaluation keys, and relinearization keys have been exchanged between the client and server, similar to previous works.

5.2 XXCMP-PDTE: XXCMP + SumPath

In XXCMP-PDTE, we combine the XXCMP comparison with the SumPath algorithm. This protocol is implemented using FV ciphertexts in polynomial mode. The protocol has two main parts 1) For each node, compare the correct client attribute to the corresponding node threshold. 2) Run SumPath and get one encrypted result per leaf. Algorithm 8 depicts this algorithm. d.left and d.right denote the left and right exiting edge from d, respectively. The final result can be compressing the leaves into the same ciphertext. Each leaf can occupy one of the coefficients of the final ciphertext.

Variables that are coloured red are encrypted when running the procedure between a server and client.

Algorithm 8 XXCMP-PDTE : PDTE using XXCMP

procedure XXCMP-PDTE$(\mathbf{x}, \mathcal{M})$
  $(\mathcal{T}, \mathbf{a}, \mathbf{t}, \mathbf{v}) \leftarrow \mathcal{M}$
  for $d \in \mathcal{D}$ **do**
    $c \leftarrow$ XXCMP$(\mathbf{x}[\mathbf{a}[d]], \mathbf{t}[d])$
    d.left $\leftarrow c$
    d.right $\leftarrow 1 - c$
  for $\ell \in \mathcal{L}$ **do**
    $s(\ell) =$ Sum of edges from root to ℓ
    $r_x, r_y \xleftarrow{\$} \mathbb{Z}_p$
    $x(\ell) \leftarrow r_x \cdot s(\ell)$
    $y(\ell) \leftarrow r_y \cdot s(\ell) + v(\ell)$
  return $\{(x(\ell), y(\ell))\}_{\ell \in \mathcal{L}}$

5.3 RCC-PDTE: RCC + SumPath

In RCC-PDTE, comparisons are performed using RCC and the tree is traversed using SumPath. One important difference between this protocol and XXCMP-PDTE is that all comparisons happen in parallel using the batched mode of FV. The results of the comparisons will occupy different slots of the ciphertext. Algorithm 9 outlines this procedure. In the implementation, we also batched the final result into one ciphertext to reduce communication costs.

Algorithm 9 RCC-PDTE : PDTE using RCC

procedure RCC-PDTE($\mathbf{x}$, $\mathcal{M}$)
  $\mathcal{T}, \mathbf{a}, \mathbf{t}, \mathbf{v} \leftarrow \mathcal{M}$
  $\mathbf{t}' \leftarrow$ array of thresholds, aligned with client attributes
  $c \leftarrow$ RCC-Compare($\mathbf{x}, \mathbf{t}'$)
  for $d \in \mathcal{D}$ **do**
    $c' \leftarrow$ Rotate c to get d to first slot
    $d.\text{left} \leftarrow 1 - c'$
    $d.\text{right} \leftarrow c'$
  for $\ell \in \mathcal{L}$ **do**
    $s(\ell)$ = Sum of edges from root to ℓ
    $r_x, r_y \xleftarrow{\$} \mathbb{Z}_p$
    $x(\ell) \leftarrow r_x \cdot s(\ell)$
    $y(\ell) \leftarrow r_y \cdot s(\ell) + v(\ell)$
  return $\{(x(\ell), y(\ell))\}_{\ell \in \mathcal{L}}$

Choosing the Hamming weight. The Hamming weight used for the constant-weight code directly affects the code length. More specifically, the code length is the smallest ℓ such that

$$\binom{\ell}{h} \geq 2^n \tag{4}$$

For $1 \leq h \leq n/2$, the code length decreases as the Hamming weight increases. But for $h > n/2$, the code length increases as the Hamming weight increases. So we do not choose the Hamming weight to be larger than half the bit precision. Given that analysis, the choice of the Hamming weight requires consideration of the trade-off between runtime and communication costs. In our evaluation, we plot several Hamming weights to show the effect.

6 EVALUATION

In this section, we present our evaluation in two parts. In Section 6.1 we benchmark the runtime of our proposed non-interactive private comparison operators in comparison with existing operators. In Section 6.2, we evaluate PDTE algorithms over decision trees trained over UCI datasets. We perform ablation studies to measure the performance of different algorithms with respect to precision, the number of client attributes and the size of the decision tree.

6.1 Benchmarking Private Comparison

In this experiment scenario, we assume a client wants to compare its input values with that of the server using a private comparison operator. We measure the computation time for the operators proposed in Section 4. Specifically, we benchmark 1) RCC 2) Folklore Comparison 3) XXCMP 4) SortingHats Comparison and 5) the work of Iliashenko et al. [21]. The first three are implemented using Microsoft SEAL [34]. In Figure 4 we plot the amortized runtime and communication as a function of the bitlength of the values. RCC is parameterized by the Hamming weight h which has a significant effect on the performance. We plot RCC for multiple Hamming weights to show the effect of the parameter.

All experiments are performed 10 times and the average results are reported. The shaded areas show one standard deviation of error.

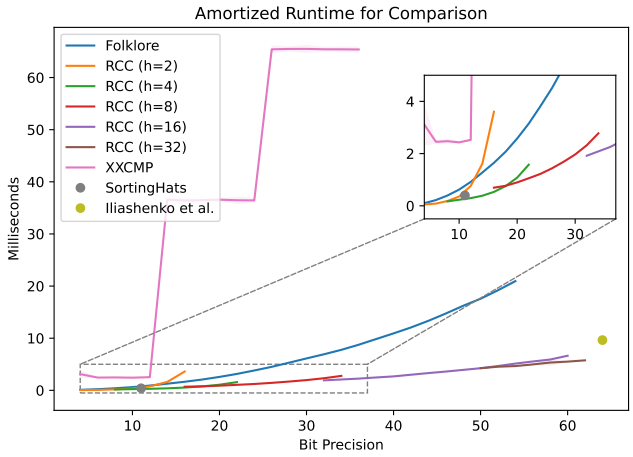


Figure 4: Amortized Time for Comparison. The shaded areas indicate one standard deviation of error. Time for a SortingHats comparison is measured by using the benchmarks in their repository. For RCC, at each point, we use the Hamming weight which has the smallest runtime.

XCMP and SortingHats perform one comparison at a time, whereas Folklore and RCC are batched and perform the comparison in a SIMD fashion. The input is encoded in the same format as that described in Figure 3. In this encoding, the number of parallel comparisons is a function of the precision and the parameters of the encryption scheme. Table 4 shows the number of comparisons for different precisions. The number of parallel comparisons decreases as the precision increases. Due to our encoding method, the number of parallel comparisons does not depend on the Hamming weight, in the case of RCC. The work of Iliashenko et al. also performs comparison in a batched manner. We use the univariate variant of their protocol which, per their experiments, produces the best results.

Bitlength	8	12	16	20	24	28	32	36
# of Comps	963	655	496	399	334	287	252	224

Table 4: Number of parallel comparisons in RCC and Folklore comparison as a function of the bit precision.

Figure 4 shows that RCC has the smallest amortized runtime for a private comparison for any bit precision. For a bit precision of 11, SortingHats and RCC have an approximately similar runtime, but SortingHats can not extend to higher bit precision.

6.2 Benchmarking PDTE

In this subsection, we compare the proposed private decision tree evaluation protocols in terms of server computation time and communication between the client and server. We first benchmark the performance whilst inferring decision trees trained on common UCI datasets and measure the communication and computation overhead. Our experiments over datasets show that the performance of these algorithms is tied to the number of client attributes and the size of the decision tree. Hence, we ablate with respect to these two parameters to understand the effect.

6.2.1 Implementation & Experimental Details. XXCMP-PDTE and RCC-PDTE are implemented as described in Section 5. We parallelized some steps in both protocols to enhance performance. Particularly, in RCC-PDTE, the constant-weight equality operator was shown to be highly parallelizable by Mahdavi and Kerschbaum [30]. We also implement a baseline solution which we denote as *Folklore-PDTE*. Folklore-PDTE is implemented similarly to RCC-PDTE, with only the comparison replaced with the Folklore comparison from Section 2.2. All three algorithms are implemented using Microsoft SEAL version 4.0[1], which implements the FV cryptosystem in polynomial and batching mode [34]. Our implementation is publicly available on Github.[2]

For SortingHats, we use the implementation provided by the authors[3]. We activate the parallelization flag and use artificial input. Experiments are conducted on an Intel(R) Xeon(R) Platinum 8368 CPU @ 2.40GHz server running Ubuntu 22.02 with 32 cores. All experiments are repeated 10 times and the average is reported. The shaded areas indicate the standard deviation of the measurements.

6.2.2 Evaluation over Datasets. We train decision trees over four datasets from the UCI repository [16], Heart, Breast, Spam, and Steel, which are also used in related work [15, 40]. We train decision trees with the desired precision using the Concrete-ML framework [31]. Table 5 shows the properties of the datasets. The structure of the decision tree changes as the precision increases. In general, a decision tree with higher precision has fewer nodes.

Name	ID	# of Classes	# of Attributes
Breast	1510	2	30
Steel	1504	2	33
Heart	1565	5	13
Spam	44	2	57

Table 5: Characteristics of UCI datasets used in our evaluation

[1]https://github.com/microsoft/SEAL
[2]https://github.com/RasoulAM/private-decision-tree-evaluation
[3]https://github.com/KULeuven-COSIC/SortingHat

In Figure 5, communication and computation are plotted as a function of the precision for Folklore-PDTE, XXCMP-PDTE, and RCC-PDTE. SortingHats does not permit arbitrary precision, so we plot it as one point in the graphs.

Figure 5 shows the results for four datasets. Folklore-PDTE is consistently the slowest of all solutions. None of the benchmarked approaches is consistently better, but in all cases, it is either XXCMP-PDTE or RCC-PDTE. In communication, Folklore-PDTE is dominant given that it has the most compact representation for a number, but given its impractical runtime, we can dismiss that. Hence, if we disregard that, SortingHats has the least communication overhead compared to XXCMP-PDTE and RCC-PDTE.

These experiments show that there is not a dominant solution that wins in all cases for all metrics. Communication and computation are a function of many factors, such as the number of client attributes and the number of decision nodes. We perform ablations to better understand the effect of each of these parameters.

6.2.3 Ablation over Number of Attributes. In this experiment, we benchmark PDTE over a synthetic decision tree with a varying number of client attributes. Specifically, we generate a synthetic balanced tree of depth 6 (with 31 decision nodes) with three different bit precision, $n = 8, 16, 26$. Note that the same results hold for trees of other sizes and shapes (balanced or unbalanced). We plot the communication complexity of RCC-PDTE and XXCMP-PDTE as we vary the number of client attributes from 5 to 100.

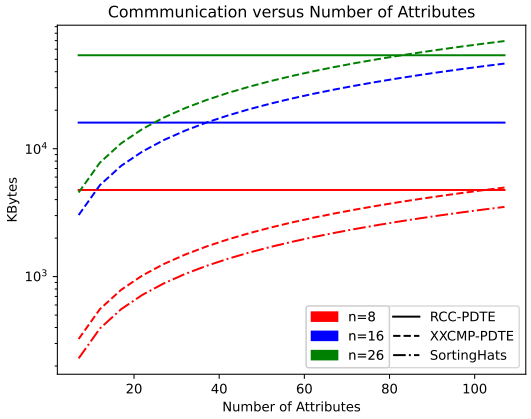
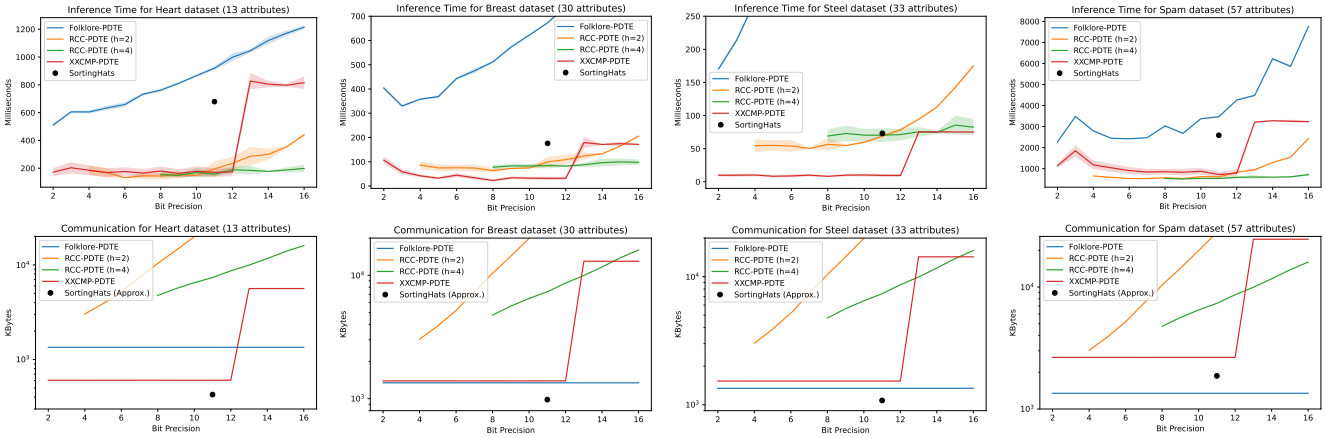


Figure 6: Communication cost of PDTE as a function of the number of attributes.

SortingHats and XXCMP-PDTE encrypt each client attribute in a separate RLWE ciphertext. Hence, the communication complexity scales linearly with the number of client attributes (notice the logarithmic vertical axis). For a given precision, RCC-PDTE has constant communication cost due to the use of batch encoding. When the number of client attributes is small, RCC-PDTE has a higher communication cost compared to other solutions. SortingHats has the smallest communication overhead when precision is less than 11 bits. For precision higher than 11 bits, XXCMP-PDTE is the best solution in terms of communication. While RCC-PDTE has the highest communication cost when the number of attributes



(a) Heart Dataset (b) Breast Dataset (c) Steel Dataset (d) Spam Dataset

Figure 5: Runtime and Communication for Private Decision Tree Evaluation over four datasets. For each dataset, the left graph plots the runtime, and the right graph plots the communication. The shaded area shows one standard deviation of error.

is small, this high overhead shrinks as the number of attributes grows and in some cases, it becomes the dominant solution. For example, at 16-bit precision, RCC-PDTE has the least communication overhead once the number of client attributes exceeds 40.

Note that the runtime of both algorithms does not have a noticeable change as the number of attributes changes. Hence, we only report the approximate runtimes of each approach in Table 6. Similar to the results shown in Figure 5, XXCMP-PDTE is the fastest for a low bit precision, but RCC-PDTE overtakes it for higher bit precision. SortingHats is only applicable for low precision and is slower in that case.

Precision (bits)	Runtime (ms)		
	SortingHats	XXCMP-PDTE	RCC-PDTE
8	648 - 662	133 - 155	149 - 170
16	-	673 - 753	187 - 234
26	-	752 - 845	747 - 966

Table 6: Approximate runtime for private evaluation of balanced decision tree of depth 6 (with 31 decision nodes). The number of attributes varies from 7-100

6.2.4 Ablation over Number of Nodes. In this experiment, we benchmark PDTE over synthetic trees as we vary the number of decision nodes. We fix the number of attributes to 32, but the same results hold for a different number of attributes. We generate balanced trees with depths up to 10, but the results do not depend on the shape of the tree, and the same results hold if the tree is unbalanced. This is because the tree traversal algorithm, SumPath, only takes up at most 10% of the total runtime. Hence, a change in the shape of the tree does not significantly impact the runtime. Figure 7 shows the runtime as a function of the number of nodes for $n = 8, 16, 26$. Note that the horizontal axis is logarithmic.

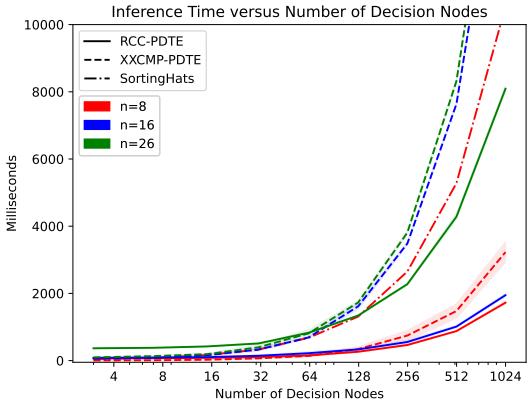


Figure 7: Runtime as a function of the number of nodes.

As expected, the runtime of XXCMP-PDTE increases linearly with the number of decision nodes, given that one comparison is performed for each decision node in the tree. Due to batched computations, the runtime of RCC-PDTE only increases if more ciphertexts are required for the comparisons. For low precision, XXCMP-PDTE has better runtime compared to RCC-PDTE. However, for larger decision trees, RCC-PDTE is faster.

In all the reported protocols, the communication cost is a function of the precision and number of client attributes (and the Hamming weight, in the case of RCC-PDTE). Communication cost does not depend on the size of the decision tree. Hence, we only report the communication cost of each protocol in Table 7.

Precision (bits)	Communication (KB)		
	SortingHats	XXCMP-PDTE	RCC-PDTE
8	2342	1486	4757
16	-	13847	16001
26	-	20770	53795

Table 7: Communication cost for private evaluation of balanced tree over an input with 32 attributes. The depth of the tree to evaluate is up to 7.

6.3 Summary of Results

For non-interactive private comparison, XXCMP and RCC can be utilized for arbitrary precision, while SortingHats has limited precision. If a single comparison with arbitrary precision is required, XXCMP is a good option; however, RCC offers a significantly better amortized time for comparing numerous pairs.

In the context of PDTE, for low precision, either SortingHats, XXCMP-PDTE, or RCC-PDTE may be faster. The optimal solution is contingent on a combination of factors, including bit precision, the number of decision nodes, and the number of client attributes. For low precision, SortingHats is superior in terms of communication, but RCC-PDTE and XXCMP-PDTE are generally faster.

When dealing with bit precision higher than 11, XXCMP-PDTE and RCC-PDTE are the only practical available options, with RCC-PDTE proving to be faster, particularly as the number of decision nodes increases.

7 DISCUSSION AND FUTURE WORK

High-precision Applications. While decision trees may be achievable with smaller precision and quantization-aware training, there are other cases where the precision can not be sacrificed. For example, in the case of intrusion detection, servers may want to check to see if an IP, which is a 32-bit number, is in a specific range or not.

Extension of SortingHats. Other protocols such as SortingHats can also be modified to accommodate larger precision. This can be done using a similar algorithm to XXCMP but with a fully homomorphic scheme such as TFHE. Specifically, the equality check that is performed in line 9 of Algorithm 4 can be performed with functional bootstrapping instead.

Other Packing Methods. Homomorphic encryption in the batched setting offers a lot of flexibility to choose the packing method. Other packing methods can improve communication, particularly when trying to reduce the time for one inference. Tools such as HELayers [5] can help find the other packing strategies.

Comparison with PROBONITE. There is currently no public implementation for PROBONITE available. We conducted a theoretical comparison of PROBONOTE with other PDTE protocols in Table 3, but a practical comparison would also be interesting as part of future work.

8 CONCLUSION

In this work, we propose two protocols for non-interactive private decision tree evaluation leveraging levelled homomorphic encryption, XXCMP-PDTE and RCC-PDTE. These protocols are based on XXCMP and RCC, two non-interactive comparison protocols which can efficiently compare numbers of arbitrary precision with a constant multiplicative depth.

Our experimental analysis demonstrates that several protocols can be used when the client's input features a small number of attributes, the decision tree remains small, and lower precision is acceptable. However, when faced with many client attributes, large decision trees or the necessity for high precision, XXCMP-PDTE and RCC-PDTE emerge as better options compared to SortingHats. In some cases, these two protocols are up to 5 times faster than SortingHats. In very large decision trees, RCC-PDTE is the best solution and can infer a decision tree with over 1000 nodes and 16 bits of precision in under 2 seconds.

REFERENCES

[1] [n. d.]. Amazon Machine Learning. https://aws.amazon.com/machine-learning. Accessed May 2, 2023.
[2] [n. d.]. Azure Machine Learning. https://azure.microsoft.com/products/machine-learning/. Accessed May 2, 2023.
[3] [n. d.]. BigML. https://bigml.com/. Accessed May 2, 2023.
[4] [n. d.]. Google Cloud Vertex AI. https://cloud.google.com/vertex-ai. Accessed May 2, 2023.
[5] Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, Hayim Shaul, and Omri Soceanu. 2023. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Privacy Enhancing Technology Symposium (PETs) 2023* (2023). https://petsymposium.org/2023/paperlist.php
[6] 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
[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy (SP)*. 962–979. https://doi.org/10.1109/SP.2018.00062
[8] Sofiane Azogagh, Victor Delfour, Sébastien Gambs, and Marc-Olivier Killijian. 2022. PROBONITE: PRivate One-Branch-Only Non-Interactive Decision Tree Evaluation. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (Los Angeles, CA, USA) *(WAHC'22)*. Association for Computing Machinery, New York, NY, USA, 23–33. https://doi.org/10.1145/3560827.3563377
[9] Jianli Bai, Xiangfu Song, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2022. Scalable Private Decision Tree Evaluation with Sublinear Communication. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan) *(ASIA CCS '22)*. Association for Computing Machinery, New York, NY, USA, 843–857. https://doi.org/10.1145/3488932.3517413
[10] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2014. Machine Learning Classification over Encrypted Data. *Cryptology ePrint Archive* (2014).
[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 309–325. https://doi.org/10.1145/2090236.2090262
[12] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. 2007. Privacy-Preserving Remote Diagnostics. Association for Computing Machinery, New York, NY, USA, 498–507. https://doi.org/10.1145/1315245.1315307
[13] Hao Chen, Ilaria Chillotti, and Ling Ren. 2019. Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, New York, NY, USA, 345–360. https://doi.org/10.1145/3319535.3354226
[14] 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.

[15] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V.L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 563–577. https://doi.org/10.1145/3548606.3560702

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

[17] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*. Springer, 617–640.

[18] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *Proceedings of the 15th international conference on Practice and Theory in Public Key Cryptography* 2012 (2012), 1–16. https://eprint.iacr.org/2012/144

[19] Yidi Hao, Baodong Qin, and Yitian Sun. 2023. Privacy-Preserving Decision-Tree Evaluation with Low Complexity for Communication. *Sensors* 23, 5 (2023), 2624.

[20] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer.

[21] Ilia Iliashenko and Vincent Zucca. 2021. Faster Homomorphic Comparison Operations for BGV and BFV. *Proceedings on Privacy Enhancing Technologies* 2021, 3 (2021), 246–264. https://doi.org/10.2478/popets-2021-0046

[22] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.

[23] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. 2018. Model Extraction Warning in MLaaS Paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. Association for Computing Machinery, New York, NY, USA, 371–380. https://doi.org/10.1145/3274694.3274740

[24] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. 2013. Delegatable Pseudorandom Functions and Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 669–684. https://doi.org/10.1145/2508859.2516668

[25] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider. 2019. SoK: Modular and Efficient Private Decision Tree Evaluation. *Proceedings on Privacy Enhancing Technologies* 2 (2019), 187–208.

[26] Hsiao-Ying Lin and Wen-Guey Tzeng. 2005. An Efficient Solution to the Millionaires' Problem Based on Homomorphic Encryption. In *Proceedings of the Third International Conference on Applied Cryptography and Network Security* (New York, NY) *(ACNS'05)*. Springer-Verlag, Berlin, Heidelberg, 456–466. https://doi.org/10.1007/11496137_31

[27] Yehuda Lindell and Benny Pinkas. 2000. Privacy preserving data mining. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings*. Springer, 36–54.

[28] Wen-jie Lu, Zhicong Huang, Qizhi Zhang, Yuchen Wang, and Cheng Hong. 2023. Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree. *USENIX Security Symposium* (2023).

[29] 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* (Incheon, Republic of Korea) *(ASIACCS '18)*. Association for Computing Machinery, New York, NY, USA, 67–74. https://doi.org/10.1145/3196494.3196503

[30] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 1723–1740. https://www.usenix.org/conference/usenixsecurity22/presentation/mahdavi

[31] Arthur Meyre, Benoit Chevallier-Mames, Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, and Celia Kherfallah. 2022. Concrete-ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.

[32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[33] J. Ross Quinlan. 1986. Induction of Decision Trees. *Machine learning* 1 (1986), 81–106.

[34] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[35] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. 2007. Multi-Dimensional Range Query over Encrypted Data. In *2007 IEEE Symposium on Security and Privacy (SP '07)*. Oakland, California, USA, 350–364. https://doi.org/10.1109/SP.2007.29

[36] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 3–18. https://doi.org/10.1109/SP.2017.41

[37] Raymond KH Tai, Jack PK Ma, Yongjun Zhao, and Sherman SM Chow. 2017. Privacy-Preserving Decision Trees Evaluation via Linear Functions. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 494–512.

[38] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 601–618.

[39] Anselme Tueno, Yordan Boev, and Florian Kerschbaum. 2019. Non-Interactive Private Decision Tree Evaluation. In *Database Security*. Springer International Publishing, 174–194.

[40] Anselme Tueno, Florian Kerschbaum, Stefan Katzenbeisser, and Privacy Enhancing Technologies Symposium. 2019-01-01. Private Evaluation of Decision Trees using Sublinear Cost. *Proceedings on Privacy Enhancing Technologies* 2019, 1 (2019-01-01).

[41] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proceedings of the VLDB Endowment* 13, 11 (2020).

[42] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. 2021. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Online, 2723–2740. https://www.usenix.org/conference/usenixsecurity21/presentation/zheng

A ARITHMETIC CONSTANT-WEIGHT EQUALITY

Mahdavi and Kerschbaum proposed constant-weight equality operators, which were equality operators used to compare constant-weight codes with a constant multiplicative depth. We use these operators as a building block in our work to achieve a PDTE protocol that has a multiplicative depth independent of precision and the depth of the tree.

Mahdavi and Kerschbaum offer a function for encoding numbers as constant-weight codes. We use this function with the added option of encoding a null value. This is useful in our case since we may need to encode null elements as well. Null elements are encoded as the all-zero string.

Algorithm 10 shows this algorithm. $CW(\ell, h)$ denotes the set of constant-weight codes with length ℓ and Hamming weight h.

Algorithm 10 `CW-Encode` [30]

Input: $x \in [2^n] \cup \{\texttt{Null}\}$, $\ell, h \in \mathbb{N}$ such that $\binom{\ell}{h} \geq 2^n$

```
if x==Null then
    return 0^ℓ
r ← x, h' ← h, y ← 0^ℓ
for ℓ' = ℓ − 1, ..., 1, 0 do
    if r ≥ C(ℓ', h') then
        y[ℓ'] = 1
        r = r − C(ℓ', h')
        h' = h' − 1
    if h = 0 then break
return y
```

Output: $y \in CW(\ell, h) \cup \{0^\ell\}$

The arithmetic constant-weight equality operator is the same as proposed by Mahdavi and Kerschbaum. The input can now be the all-zero string as well. Comparing anything with the null string will yield a non-match.

B XXCMP FOR ARBITRARY-LENGTH NUMBERS

XXCMP can compare numbers of arbitrary size. Algorithm 11 shows the general XXCMP algorithm which compares two number $a, b \in [N^k]$, for some known parameter k. The output of the comparison is in the constant

Algorithm 11 Computing $\mathbb{I}[a > b]$ using Extended XCMP (XXCMP) for $a, b \in [N^k]$ such that $a = (a_{k-1}, \cdots, a_1, a_0)_N$ and $b = (b_{k-1}, \cdots, b_1, b_0)_N$ where $a_i, b_i \in [N]$

procedure XXCMP(A, b) ▷ $A \in R_p^2, b \in [N^2]$
  $(X^{a_{k-1}}, \cdots, X^{a_1}, X^{a_0}) \leftarrow A$
  for $i \in [k]$ **do**
    $gt_i \leftarrow \text{XCMP}_0(X^{a_i}, b_i)$
  for $i \in [k]$ **do**
    $eq_i \leftarrow$ Oblivious-Expansion(X^{a_i}, b_i)
  $C = \sum_{i=0}^{k-1} gt_i \cdot \prod_{j=i+1}^{k-1} eq_j$
  return C

C RC/PE INCLUSION

We can test the relationship between a range cover and point encoding using Algorithm 12. Assume we want to check $c \in [a, b]$ using $RC_n(a, b)$ and $PE_n(c)$.

In this notation, $RC_n(a, b)$ contains $2n$ prefix nodes and we assume the prefix nodes from level i of the prefix tree are in $RC_n(a, b)[2i]$ and $RC_n(a, b)[2i + 1]$ (and some levels may be empty). $PE_n(c)$ contains n prefix nodes and the prefix node from level i of the prefix tree is in $PE_n(c)[i]$. The inclusion algorithm requires $2n$ equality check at most.

Algorithm 12 RC/PE Inclusion

Input: $RC_n(a, b), PE(c, n)$

$\theta_{included} = 0$
for $i \in [n]$ **do**
  $\theta_{included} = \theta_{included} + \mathbb{I}[RC_n(a, b)[2i] == PE_n(c)[i]]$
  $\theta_{included} = \theta_{included} + \mathbb{I}[RC_n(a, b)[2i + 1] == PE_n(c)[i]]$

Output: $\mathbb{I}[c \in [a, b]]$