BPDTE: Batch Private Decision Tree Evaluation via Amortized Efficient Private Comparison

Huiqiang Liang[1], Haining Lu[1,2], and Geng Wang[2]

[1] ViewSouces (Shanghai) Technology Company Limited, China
hqliang_crypto@163.com
[2] Shanghai Jiao Tong University, China
{hnlu,wanggeng}@sjtu.edu.cn

Abstract. Machine learning as a service requires the client to trust the server and provide its own private information to use this service. Usually, clients may worry that their private data is being collected by server without effective supervision, and the server also aims to ensure proper management of the user data to foster the advancement of its services. In this work, we focus on private decision tree evaluation (PDTE) which can alleviates such privacy concerns associated with classification tasks using decision tree. After the evaluation, except for some hyperparameters, the client only receives the classification results from the server, while the server learns nothing.
Firstly, we propose three amortized efficient private comparison algorithms: TECMP, RDCMP, and CDCMP, which are based on the leveled homomorphic encryption. They are non-interactive, high precision (up to 26624-bit), many-to-many, and output expressive, achieving an amortized cost of less than 1 ms under 32-bit, which is an order of magnitude faster than the state-of-the-art. Secondly, we propose three batch PDTE schemes using this private comparison: TECMP-PDTE, RDCMP-PDTE, and CDCMP-PDTE. Due to the batch operations, we utilized a clear rows relation (CRR) algorithm, which obfuscates the position and classification results of the different row data. Finally, in decision tree exceeding 1000 nodes under 16-bit each, the amortized runtime of TECMP-PDTE and RDCMP-PDTE both more than 56× faster than state-of-the-art, while the TECMP-PDTE with CRR still achieves 14× speedup. Even in a single row and a tree of fewer than 100 nodes with 64-bit, the TECMP-PDTE maintains a comparable performance with the current work.

Keywords: Private Comparison · Homomorphic Encryption · Private Decision Tree Evaluation

1 Introduction

Machine learning has been widely used and has well performance in services such as healthcare, financial services, and data classification [4, 36, 31]. Typically, users need to provide their own sensitive information to use these services. Therefore, users may worry about their private data being collected without effective supervision then choose to avoid using such services, and service providers also hope that users' private data will be properly managed and ensuring the widespread use of their services. Additionally, service providers also aim to prevent the leakage of critical parameters, which could result in the illegal distribution and abuse of their services.

As an important classification model in machine learning, decision tree [29, 28] are widely used because of their simplicity, interpretability, and ease of training. And the private decision tree evaluation (PDTE) is proposed to enhance the privacy security of decision tree. When clients utilize the server's decision tree for classification, they receive the classification result without revealing their own data, and the server does not disclose critical parameters about their decision tree.

The private comparison algorithm is the core component of PDTE, exhibits nonlinearity and atomicity, making it suitable for use in multiple ways. In private comparison, both participants aim to compare their respective numbers to determine which one is bigger. Similar to the millionaire's problem [38], there are numerous solutions available now, including interactive [18, 7] and non-interactive [16, 35] approaches. In practical applications, private comparison have four important metrics: non-interactive, efficiency, high-precision, and output expressive (the comparison result can be used in subsequent computations).

In this paper, we focus on the secure and efficient PDTE. Firstly, constructing private comparison algorithms with balancing these metrics. Secondly, we implement the PDTE schemes using our private comparison to further enhance its performance.

Existing solutions of PDTE can be categorized into two types: interactive [6, 21, 24] and non-interactive [1, 25, 34, 14, 5] which mainly depend on the private comparison they used. The first type involves interactive communication, utilizing techniques such as secure multiparty computation, secret sharing, and garbled circuits to complete PDTE through multiple rounds of communication. The second type is non-interactive, requiring only a single round of communication to complete PDTE, based on techniques such as leveled homomorphic encryption (LHE) or fully homomorphic encryption (FHE). Interactive schemes require higher bandwidth and simultaneous online requirements for both client and server, while non-interactive schemes demand a higher computational capabilities, considering limitations on multiplication depth in LHE or the heavy bootstrapping cost in FHE [13]. There are some practical non-interactive PDTE schemes, such as SortingHat [14] and Level Up [2]. SortingHat has a very fast computation speed, but it is limited to low precision (under 16-bit). Level up has two approaches, XXCMP-PDTE and RCC-PDTE, offering 36-bit and 128-bit of precision while maintaining a low computation. For a single row data in the tree with over 1000 nodes in 10-bit, SortingHat, XXCMP-PDTE and RCC-PDTE require 5s, 3s, and 12s on a single core. However, when facing the demand to classify massive datasets with millions of rows, existing schemes are not efficient.

In general sense, PDTE includes three participants: model party, data party and third party. The model party owns the model, the data party owns the data, and the third party possesses evaluation capability. Additionally, there are three roles: encryptor, decryptor and evaluator. The encryptor has the encrypt key, decryptor has decrypt key, and the evaluator holds the evaluation key. In most scenarios, the client represents the data party with an encryptor and decryptor, while the server represents both the model party and the third party with an evaluator. This is also how our client and server are configured.

During the evaluation process of private decision tree, the evaluator start from the root node, assigning state values to the child nodes based on the comparison results at each parent node using information from model party and data party, until reaching the leaf nodes to obtain their state values, thereby deriving the classification results and send them to the data party. This process can be divided into three components: node comparison, node selection, and tree traversal, We discuss each component and optimize them to enhance the overall performance.

Node comparison is the core building block. The evaluator identifies the threshold and attribute based on the information provided by the model party and the data party. Then the private comparison takes over, deriving comparison results based on the threshold and attribute. A private comparison that is efficient, non-interactive, output-expressive [25], and high-precision can translate these advantages into the virtues of node comparison. Efficiency allows the Node comparison to achieve relatively low costs in computation and communication. Non-interactive means that the evaluator does not need to interact with the other parties during the comparison process. Output expressive ensures that the comparison results can be computed subsequently without requiring additional operations. High precision refers to the ability to compare more digits simultaneously.

In this paper, we present three efficient private comparison algorithms in Table 1: TECMP, RDCMP, and CDCMP, which are non-interactive, amortized efficient, output expressive and high-precision. TECMP utilizes the thermometer encoding [9], based on batch operations in LHE such as BFV [20]. RDCMP and CDCMP achieve a balance between multiplication depth and multiplicative complexity by improving the FOLKLORE [2] in recursive and dichotomy way. Private comparisons have a significant influence on the computation cost of PDTE, it also determines the encoding method of data input and the output length of comparison results, so their impact on communication consumption is also significant. Prior to this, the bottleneck of many works were the private comparison. Sortinghat was limited in output expressive and bit precision. Level up's XXCMP and RCC offer high-percision and lower runtime costs. Our private comparison have high-precision (up to 26624-bit), amortized runtime are $13\times$, $42\times$, and $7\times$ faster than the RCC algorithm for 32-bit numbers. Additionally, the amortized communication cost compare with RCC is reduced by $20\times$, $14\times$, and $37\times$ respectively.

In node selection, as shown in Figure 1, the state value of the child node is obtained by comparing the threshold with the attribute. Following the Sortinghat method, at least one multiplication and subtraction are required in each intermediate node.

This paper are aim to evaluates multiple rows of data at once through appropriate batch encoding. After the node comparsion, the comparison results of attributes in the same row are placed in the same slot

Table 1. Private comparison, the a,b represent the ciphertext of a,b

Methods	inputs	bits	expressive	batchable
XCMP [25]	a b, a b	16	ADD limited MUL.	no
SortingHat [14]	a b	12	ADD limited MUL.	no
XXCMP [2]	a b	36	ADD limited MUL.	yes
RCC [2]	a b, a b	287	ADD MUL.	yes
FOLKLORE [2]	a b, a b	1024	ADD MUL.	yes
TECMP	a b	26624	ADD MUL.	yes
RDCMP	a b, a b	1024	ADD MUL.	yes
CDCMP	a b, a b	287	ADD MUL.	yes

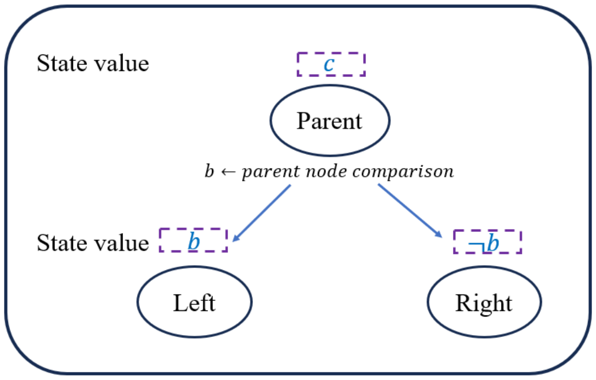

Fig. 1. Node selection

position of different ciphertexts, while the comparison results of different rows have different slot positions. The multiple rows of data only requiring one same operation results with a lower amortization time.

In tree traversal, the classification results of leaf nodes are obtained based on the state values of all nodes in the tree. There are three methods for utilizing the state values to compute the classification results: SumPath, MulPath [21] and SinglePath [5]. SumPath adds the state values from each path from the root to the leaves to refresh the state value of the leaves. MulPath multiply the state values from each path from the root to the leaves to refresh the state value of the leaves. SinglePath, starting from the layer where the root is located, determines the state values of the next layer based on the state values in this layer, then use the state value of leaves to determine the classification result.

In Figure 2, we summary the relation in tree traversal methods:

- SumPath: $c_{leaf_i} = \sum_{node_j \in \{root \to \ldots \to leaf_i\}} c_{node_j}$
- MulPath: $c_{leaf_i} = \prod_{node_j \in \{root \to \ldots \to leaf_i\}} c_{node_j}$
- SinglePath: $root \to node_{1*} \to node_{2*} \to \ldots \to leaf_i$

Each of the methods has advantages and disadvantages. In Sumpath, addition and subtraction operations are required, while the tree traversal does not consume the multiplication depth, but the classification result is the multiple ciphertexts. Multiplication is required in MulPath, but the classification result is a single ciphertext. In each layer of SinglePath, only one node comparison is required, but determining which one in this layer may require a larger workload. In LHE, multiplication depth is a precious resource. During tree traversal, MulPath inevitably requires the multiplication depth of the tree depth. Currently, there is no suitable method under BFV for node comparsion at each level in SinglePath. Therefore, we chose SumPath in this paper to avoid consuming the multiplication depth.

We observed that the leaf state values of SumPath follow a binomial distribution in the perfect binary tree, with only one state value being 0. To conceal this distribution, we apply masking by multiplying them with a random number like level up [2]. At the same time, the state value with 0 remains unchanged, and the classification result can be found based on the position of this 0. This will cause a problem, when classifying the SumPath under batch encoding of multiple rows of data, classification result is actually the position of

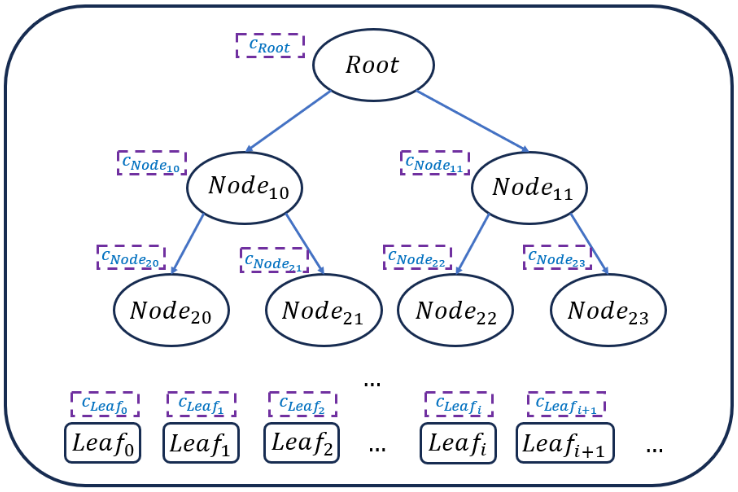

Fig. 2. Tree traversal, the nodes of the tree are numbered by layer with the root node at level 0. Numbering starts from left to right, where c represents the state value.

the leaf node. It is necessary to confuse the classification result and the position of the classification result, so we designed the clear rows relation (CRR) algorithm to clear the inter-row relationship, which disrupts the relationship between the classification result and the leaf node position through $O(logn)$ times permutations and cipher-plain multiplication. to ensure the probability that the classification result of n rows of data is associated with the position of the leaf node with a probability around $1/n$.

Contributions. Taken together, our contributions are as follows:

The first is private comparison.

- **High precision**: By appropriately encoding strategy, we can achieve a comparison precision of 26624 bits when the multiplication depth of LHE is less than 12.
- **Amortized efficient**: By encoding multiple data into one ciphertext, we can perform multiple comparisons simultaneously. This results in highly efficient amortized computation costs and significantly reduces the amortized communication consumption.
- **Output expressive**: The comparison results are in the same slot, the subsequent addition or multiplication can be performed.
- **Arbitrary input**: It supports cipher-plain, cipher-cipher input, and many-to-one, many-to-many input.

Secondly, in terms of PDTE.

- **Much faster**: We propose the PDTE schemes that combines our private comparison with SumPath. In a tree with 16-bit and more than 1000 nodes, TECMP-PDTE and RDCMP-PDTE achieve an amortized cost that is more than 56 times faster than the current fastest solution, XXCMP, while the TECMP-PDTE with CRR algorithm remains 14 times faster than the state-of-the-art.
- **Security enhanced**: The security of SumPath with batch encoding is enhanced. During batch evaluation of multiple rows of data, the output conceals both the classification results and the positions of leaf nodes.

In Section 2, we provide an overview of existing work, which is broadly categorized into interactive and non-interactive approaches. Section 3 covers fundamental concepts, including communication model, decision tree, leveled homomorphic encryption, and folklore compare. In Section 4, we introduce the thermometer encoding, optimization method of folklore algorithm, then design the private comparisons: TECMP, RDCMP, and CDCMP. Section 5 introduces the batching methods and CCR algorithm, presenting our PDTE scheme: TECMP-PDTE, RDCMP-PDTE, and CDCMP-PDTE. Section 6 presents the benchmark of runtime and communication costs of our private comparison and PDTE, comparing them with the current solutions such as Sortinghat [14] and Level up [2]. In Sections 7 and 8, we discuss the current work and future research directions.

2 Related Work

Private decision tree evaluation can be considered as two-party secure computation or multi-party secure computation. It can utilize methods such as secret sharing, garbled circuits, and oblivious transfer to address it [6, 8, 21, 24] involving interaction. In [7], Bost et al. used multivariate polynomials to obtain the classification results through interactive communication. Tueno et al. [34] employed Oblivious RAM, requiring d-round communication. Liu et al. [23] trained and evaluated on sensitive data, giving an interactive method based on function secret sharing.

Furthermore, Tai et al. [32] employ the private comparison proposed in [15, 37], and tree traversal uses the SumPath. However, the comparison result needs to be decrypted for subsequent computations, which requires additional interaction, thereby increasing the communication cost. In the work of Nateghizad et al. [27] use DGK and Paillier cryptosystem, but decryption is required before the subsequent processing proceeds. Kiss et al. [21] abstracted certain designed modules, such as fundamental components including node selection, comparison, and path evaluation. As we can see, using secure multi-party computation methods requires multiple rounds of interactions and a significant communication load. In practice, more situations prefer to use non-interactive to accomplish the PDTE scheme.

The non-interactive PDTE requires each basic component to be non-interactive. In its key component, private comparison, XCMP [25] encodes data into polynomial coefficients, extracts constant terms as comparison results with only one multiplication, is very fast, and non-interactive. However, the comparison result does not satisfy the output expressive. It can only perform addition and scalar multiplication, cannot perform multiplication in two comparison results, which limits the subsequent computations. In addition, the comparison precision of XCMP is constrained by the degree of the polynomial, which is less than or equal to the degree of the polynomial, therefore, private comparisons with small bits ≤ 16 can be used, and it leads to significant communication wastage.

Subsequently, PDT-BIN by Tueno et al. [33] employed the private comparison from [22] and [11], while PDT-INT utilized the private comparison from [12]. Both approaches implement non-interactive PDTE, but their overall computational efficiency is low.

SorthingHat [14] proposed an efficient non-interactive PDTE, using the PolyComp algorithm, which is an improved version of XCMP [25] with a very fast speed, but is limited by comparison precision. In addition, the comparison results of PolyComp is RLWE ciphertext, requires the operation of [10] to be converted into RGSW ciphertext for subsequent computations. This makes PolyComp inflexible in PDTE and does not perform well in high precision. Another PDTE based on gate circuit supports arbitrary precision and trees of arbitrary depth by using FHE, among them, the performance of the gate circuit private comparison with cipher-plain input is slightly lower compared to the cipher-cipher input. However, the overall cost is still far from practical application due to the heavy Bootstrapping operation.

PROBONITE [5] also employs the XCMP. Its characteristic is the tree traversal is SinglePath. Each layer uses blind rotation to find the comparison node, which aligns with the general rules for evaluating decision trees on plaintext. However, confirming the comparison node at each layer incurs significant consumption, and the computation cost still needs to be benchmarked.

Level Up [2] utilizes the extended precision version of XCMP called XXCMP [25] to compute the equation through extra operations [3]. Additionally, the RCC comparison utilizes methods from [19, 30] and [26], then further improvement. XXCMP supports cipher-plain input, RCC supports cipher-plain and cipher-cipher input, and is batchable. Combined with the SumPath method, its PDTE approach is highly efficient.

Note that the folklore algorithm mentioned in level up [2] is a commonly used method for private comparison in gate circuits. This is consistent with the concepts proposed in [11] and [22], as well as thecomparison in SortingHat [14]. Compute the greater than or equal to each bit in sequence, and then recursively calculate the final comparison result based on the comparison result of each bit.

So far, PDTE has evolved from interactive to non-interactive, and computational efficiency has gradually improved. Scholars have widely focused on finding efficient private comparison algorithms with output expressive. On the other hand, LHE, such as BFV combined with batch technology is very suitable for private comparison and PDTE design. Unlike the secure multi-party computation, LHE does not require multiple rounds of interaction. Also, unlike FHE, LHE does not necessitate heavy Bootstrapping, thereby achieving a balance between communication and computation. Moreover, decision trees, as a fundamental model, do not necessarily require highly nonlinear operations. Furthermore, LHE supports batching, which is another advantageous feature.

3 Preliminaries

Table 2 standardizes all symbols used throughout this paper. we use the function: $b \leftarrow I(statement)$, where $b \in \{0, 1\}$. If *statement* is true, $b = 1$; otherwise, $b = 0$.

Table 2. Summary of notation

Symbol	Description
λ	security parameter
N	polynomial degree
p	plaintext modulus
q	ciphertext modulus
R_q	polynomial ring with modulo q
Z_p^N	integers ring with modulo p in dimension N
d_{max}	max multiplication depth
$Rotate_k$	left rotate k slot
n	bits precision
$[n]$	$0, 1, ..., n-1$
B	$0, 1$
x	client attribute vector
len	client attribute vector length
$\mathcal{T}$	decision tree
$\mathcal{N}$	decision tree nodes set
$\mathcal{S}$	decision tree structure
$\mathcal{I}$	internal nodes set in $\mathcal{N}$
$\mathcal{L}$	leaf nodes set in $\mathcal{N}$
m	number of $\mathcal{N}$, $(\lvert\mathcal{N}\rvert)$
s	state value of node in $\mathcal{N}$
a	attribute index of node in $\mathcal{I}$
t	threshold of node in $\mathcal{I}$
v	classification result of node in $\mathcal{L}$
d	depth of $\mathcal{T}$

3.1 Stakeholders & Non-interactivity

In private decision tree evaluation (PDTE), if the private model is not a decision tree but another model, PDTE can generalize to the private model evaluation (PME). It considers three roles: encryptor, decryptor, and evaluator, each equipped with different keys for encryption, decryption, and evaluation. And it also involves three entities: data party, model party, and the third party. The data party owns the data, the model party owns the model, such as decision trees, linear polynomials, and random forests. Third party has sufficient computational power to handle the evaluation. The security goal is for the data party to get the model party's evaluation result without exposing their data privacy, while the model party keeps their model private, and the third party learn nothing.

In classical client-server scenario, the client as the data party with the roles of encryptor and decryptor, while the server as both the model party and the third party with the role of evaluator. This setup is employed in works such as $[14, 5, 2, 25]$, where the model used is typically a decision tree in their PDTE implementations. In another way, the client as the model party with the roles of encryptor and decryptor, while the server as the data party with the role of evaluator. This setup can also enable PDTE if suitable measures are implemented to safeguard the privacy of the model.

In client-cloud-server scenario, the client as the data party with the roles as encryptor and decryptor, while the cloud serves as the third party with the role of evaluator. The server as the model party with the role of encryptor. The cloud is assumed to have significant computational power, which can alleviate the computational burden on both the client and server.

For non-interactive PME, there should be no additional interactions beyond the initial send and final response.

In this paper, we employ the client-server scenario, where the private model is the decision tree, then we design the non-interactive PDTE schemes.

3.2 Decision Tree and PDTE

Decision tree is a typical classification algorithm that can classify a series of input attributes in order to obtain a classification result. Commonly seen in the binary decision tree, they consist of internal nodes and leaf nodes. The internal nodes contain thresholds and indices, while the leaf nodes contain classification results. The evaluation of decision tree begins with inputting a vector of attributes. The root node identifies the specific attribute based on its index and compares it with the threshold, depending on the comparison result, it traverses to either the left or right child, continuing in this manner until it reaches a leaf node. Finally, it returns the classification result associated with the leaf node.

We record the decision tree as $\mathcal{T} = (\mathcal{N}, \mathcal{S})$, a node set $\mathcal{N} = \mathcal{I} + \mathcal{L}$ with a tree structure $\mathcal{S}$, where $\mathcal{I}$ represents the internal node set and $\mathcal{L}$ represents the leaf node set. Each internal node contains a threshold t, attribute index a and state value s, and each leaf node contains a classification result v and state value s. The tree structure S records the two child nodes of the parent node. Its input attribute vector is $\boldsymbol{x}$, and its output is a classification result v^*.

PDTE means that under the client-server model, the server holds a decision tree $\mathcal{T}$ and the client holds the private data $\boldsymbol{x}$. The client uses the server's decision tree for evaluation, obtains the evaluation results without knowing anything else about it, and keeping its own data privacy. The server does not leak its own private decision tree.

In the PDTE, it is mainly divided into three parts: node comparison, node selection and tree traversal. The node comparison is performed in the internal node, inputting the attribute vector $\boldsymbol{x}$, and based on the attribute index a of the internal node and the threshold value t. The comparison result is $c = I(x[a] > t), c \in B$. Node selection is based on the comparison result c and the tree structure S. It updates the state value of the left and right child nodes, assigning 1 to one child node's state value and assigning 0 to the other. Tree traversal begins from the root, updating the leaf's state values based on all node's state values. Then, it returns the classification result v^* based on the state values of the leaf nodes. Different from the decision tree evaluation, PDTE cannot distinguish which internal node is chosen, and which leaf node's classification result is returned.

3.3 Leveled Homomorphic Encryption with Batching

Homomorphic encryption processes the plaintext on its ciphertext. Leveled homomorphic encryption (LHE) is limited with cipher-cipher multiplication, each operation in LHE increases the noise in the ciphertext, when the noise level in the ciphertext reaches a certain threshold, it will result in decryption failure. The noise growth of cipher-cipher multiplication is exponential, and the other operations is linear. The multiplication depth of LHE mainly depended on the depth of cipher-cipher multiplication, which can be configured using a suitable parameter.

LHE can avoid the heavy Bootstrapping, thereby achieving higher computational efficiency. BFV is a LHE based on Ring Learning With Errors (RLWE) problem, its ciphertext consists of two elements from the R_q, while the corresponding plaintext resides in the ring R_p, and the Chinese Remainder Theorem (CRT) can be utilized to perform batch operations. In batch operations, while the ciphertext remains unchanged, the corresponding plaintext becomes a vector in $2 \times \mathbb{Z}_p^{N/2}$, as Table 3 shows:

The plaintext of BFV is an array of $x \in Z_p^N$, which in a strict sense is

$$x = \begin{bmatrix} x_a \\ x_b \end{bmatrix} = \begin{bmatrix} x_{a,0}, x_{a,1}, ..., x_{a,N/2-1} \\ x_{b,0}, x_{b,1}, ..., x_{b,N/2-1} \end{bmatrix}, x_{a,i}, y_{b,i} \in Z_p \tag{1}$$

Then

Table 3. Batched BFV operation, $x, y \in 2 * Z_p^{N/2}$, the x, y represent the ciphertext of x, y.

operation	cipher	plain
cipher-plain add	$x \oplus y$	$x + y$
cipher-cipher add	$x \oplus y$	$x + y$
cipher-plain sub.	$x \ominus y$	$x - y$
cipher-cipher sub.	$x \ominus y$	$x - y$
cipher-cipher mul.	$x \odot y$	$x \cdot y$
cipher-plain mul.	$x \odot y$	$x \cdot y$
left rotate	$Rotate_k(x)$	$Rotate_k(x)$

$$z = x + y = \begin{bmatrix} z_a \\ z_b \end{bmatrix} = \begin{bmatrix} x_{a,0} + y_{a,0}, x_{a,1} + y_{a,1}, ..., x_{a,N/2-1} + y_{a,N/2-1} \\ x_{b,0} + y_{b,0}, x_{b,1} + y_{b,1}, ..., x_{b,N/2-1} + y_{b,N/2-1} \end{bmatrix}$$
$$z = x \cdot y = \begin{bmatrix} z_a \\ z_b \end{bmatrix} = \begin{bmatrix} x_{a,0} \cdot y_{a,0}, x_{a,1} \cdot y_{a,1}, ..., x_{a,N/2-1} \cdot y_{a,N/2-1} \\ x_{b,0} \cdot y_{b,0}, x_{b,1} \cdot y_{b,1}, ..., x_{b,N/2-1} \cdot y_{b,N/2-1} \end{bmatrix}$$
$$Rotate_k(x) = \begin{bmatrix} x_{a,k}, x_{a,k+1}, ..., x_{a,N/2-1}, x_{a,0}, ..., x_{a,k-1} \\ x_{b,k}, x_{b,k+1}, ..., x_{b,N/2-1}, x_{b,0}, ..., x_{b,k-1} \end{bmatrix}$$

3.4 Basic Idea of Folklore

The numbers are represented in binary representation from the least significant bit (LSB) to the most significant bit (MSB).

$$\boldsymbol{a} = \{a_0, a_1, ..., a_{n-1}\}, a_i \in \{0, 1\} \tag{2}$$
$$\boldsymbol{b} = \{b_0, b_1, ..., b_{n-1}\}, b_i \in \{0, 1\} \tag{3}$$

The process involves calculating the comparison result bit by bit:

$$\boldsymbol{g} = \{g_0, g_1, ..., g_{n-1}\}, g_i = I(a_i > b_i) = a_i \cdot (1 - b_i) \tag{4}$$
$$\boldsymbol{e} = \{e_0, e_1, ..., e_{n-1}\}, e_i = I(a_i = b_i) = 1 - (a_i - b_i)^2 \tag{5}$$

Then the comparsion result is:

$$\theta = I(a > b) = \sum_{i=0}^{n-1} g_i \cdot \prod_{k=i+1}^{n-1} e_k \tag{6}$$

Each multiplication in n ciphertext can use dichotomy to reduce the multiplication depth to $log(n-1)+1$, so the multiplication depth of forklore algorithm is $log(n-1)+2$.

4 Amortized Private Comparison

In this section, we propose two types of non-interactive amortized comparison based on LHE with Batch operations. The first is TECMP, which uses thermometer encoding combined with folklore. The second are RDCMP and CDCMP, a class of improved folklore algorithm using recursive and dichotomy method.

4.1 Thermometer Encoding

Thermometer Encoding (TE) [9] is an extension of One-Hot encoding, when there is a sequential relationship between variables, it can effectively retain the sorting information.

During the comparison of two numbers, if $I(a > b) = 1, a, b \in [2^n]$, then

$$I(x > b) = 1, x \in \{a, a+1, ..., 2^n - 1\} \tag{7}$$

Algorithm 1 Thermometer Encoding

```
1: Procedure : $TE(b, n)$
2: $\theta \leftarrow 1^{2^n}$;
3: for all $i \in [b]$ do
4:     $\theta[i] = 0$;
5: end for
6: return $\theta$.
```

Therefore, we use thermometer encoding to record these comparison information, as shown in Algorithm 1:

$$\theta = \{\theta_0, \theta_1, ..., \theta_{2^n-1}\} \leftarrow TE(b, n), \theta_i = I(i > b), i, b \in [2^n] \tag{8}$$

In XCMP [25], the comparison numbers a and b are respectively encoded as $X^a, T = -(1 + X + ... + X^{N-b-1})$, multiply them then obtain the $-(X^a + X^{a+1} + ... + X^{N+a-b-1}) mod(X^N - 1)$. If $a > b$, then the constant is 1, otherwise is 0. the TE is somewhat similar with XCMP. The difference is that the plaintext of TE in Z_q, not in R_q, and the rotate instead of multiplication. As shown in Algorithm 2.

Algorithm 2 Greater Than $I(a > b)$

```
1: Pre-procedure :
2: $\theta \leftarrow TE(b, n)$;
3: Procedure : $GT(a, \theta, n)$
4: $\theta_{gt} \leftarrow Rotate_a(\theta)$;
5: return $\theta_{gt}$.
```

Then, we can obtain the result of $I(a = b)$ through Algorithm 3.

Algorithm 3 Equal To $I(a = b)$

```
1: Pre-procedure :
2: $\theta \leftarrow TE(b, n)$;
3: $\theta_{gt} \leftarrow GT(a, \theta, n)$;
4: $W \leftarrow 0^{2^n}, W[0] = 1$;
5: Procedure : $ET(a, \theta_{gt}, W, n)$
6: if $a < 2^n - 1$ then
7:     $\theta'_{gt} \leftarrow Rotate_1(\theta_{gt})$;
8: else
9:     $\theta'_{gt} \leftarrow W$;
10: end if
11: $\theta_{eq} \leftarrow \theta'_{gt} \ominus \theta_{gt}$;
12: return $\theta_{eq}$.
```

Through the combination of TE and rotation, we can compute $I(a > b)$ and $I(a = b)$, both of the results are located at the 0th slot of the ciphertext. Additionally, since we only performed rotation, some information is still retained in other positions, so the final output needs to be clean up. This entails multiplying by a plaintext where the 0th slot is 1 and the remaining positions are 0, as shown in Algorithm 4.

Lemma 1. *Algorithm 2 and Algorithm 3 are correctness.*

Proof. Thermometer Encoding

$$\theta = [I(0 > b), I(1 > b), ..., I(2^n - 1) > b] \tag{9}$$

Algorithm 4 Clear Up

```
1: Pre-procedure :
2: $W \leftarrow 0^{2^n}, W[0] = 1$;
3: Procedure : $CU(\theta_{in}, W)$
4: $\theta_{out} \leftarrow \theta_{in} \otimes W$;
5: return $\theta_{out}$.
```

Rotate a slots is:

$$\theta_{gt} = [I(a > b), I(a+1 > b), ..., I(2^n - 1 > b), I(0 > b), ..., I(a-1 > b)] \tag{10}$$

Then the 0th slot of θ_{gt} is $I(a > b)$.
When $a < 2^n - 1$, θ_{gt} Rotate 1 slot is:

$$\theta'_{gt} = [I(a+1 > b), I(a+2 > b), ..., I(2^n - 1 > b), I(0 > b), ..., I(a > b)] \tag{11}$$

When $a = 2^n - 1$, $a + 1 = 2^n, b \in [2^n]$, so $b < a + 1, I(a + 1 > b) = 1$,

$$\theta'_{gt} = [1, 0, 0, ..., 0] = [I(a+1 > b), 0, 0, ..., 0] \tag{12}$$

The 0th slot of θ'_{gt} is $I(a + 1 > b)$.
Calculate $\theta_{eq} \leftarrow \theta'_{gt} \ominus \theta_{gt}$, and

$$I(a = b) = I(a + 1 > b) - I(a > b) \tag{13}$$

So the 0th slot of θ_{eq} is $I(a = b)$.

However, the encoded information is exponential, which means that when we compare 8-bit number, we need 256 slots to store this information, and our number of slots has an upper limit, such as 2^{14}, then one ciphertext up to 13-bit number can be compared.

Finally, in XCMP, a difficulty is that subsequent multiplication operations cannot be performed on the comparison results. Therefore, the result of $I(a = b)$ is not compatible with the $I(a > b)$, thus preventing further precision expansion.

$$a = a_1 \cdot 2^n + a_0, a_i \in [2^n] \tag{14}$$

$$b = b_1 \cdot 2^n + b_0, b_i \in [2^n] \tag{15}$$

$$I(a > b) = I(a_1 > b_1) + I(a_1 = b_1) \cdot I(a_0 > b_0) \tag{16}$$

In TE, this problem becomes simpler. The result of $I(a > b)$ is in the 0th slot and the value is 1 or 0. It can directly addition and multiplication of numbers within the same 0th slot without need for extra operations. Alternatively, performing some rotations for addition and multiplication with numbers in other slots.

4.2 Minor Optimized of Folklore Algorithm

Now, we slightly optimize the folklore algorithm to complete our private comparison. Observe the computations in folklore is $\theta = \theta_{norm}$, and θ can be computed in recursive and dichotomy way.

$$\theta_{norm} = \sum_{i=0}^{n-1} g_i \cdot \prod_{k=i+1}^{n-1} e_k \tag{17}$$

$$\theta_{rec} = \theta_{n-1}, \theta_{n-1} = g_{n-1} + e_{n-1} \cdot \theta_{n-2}, \theta_0 = g_0 \tag{18}$$

$$\theta_{dic} = g_{\{s+1,s+2,...,n-1\}} + e_{\{s+1,s+2,...,n-1\}} \cdot g_{\{0,1,...,s\}} \tag{19}$$

$$g_{\{i,i+1,..,j\}} = I(a_i a_{i+1} ... a_j > b_i b_{i+1} ... b_j) \tag{20}$$

$$e_{\{i,i+1,..,j\}} = I(a_i a_{i+1} ... a_j = b_i b_{i+1} ... b_j) \tag{21}$$

Algorithm 5 Recursive Method

```
1: Procedure : $(\theta_{gt}, \theta_{eq}, n)$
2: $\theta_{rec} = \theta_{gt}[0]$
3: for all $i \in 1, 2, ..., n-1$ do
4:     $\theta_{rec} \leftarrow \theta_{rec} \odot \theta_{eq}[i]$;
5:     $\theta_{rec} \leftarrow \theta_{rec} \oplus \theta_{gt}[i]$;
6: end for
7: return $\theta_{rec}$.
```

Algorithm 6 Dichotomy Method, n is power of 2

```
1: Procedure : $DM(\theta_{gt}, \theta_{eq}, n)$
2: $d = log(n)$;
3: for all $i \in [d]$ do
4:     $t = 1 \ll i$;
5:     $w = 1 \ll (i+1)$;
6:     for all $j \in \{0, w, 2 \cdot w, ..., n-w\}$ do
7:         $\theta_{gt}[j] \leftarrow \theta_{gt}[j] \odot \theta_{eq}[j+t]$;
8:         $\theta_{gt}[j] \leftarrow \theta_{gt}[j] \oplus \theta_{gt}[j+t]$;
9:         $\theta_{eq}[j] \leftarrow \theta_{eq}[j] \odot \theta_{eq}[j+t]$;
10:     end for
11: end for
12: $\theta_{dic} = \theta_{gt}[0]$;
13: return $\theta_{dic}$.
```

After compute each $g_i, e_i, i \in [n]$. There are three methods to compute $\theta = I(a > b)$. First, the folklore, using a normal approach, which is brute force computation. Second, the recursive way, computing sequentially from high bit to low bit. Third, the dichotomy, computing by dichotomy from the middle. In algorithm 5 and algorithm 6, we set n to a power of 2 without losing generality.

Lemma 2. *The dichotomy method of compute $\theta_{dic} = I(a > b)$ is correctness.*

Proof.

$$\boldsymbol{a} = \{a_0, a_1, ..., a_{n-1}\}, a_i \in \{0, 1\} \tag{22}$$

$$\boldsymbol{b} = \{b_0, b_1, ..., b_{n-1}\}, b_i \in \{0, 1\} \tag{23}$$

Where

$$a = a_0 + a_1 \cdot 2 + ... + a_{n-1} \cdot 2^{n-1} \tag{24}$$

$$b = b_0 + b_1 \cdot 2 + ... + b_{n-1} \cdot 2^{n-1} \tag{25}$$

If $0 \le r \le s < t \le n, r, s, t \in [n]$

$$g_0 = I(a_r a_{r+1} ... a_s > b_r b_{r+1} ... b_s) \tag{26}$$

$$e_0 = I(a_r a_{r+1} ... a_s = b_r b_{r+1} ... b_s) \tag{27}$$

$$g_1 = I(a_{s+1} a_{s+2} ... a_t > b_{s+1} b_{s+2} ... b_t) \tag{28}$$

$$e_1 = I(a_{s+1} a_{s+2} ... a_t = b_{s+1} b_{s+2} ... b_t) \tag{29}$$

Then

$$g = g_1 + e_1 \cdot g_0 = I(a_r a_{r+1} ... a_t > b_r b_{r+1} ... b_t) \tag{30}$$

$$e = e_1 \cdot e_0 = I(a_r a_{r+1} ... a_t = b_r b_{r+1} ... b_t) \tag{31}$$

For $n = 8$-bit, a dichotomy example, as shown in the Figure 3:

θ_{gt}	g_0	g_1	g_2	g_3	g_4	g_5	g_6	g_7
θ_{eq}	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
$g_i^1 = g_{i+1} + e_{i+1} \cdot g_i$	g_0^1		g_2^1		g_4^1		g_6^1	
$e_i^1 = e_{i+1} \cdot e_i$	e_0^1		e_2^1		e_4^1		e_6^1	
$g_i^2 = g_{i+2}^1 + e_{i+2}^1 \cdot g_i^1$	g_0^2				g_4^2			
$e_i^2 = e_{i+2}^1 \cdot e_i^1$	e_0^2				e_4^2			
$g_i^3 = g_{i+4}^2 + e_{i+4}^2 \cdot g_i^2$	g_0^3							

Fig. 3. Dichotomy Example

In original folklore, where $g_i = a_i \cdot (1 - b_i), e_i = 1 - (a_i - b_i)^2$, it can optimize the encode then computation as:

$$\boldsymbol{a} = \{a_0, a_1, ..., a_{n-1}\}, a_i \in \{0, 1\} \tag{32}$$
$$\boldsymbol{b} = \{b'_0, b'_1, ..., b'_{n-1}\}, b'_i = 1 - b_i \in \{0, 1\} \tag{33}$$

$$g_i = a_i \cdot b'_i \tag{34}$$
$$e_i = 1 - (a_i - b_i)^2 \tag{35}$$
$$= 1 - a_i^2 - b_i^2 + \cdot a_i \cdot b_i \tag{36}$$
$$= 1 - a_i - b_i + 2 \cdot a_i \cdot b_i \tag{37}$$
$$= 1 - b_i - 2 \cdot a_i \cdot (1 - b_i) + a_i \tag{38}$$
$$= b'_i - 2 \cdot a_i \cdot b'_i + a_i \tag{39}$$
$$= b'_i - 2 \cdot g_i + a_i \tag{40}$$

At most one cipher-cipher multiplication and one multiplication depth are required. Note that each slot of the plaintext is modulus $p \geq 65537$, so there will be no risk of data overflow in the optimized calculation steps.

4.3 Thermometer Encoding Compare (TECMP)

The comparison result of thermometer encoding is output expressive, allowing for continued multiplication and addition. So we next perform precision promotion on thermometer encoding.

For folklore, as long as g_i and e_i of the corresponding bits are calculated, then θ can be calculated in the same way. And the corresponding numbers do not need to be binary expansion, it can be quaternary or octal expansion, so each block can be m-bit length too. We perform thermometer encoding on the m-bit length block and compute the g_i, e_i, then we can compare numbers with $m \cdot 2^{d_{max}}$ bit length by normal and dichotomy way, $m \cdot d_{max}$ bit length by recursive way.

Firstly, the numbers are expanded by m-bit, thermometer encoding b_i.

$$\boldsymbol{a} = \{a_0, a_1, ..., a_{l-1}\}, a_i \in [2^m] \tag{41}$$
$$\boldsymbol{b} = \{b_0, b_1, ..., b_{l-1}\}, b_i \in [2^m] \tag{42}$$

Secondly, via algorithm 2 and algorithm 3 calculation each $g_i = I(a_i > b_i), e_i = I(a_i = b_i)$.

Finally, using a normal, or recursive, or dichotomy way to compute the $I(a > b)$, as shown in Algorithm 7 in dichotomy method.

In algorithm 7, l is set as a power of 2 without losing generality. we utilize algorithm 6 to compute $I(a > b)$ because the multiplication depth increases slowly. Meanwhile, it's worth noting that algorithm 5 requires the fewest number of multiplications and will effectively in a low multiplication depth.

Algorithm 7 Thermometer Encoding Compare $I(a > b), a, b \in [2^{m \cdot l}]$, l is power of 2

```
1: Pre-procedure :
2: $W \leftarrow 0^{2^m}, W[0] = 1$;
3: for all $i \in [l]$ do
4:     $a_i \leftarrow (a \gg i \cdot m)\&(2^m - 1)$;
5:     $b_i \leftarrow (b \gg i \cdot m)\&(2^m - 1)$;
6:     $\theta_i \leftarrow TE(b_i, m)$;
7: end for
8: Procedure : $TECMP(\{a_0, a_1, ..., a_{l-1}\}, \{\theta_0, \theta_1, ..., \theta_{l-1}\}, l)$
9: for all $i \in [l]$ do
10:     $\theta_{gt}[i] \leftarrow GT(a_i, \theta_i, l)$;
11:     $\theta_{eq}[i] \leftarrow ET(a_i, \theta_{gt}[i], W, l)$;
12: end for
13: $\theta_{TECMP} = DM(\theta_{gt}, \theta_{eq}, l)$;
14: return $\theta_{TECMP}$.
```

In summary, TECMP effectively combines the advantages of folklore and thermometer encoding. It does not require excessive multiplication to reduce the multiplication depth, and mitigate the exponential slot expansion. Balancing the number of multiplications, multiplication depth and the slot numbers. Moreover, it can perform high-precision comparison and batch operations, making the amortized computing cost highly efficient. However, it only supports cipher-plain input and many-to-one scenarios, which may limit its applicability in various contexts. Therefore, we propose another type of comparison algorithm, RDCMP and CDCMP, to further enhance the versatility of the private comparison.

4.4 Row Dichotomy Compare (RDCMP)

To address the arbitrary input problem of TECMP, we propose a special case where $m = 1$. In this case, we compute the g_i, e_i using the optimized folklore, which does not require rotation to compute, thus eliminating the need for plaintext. Meanwhile, in the LHE, multiplication depth is a valuable resource, therefore, we use the dichotomy method to maintain the original multiplication depth $logn$ of folklore(normal) and reduce the number of multiplications as much as possible.

Algorithm 8 Row Dichotomy Compare $I(a > b), a, b \in [2^n]$, n is power of 2

```
1: Pre-procedure :
2: for all $i \in [n]$ do
3:     $a_i \leftarrow (a \gg i)\&1$;
4:     $b'_i \leftarrow 1 - (b \gg i)\&1$;
5: end for
6: Procedure : $RDCMP(\{a_0, a_1, ..., a_{n-1}\}, \{b'_0, b'_1, ..., b'_{n-1}\}, n)$
7: for all $i \in [n]$ do
8:     $\theta_{gt}[i] \leftarrow a_i \odot b'_i$;
9:     $\theta_{eq}[i] \leftarrow b'_i \ominus \theta_{gt}[i] \ominus \theta_{gt}[i] \oplus a_i$;
10: end for
11: $\theta_{RDCMP} = DM(\theta_{gt}, \theta_{eq}, n)$;
12: return $\theta_{RDCMP}$.
```

n is set as a power of 2, usually 8, 16, 32, ..., 1024, we emphasize that it is enough. Even if n is not a power of 2, algorithm 8 can be adaptively adjusted.

The dichotomy form used by RDCMP can effectively reduce the multiplication depth. Since the multiplication depth is $logn + 1$, we support high-precision bit length comparison, and the number of multiplications is $3n - 2$, which is much lower than $n \cdot (n - 1)/2$.

In Figure 4 RDCMP encoding, considering the packaging strategy, if we package $\{a_0, a_1, , a_{n-1}\}$ and $\{b_0, b_1, , b_{n-1}\}$ into n ciphertexts respectively, then we can do it in the same position.

4.5 Column Dichotomy Compare (CDCMP)

Additionally, considering that n is very large, the number of ciphertexts in the RDCMP packaging strategy is also n, and N numbers are compared at the same time. In some case, we hope that the number of ciphertexts will be fewer. Therefore, we pack $a_0, a_1, ..., a_{n-1}$ into one ciphertext as shown in Algorithm 9.

Algorithm 9 Column Dichotomy Compare $I(a > b), a, b \in [2^n]$, n is power of 2

```
1: **Pre-procedure :**
2: **for all** $i \in [n]$ **do**
3:     $a_i \leftarrow (a \gg i)\&1$;
4:     $b'_i \leftarrow 1 - (b \gg i)\&1$;
5: **end for**
6: **Procedure :** $CDCMP(\{a_0, a_1, ..., a_{n-1}\}, \{b'_0, b'_1, ..., b'_{n-1}\}, n)$
7: $\theta_{gt} \leftarrow a \odot b'$;
8: $\theta_{eq} \leftarrow b' \ominus \theta_{gt} \ominus \theta_{gt} \oplus a$;
9: $d = log(n)$;
10: **for all** $i \in [d]$ **do**
11:     $t = 1 \ll i$;
12:     $\theta'_{gt} \leftarrow Rotate_t \theta_{gt}$;
13:     $\theta'_{eq} \leftarrow Rotate_t \theta_{eq}$;
14:     $\theta_{eq} \leftarrow \theta_{eq} \odot \theta'_{eq}$;
15:     $\theta_{gt} \leftarrow \theta_{gt} \odot \theta'_{eq}$;
16:     $\theta_{gt} \leftarrow \theta_{gt} \oplus \theta'_{gt}$;
17: **end for**
18: $\theta_{CDCMP} = \theta_{gt}$;
19: **return** $\theta_{CDCMP}$.
```

Compared with RDCMP, CDCMP mainly solves the problem of packed ciphertexts size. it only has two ciphertexts to compare and fewer multiplications.

Table 4. Private Comparison Attributes

Symbol	TECMP	RDCMP	CDCMP
Bit precision	$l \cdot m$	n	n
Mul. depth	$log(l)$	$log(n) + 1$	$log(n) + 1$
Poly. degree	N	N	N
Cipher num.	l	$2 \cdot n$	2
Mul. times	$2 \cdot (n - 1)$	$n + 2 \cdot (n - 1)$	$1 + 2 \cdot log(n)$
Rotate times	$2m$	0	$2 \cdot log(n) + 1$
Input type	plain-cipher	cipher-cipher, plain-cipher	cipher-cipher, plain-cipher
Input form	many-to-one	many-to-many	many-to-many
batch num.	$N/2^m$	N	N/n

In summary, We present all the properties of TECMP, CDCMP, and RDCMP in the Table 4, and the packaging strategy as shown in the Figure 4.

5 Private Decision Tree Evaluation

This section provides a detailed description of the security model and propose ours PDTE schemes.

5.1 Security Model

In client-server model, which the client serves as the data party, holding the attribute vector, and the server serves as the model party and evaluator, holding the decision tree and performing the evaluation process.

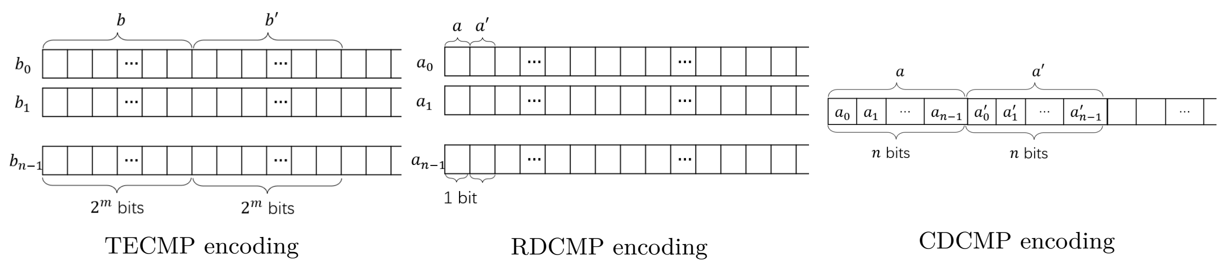

Fig. 4. Visual comparisons of original models.

The security goal is that the client does not have any decision tree private parameters except the server's classification results, and the server cannot learn any important information. The communication process involves the client initiating communication with the server, the server receiving the client's require and responding, then the client receiving the server's response. There is one round of necessary questioning and answering, without any intermediate interaction.

Similar to previous work, this work is based on the semi-honest adversary model, where the client and server execute the protocol honestly, cannot infer additional information based on the received information.

For the decision tree information that the client can inevitably obtain from the classification results, we consider this aspect beyond the scope of our discussion in this paper.

5.2 Batch-PDTE: Batch-CMP with SumPath

The scheme uses a LHE such as BFV. It is assumed that the public key, evaluation key, and relinearization key have been exchanged between client and server.

In Batch-PDTE, we integrate private comparison with the SumPath. The data is encoded for batching and the scheme mainly consists of three parts:

- For each internal node, privately compare the client attribute with the node's threshold to update the node's state value;
- Run SumPath to obtain a batch of classification results;
- Perform clear rows relation on the classification results.

In algorithm 10, our TECMP, RDCMP, and CDCMP are collectively referred to as BCMP.

Algorithm 10 Batch-PDTE

```
1: Procedure : $BPDTE(x, M)$
2: $(L, a, t, v) \leftarrow M$;
3: for all $d \in D$ do
4:     $c \leftarrow BCMP(x[a[d]], t[d])$;
5:     $d.left \leftarrow c$;
6:     $d.right \leftarrow 1 - c$;
7: end for
8: for all $l \in L$ do
9:     $s[l]$ = Sum of edges from root to leaf $L$;
10: end for
11: for all $l \in L$ do
12:     $r_x, r_y \leftarrow_r Z_p^N$;
13:     $x[l] \leftarrow r_x \odot s[l]$;
14:     $y[l] \leftarrow r_y \odot s[l] + v[l]$;
15: end for
16: return $\{x, y\}$.
```

There are L ciphertexts in x, and the classification result is recorded at the same position of the plaintext corresponding to y. In x, there is only one zero in the same position, and the rest are all random numbers.

In y, after decrypting x and y, return classification result in y based on the position of 0 in x. For example, when $L = 5$, there are 3 rows of data to be evaluated simultaneously. The plaintext in x, y is as shown below.

$$L = \{L_0, L_1, L_2, L_3, L_4\} \tag{43}$$

$$x = \{x_0, x_1, x_2, x_3, x_4\} \tag{44}$$

$$= \begin{bmatrix} r_x00 \\ r_x10 \\ r_x20 \end{bmatrix}, \begin{bmatrix} r_x01 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ r_x12 \\ r_x22 \end{bmatrix}, \begin{bmatrix} r_x03 \\ r_x13 \\ r_x23 \end{bmatrix}, \begin{bmatrix} r_x04 \\ r_x14 \\ r_x24 \end{bmatrix} \tag{45}$$

$$y = \{y_0, y_1, y_2, y_3, y_4\} \tag{46}$$

$$= \begin{bmatrix} r_y00 \\ r_y10 \\ r_y20 \end{bmatrix}, \begin{bmatrix} r_y01 \\ L_1 \\ L_1 \end{bmatrix}, \begin{bmatrix} L_2 \\ r_y12 \\ r_y22 \end{bmatrix}, \begin{bmatrix} r_y03 \\ r_y13 \\ r_y23 \end{bmatrix}, \begin{bmatrix} r_y04 \\ r_y14 \\ r_y24 \end{bmatrix} \tag{47}$$

Algorithm 11 Clear Rows Relation

```
1: **Procedure :** $CRR(len, x, y, L, N)$
2: $P_L$ = Random permutation in $L$;
3: **for all** $l \in L$ **do**
4:     $x\prime[l] = x[P_L[l]]$;
5:     $y\prime[l] = y[P_L[l]]$;
6: **end for**
7: **for all** $i \in [log(len)]$ **do**
8:     $P_{len}$ = Random permutation in $len$;
9:     $W_0, W_1 \leftarrow 0^N$
10:    **for all** $j \in [len/2]$ **do**
11:        $W_0[P_{len}[j]] = 1$;
12:        $W_1[P_{len}[len/2 + j]] = 1$;
13:    **end for**
14:    $P'_L$ = Random permutation in $L$;
15:    **for all** $l \in L$ **do**
16:        $x'_0[l] = x'[l] \odot W_0$;
17:        $x'_1[l] = x'[l] \odot W_1$;
18:        $y'_0[l] = y'[l] \odot W_0$;
19:        $y'_1[l] = y'[l] \odot W_1$;
20:    **end for**
21:    **for all** $l \in L$ **do**
22:        $x'[l] = x'_0[P'_L[l]] \oplus x'_1[l]$;
23:        $y'[l] = y'_0[P'_L[l]] \oplus y'_1[l]$;
24:    **end for**
25: **end for**
26: **return** $\{x, y\}$.
```

Note that the classification results L_1 of multiple rows are all present in the plaintext of y_1. When the client decrypts y_1, two risks arise: knowing the classification results L_1 and the corresponding positions of decision tree leaves, and knowing that different rows of data correspond to the same decision tree leaf. This violates our intention of only knowing the classification results.

Therefore, we added a clear rows relation (CRR) in Algorithm 11 to obfuscate the position of the classification results. We expect that the probability that a semi-honest adversary determines that two classification results are at the same position does not exceed $1/len$, where len represents the number of rows in client data. In detail, since these L ciphertexts record the classification results of len rows of data, we first perform a random permutation P_L to obscure the original positions. Then, we randomly split and merge the each

Algorithm 12 Batch PDTE with CRR

```
1: Procedure : $PDTE(x, M)$
2: $\{L\} \leftarrow M$;
3: $len \leftarrow Len(x)$;
4: $\{x, y\} \leftarrow BPDTE(x, M)$;
5: $\{x', y'\} \leftarrow CRR(len, x, y, L, N)$;
6: return $\{x', y'\}$.
```

x_i, y_i. This process is repeated $log(len)$ times. For the adversary, each separation and subsequent merging has a probability of 1/2 to confuse the positions. So, in the end, there is a probability of $1/len$ that distinguish the relationship between the classification results and the leaf nodes.

6 Run Time and Communication Cost

In this section, we present two benchmark. In Table 6 we evaluate and benchmark the private comparisons, and in Table 7 we evaluate the PDTE schemes on the real UCI dataset, which we measure at different bit precision. The source code[3] has been made public on Github.

6.1 Run Time in Private Comparsion

The device is 13th Gen Intel(R) Core(TM) i7-13700KF. Comparing TECMP, RDCMP, CDCMP with RCC, Folklore, XXCMP, they all operated under SEAL[4] library.

Table 5. The trees trained on the UCI dataset [17]

UCI	Data		Tree		
name	num	bit	node	internal	leaf
Heart	13	11	9	4	5
Breast	30	11	33	16	17
Spam	57	11	116	57	59
Electricity	8	10	1107	553	554

Table 6. Benchmark of private comparsion, amortized numbers, amortized time (ms), amortized communication (KB)

	XXCMP [2]			RCC [2]			FOLKLORE [2]			TECMP			RDCMP			CDCMP		
Bit	num	time	comm.	num	time	comm.	num	time	comm.	num	time	comm.	num	time	comm.	num	time	comm.
8	1	5.3	177	480	0.2	10.8	963	0.5	2.6	2048	0.04	1.1	16383	0.03	0.7	2048	0.1	1.3
12	1	5.5	177	326	0.6	28	655	1.2	3.7	1024	0.1	2.1	16383	0.07	1.4	1024	0.4	2.6
16	1	57	1297	248	1.1	66	496	2.2	5.0	2048	0.1	1.9	16383	0.07	1.4	1024	0.4	2.6
32	1	109	1729	252	6.8	197	252	9.6	9.8	4096	0.3	5.6	16383	0.16	2.7	512	0.9	5.3
36	1	109	1729	224	8.3	252	224	12.1	11.0	2048	0.7	11.1	16383	0.3	5.3	256	2.1	10.5
64				126	27.2	736	127	41.3	19.4	4096	0.6	10.8	16383	0.3	5.3	256	2.1	10.5
128				62	121	4815	63	180	39	4096	1.4	21.3	16383	0.6	12.4	128	5.1	24.5
256				30	1944	147534	31	785	79.5	4096	2.8	42.3	16383	1.2	24.6	64	11.5	49.2
512							15	3453	164	4096	5.8	98.6	16383	2.5	49.2	32	26.0	98.3
1024							7	15721	352	4096	11.5	197	16383	7.0	114	16	83.6	228

3 https://github.com/LoCCS-ViewSources/Batch-PDTE
4 https://github.com/microsoft/SEAL

In Table 6, we compare bit precision from 8 to 1024 bits. due to the large batch size, RDCMP exhibits the lowest amortized cost, the next are TECMP and CDCMP, and the CDCMP total communication cost are constant. At 32-bit, TECMP, RDCMP, and CDCMP are respectively 22, 42, and 7 times faster than the RCC algorithm in amortized runtime, and an order of magnitude lower in the amortized communication costs. Additionally, TECMP achieves a bit precision up to 26624 bits under a multiplication depth of 12 in our experiments.

6.2 Run Time in PDTE

In this subsection, we compare the evaluation time of our Batch-PDTE and the communication costs. We first benchmark on the decision tree trained on the UCI dataset [17], and then show the comparison of our experiments with sortinghat and level up.

In Table 5, the parameters of the tree mainly consists of the following parts: the bit precision of the data, the length of the client's attribute vector, and the number of nodes in the decision tree, including both internal and leaf nodes.

In Table 7, we present the PDTE based on the three private comparisons, considering whether to include the CRR algorithm. Our PDTE scheme is evaluated across four trees in Table 5. Incorporating the CRR algorithm requires a certain multiplication depth, leading to a reduction in the row of data evaluated at one time and consequently increasing the amortized costs. Without CRR, the RDCMP-PDTE exhibits the lowest amortized time and communication cost, requiring only 40ms and 100KB in the electricity dataset. With the CRR algorithm, TECMP-PDTE-CRR achieves the lowest amortized time and cost, at around 210ms and 1MB.

Table 7. Benchmark of PDTE, amortized numbers, amortized time (ms), amortized communication (KB)

	TECMP-PDTE			RDCMP-PDTE			CDCMP-PDTE		
Name	num	time	comm.	num	time	comm.	num	time	comm.
heart	1024	0.4	26	16383	0.3	17	1024	1.6	21
breast	1024	1.8	65	16383	1.4	42	1024	6.3	84
spam	1024	6.8	145	16383	5.4	84	1024	22.8	227
electricity	1024	52.2	481	16383	40.7	100	1024	209	1465
	TECMP-PDTE-CRR			RDCMP-PDTE-CRR			CDCMP-PDTE-CRR		
Name	num	time	comm.	num	time	comm.	num	time	comm.
heart	2048	1.8	55	512	16.7	777	256	13.1	164
breast	2048	6.6	137	512	68.1	1833	256	50.3	456
spam	1024	23.4	306	512	268	3666	256	179	1234
electricity	1024	210	1016	512	2095	4408	256	1683	7961

Table 8. Soringhat [14] and Level up [2] in the single row, time (ms), communication(KB)

	Sortinghat		XXCMP-PDTE		FOLKLORE-PDTE		RCC-PDTE	
Name	time	comm.	time	comm.	time	comm.	time	comm.
heart	38.36	8096	9	780	928	1568	200	7454
breast	153.8	18568	47	1570	1560	1568	405	7454
spam	543.1	35200	200	2824	3986	1568	1378	7454
electricity	5232	5016	2945	548	36791	1568	12225	6589

In Table 8, we ran the Sortinghat and Level up PDTE schemes on the same machines. In electricity, XXCMP-PDTE has the lowest communication and time cost, at 2.945s and 548KB, followed by Sortinghat and RCC-PDTE. It is worth noting that Table 5 is all around 11 bits, and the bit precision is relatively low.

Based on the data from both Table 7 and Table 8, we can draw the following conclusions: TECMP-PDTE and RDCMP-PDTE are 56 and 73 times more efficient than XXCMP-PDTE in the electricity tree, respectively. After adding CRR, the per-row amortized time of TECMP-PDTE-CRR is 14 times than the XXCMP-PDTE in the same dataset.

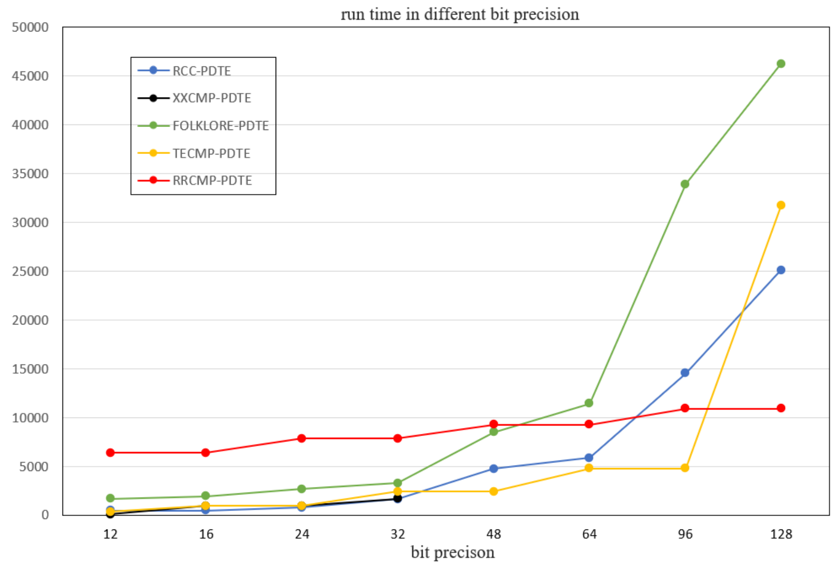

Fig. 5. High precision in breast data with single row

Due to our data being compared with amortized costs at relatively low precision, in Figure 5, we evaluate these PDTE at high precision with a single-row data in breast tree. The RRCMP-PDTE exhibits stability and is insensitive to changes in precision, performing faster than others after 128 bits. TECMP-PDTE has the lowest time consumption at around 64 bits. In terms of low precision comparison, XXCMP-PDTE, RCC-PDTE, and TECMP-PDTE show similar performance.

6.3 Summary of the Result

Among all private comparisons, our private comparison demonstrates superior performance in amortization, enabling comparisons with arbitrary precision and in arbitrary forms, while the compared results maintain the output expressive.

With the support of CRR, by obfuscating the classification results and leaf positions, we further improve the security of batch-processed PDTE, but also sacrifice some computational efficiency and multiplication depth. However, due to the speed improvement of our private comparison method, the Batch-PDTE is still very efficient, about 14 times faster than the state-of art. In terms of communication cost, they are greatly reduced due to the adoption of packaging strategies. Even in a single row, our Batch-PDTE performs well over the 64-bit.

7 Disscussion and Future Work

Next, we will expand the existing work from the following aspects.

Include the different application scenarios: the data party, model party, and third party, the encryptor, decryptor, and evaluator are all independent, so we can use a client-cloud-server architecture, or integrate the model party with the encryptor and decryptor.

Expanding private comparison: private comparison as a fundamental module, has various applications such as selecting maximum values, sorting among multiple data parties, and simulating the activation function in neural networks.

Improved security: We are based on semi-honest assumptions and do not pay attention to malicious adversary. Ensuring that evaluators honest to prescribed evaluation scheme can effectively enhance the security of decision trees. Additionally, decision tree models inherently possess some privacy leakage problems, making it challenging to obfuscate decision results and tree parameters.

8 Conclusion

In this work, we propose two types of batch private comparison and their PDTE schemes. These private comparison are non-interactive, batchable, and amortized efficient. they can achieve high-precision comparison at low multiplication depth, and is suitable for cipher-cipher, cipher-plain, one-to-many and many-to-many input type. Meanwile, The security of the Batch-PDTE is further improved, and the SumPath is improved to confuse the relationship between the leaf node position and the classification results.

We also compared the computation and communication cost of several private comparison protocols at various bit precision. RDCMP's amortization time is the fastest, and CDCMP's communication cost is low. TECMP is relatively moderate, with great computing speed and communication cost. Compared with XXCMP, RCC and FOLKLORE, there is an order of magnitude advantage, and the comparison bit precision is higher.

PDTE benefits from the computational speed and communication efficiency of the private comparison algorithm, resulting a very fast amortized computation speed and low communication costs. Compared to the state-of-art, TECMP-PDTE shows an improvement of approximately 14 times in computing speed and lower communication costs. Meanwhile, RDCMP-PDTE and CDCMP-PDTE are affected by the multiplication depth in SumPath with CRR and do not fully utilize the batch processing power. In scenarios with lower security requirements, such as CRR-free, it will exhibit more robust performance.

Acknowledgments. Thanks to the support from the Viewsouces company and the teachers from Shanghai Jiao Tong University.

References

1. Akavia, A., Leibovich, M., Resheff, Y.S., Ron, R., Shahar, M., Vald, M.: Privacy-preserving decision trees training and prediction. ACM Transactions on Privacy and Security **25**(3), 1–30 (May 2022). https://doi.org/10.1145/3517197
2. Akhavan Mahdavi, R., Ni, H., Linkov, D., Kerschbaum, F.: 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. pp. 2945–2958. CCS '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3576915.3623095
3. Angel, S., Chen, H., Laine, K., Setty, S.: Pir with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 962–979. SP '18, Institute of Electrical and Electronics Engineers, San Francisco, CA, USA (2018). https://doi.org/10.1109/SP.2018.00062
4. Azar, A.T., El-Metwally, S.M.: Decision tree classifiers for automated medical diagnosis. Neural Computing and Applications **23**(7), 1433–3058 (Dec 2013). https://doi.org/10.1007/s00521-012-1196-7
5. Azogagh, S., Delfour, V., Gambs, S., Killijian, M.O.: Probonite: Private one-branch-only non-interactive decision tree evaluation. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 23–33. WAHC'22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3560827.3563377
6. Bai, J., Song, X., Cui, S., Chang, E.C., Russello, G.: Scalable private decision tree evaluation with sublinear communication. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. pp. 843–857. ASIA CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3488932.3517413
7. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. Cryptology ePrint Archive **2014**(331), 1–34 (Jan 2015). https://doi.org/10.14722/NDSS.2015.23241
8. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. pp. 498–507. CCS '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1315245.1315307
9. Buckman, J., Roy, A., Raffel, C., Goodfellow, I.: Thermometer encoding: One hot way to resist adversarial examples. In: International Conference on Learning Representations. ICLR '18 (2018)

10. Chen, H., Chillotti, I., Ren, L.: Onion ring oram: Efficient constant bandwidth oblivious ram from (leveled) tfhe. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 345–360. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3354226
11. Cheon, J.H., Kim, M., Kim, M.: Search-and-compute on encrypted data. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) Financial Cryptography and Data Security. FC '15, vol. 8976, pp. 142–159. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_11
12. Cheon, J.H., Kim, M., Lauter, K.: Homomorphic computation of edit distance. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) Financial Cryptography and Data Security. FC '15, vol. 8976, pp. 194–212. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_15
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**, 57 (2020). https://doi.org/https://doi.org/10.1007/s00145-019-09319-x
14. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 563–577. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560702
15. Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for on-line auctions. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) Information Security and Privacy. ACISP '07, vol. 4586, pp. 416–430. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73458-1_30
16. Damgard, I., Geisler, M., Kroigard, M.: A correction to 'efficient and secure comparison for on-line auctions'. Int. J. Appl. Cryptol. **1**(4), 323–324 (2009). https://doi.org/10.1504/IJACT.2009.028031
17. Dua, D., Graff, C., et al.: Uci machine learning repository (2017)
18. Gentry, C., Halevi, S., Jutla, C., Raykova, M.: Private database access with he-over-oram architecture. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) Applied Cryptography and Network Security. ACNS '15, vol. 9092, pp. 172–191. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_9
19. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 669–684. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516668
20. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology. ASIACRYPT '21, vol. 13092, pp. 608–639. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-92078-4_21
21. Kiss, Á., Naderpour, M., Liu, J., Asokan, N., Schneider, T.: Sok: Modular and efficient private decision tree evaluation. Proceedings on Privacy Enhancing Technologies **2019**(2), 187–208 (Dec 2019). https://doi.org/10.2478/popets-2019-0026
22. Lin, H.Y., Tzeng, W.G.: An efficient solution to the millionaires' problem based on homomorphic encryption. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) Applied Cryptography and Network Security. ACNS '05, vol. 3531, pp. 456–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11496137_31
23. Liu, K., Tang, C., et al.: Secure two-party decision tree classification based on function secret sharing. Complexity **2023**, 1–13 (Sep 2023). https://doi.org/10.1155/2023/5302915
24. jie Lu, W., Huang, Z., Zhang, Q., Wang, Y., Hong, C.: Squirrel: A scalable secure Two-Party computation framework for training gradient boosting decision tree. In: 32nd USENIX Security Symposium. pp. 6435–6451. USENIX Security '23, USENIX Association, Anaheim, CA (2023)
25. Lu, W.j., Zhou, J.j., Sakuma, J.: Non-interactive and output expressive private comparison from homomorphic encryption. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. pp. 67–74. ASIACCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3196494.3196503
26. Mahdavi, R.A., Kerschbaum, F.: Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In: 31st USENIX Security Symposium. pp. 1723–1740. USENIX Security '22, USENIX Association, Boston, MA (2022)
27. Nateghizad, M., Erkin, Z., Lagendijk, R.L.: An efficient privacy-preserving comparison protocol in smart metering systems. EURASIP Journal on Information Security **2016**(11), 1–8 (May 2016). https://doi.org/10.1186/s13635-016-0033-4
28. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 506–519. ASIA CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3052973.3053009

29. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers - a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **35**(4), 476–487 (2005). https://doi.org/10.1109/TSMCC.2004.843247
30. Shi, E., Bethencourt, J., Chan, T.H.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: 2007 IEEE Symposium on Security and Privacy (SP '07). pp. 350–364. SP '07, Institute of Electrical and Electronics Engineers, Berkeley, CA, USA (2007). https://doi.org/10.1109/SP.2007.29
31. Singh, A., Guttag, J.V.: A comparison of non-symmetric entropy-based classification trees and support vector machine for cardiovascular risk stratification. In: 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. pp. 79–82. IEEE, Boston, MA, USA (2011). https://doi.org/10.1109/IEMBS.2011.6089901
32. Tai, R.K.H., Ma, J.P.K., Zhao, Y., Chow, S.S.M.: Privacy-preserving decision trees evaluation via linear functions. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) Computer Security – ESORICS 2017. ESORICS '17, vol. 10493, pp. 494–512. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_27
33. Tueno, A., Boev, Y., Kerschbaum, F.: Non-interactive private decision tree evaluation. In: Singhal, A., Vaidya, J. (eds.) Data and Applications Security and Privacy XXXIV. DBsec '20, vol. 12122, pp. 174–194. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-49669-2_10
34. Tueno, A., Kerschbaum, F., Katzenbeisser, S.: Private evaluation of decision trees using sublinear cost. Proceedings on Privacy Enhancing Technologies **2019**(1), 266–286 (Sep 2019). https://doi.org/10.2478/popets-2019-0015
35. Veugen, T.: Improving the dgk comparison protocol. In: 2012 IEEE International Workshop on Information Forensics and Security (WIFS). pp. 49–54. IEEE, Costa Adeje, Spain (2012). https://doi.org/10.1109/WIFS.2012.6412624
36. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2011)
37. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. Proceedings on Privacy Enhancing Technologies **2016**(4), 335–355 (Feb 2016). https://doi.org/10.1515/popets-2016-0043
38. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). pp. 160–164. Chicago, IL, USA (1982). https://doi.org/10.1109/SFCS.1982.38