On the Gini-impurity Preservation For Privacy Random Forests

Anonymous Author(s)
Affiliation
Address
email

Abstract

Random forests have been one successful ensemble algorithms in machine learning. Various techniques have been utilized to preserve the privacy of random forests from anonymization, differential privacy, homomorphic encryption, etc., whereas it rarely takes into account some crucial ingredients of learning algorithm. This work presents a new encryption to preserve data's Gini impurity, which plays a crucial role during the construction of random forests. Our basic idea is to modify the structure of binary search tree to store several examples in each node, and encrypt data features by incorporating label and order information. Theoretically, we prove that our scheme preserves the minimum Gini impurity in ciphertexts without decrypting, and present the security guarantee for encryption. For random forests, we encrypt data features based on our Gini-impurity-preserving scheme, and take the homomorphic encryption scheme CKKS to encrypt data labels due to their importance and privacy. We conduct extensive experiments to show the effectiveness, efficiency and security of our proposed method.

1 Introduction

From the pioneer work [1], random forests have been one successful ensemble algorithms [2–4], with diverse applications such as ecology [5], computational biology [6], objection recognition [7], remote sensing [8], computer vision [9], etc. The basic idea is to construct a large number of random trees individually and make prediction based on an average of their predictions. Numerous variants of random forests have been developed to improve performance under different settings [10–21], as well as theoretical understandings on the success of random forests [21–26]. The splitting criterion, such as Gini impurity and information gain, has been one of the most important ingredients during the construction of random forests [1, 27].

Various techniques have been adopted to preserve the privacy of random forests, particularly for sensitive tasks such as medical diagnosis, financial predictions, and so on. For example, differential privacy [28] has been successfully applied to preserve the privacy of random forests [29, 30] and decision trees [31–33], by adding certain noise perturbations. Another relevant approach is the secure multi-party computation for random forests and decision tree [34–38], where the privacy is preserved by multi-party joint computation over respective data inputs without leakage.

Homomorphic encryption [39–42] has been one of the most important cryptosystems in privacy-preserving computing [43–46]. Based on such scheme, various algorithms have been developed to train privacy random forests and decision trees [47–51], while there are still some methods only focusing on inference without training due to computational costs [52–57]. Homomorphic encryption has also been used for regression problem [58, 59], neural network [60–64], collaborative filtering [65], and so on. In addition, LeFevre et al. [66] took the anonymization [67] for random forests by grouping similar attributes so as to hardly identify specific individual information.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Table 1: Comparisons of communications and complexities for different privacy-preserving decision trees. Here, n denotes the number of examples in training dataset, and τ is the cardinality of label space. Let h and κ denote the height and number of leaves of decision tree $(h < \kappa)$, respectively. We also denote by $\bar{\jmath}$ the average number of possible splitting features and positions in the construction of decision trees, and p denotes the number of clients for secure multi-party computation.

Scheme	Training communication		Training comp. complexity		Predictive communication		Predictive comp. complexity		Privacy of model
	Rounds	Bandwidth	Client	Server	Rounds	Bandwidth	Client	Server	
SMCDT [68]	$O(\kappa)$	$O(\bar{\jmath}\tau n)$	$O(\kappa\bar{\jmath}\tau n)$	$O(\kappa\bar{\jmath}\tau)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✗
PPID3 [35]	$O(\kappa)$	$O(p^2\bar{\jmath}\tau n)$	$O(\kappa p^2\bar{\jmath}\tau n)$	$O(\kappa p^2\bar{\jmath}\tau n)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✗
SID3 [36]	$O(hp)$	$O(\kappa\bar{\jmath}\tau)$	$O(\kappa\bar{\jmath}\tau n)$	$O(\kappa\bar{\jmath}\tau)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✗
OPPC4.5 [38]	$O(\kappa p)$	$O(p\bar{\jmath}\tau)$	$O(\kappa\bar{\jmath}\tau(n+p))$	$O(m\bar{\jmath}\tau p)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✗
PivotRFs [69]	$O(\kappa p)$	$O(\bar{\jmath}\tau + \tau n)$	$O(\kappa\bar{\jmath}\tau n)$	$O(\kappa\bar{\jmath}\tau)$	$O(p)$	$O(\kappa)$	$O(\kappa)$	$O(\kappa)$	✓
MulPRFs [70]	$O(h)$	$O(\log n + \log d)$	$O(hdn \log n)$	$O(hdn \log n)$	$O(h)$	$O(1)$	$O(h)$	$O(h)$	✗
PPD-ERTs [71]	$O(hp)$	$O(\kappa\bar{\jmath}\tau)$	$O(\kappa\bar{\jmath}\tau n)$	$O(\kappa\bar{\jmath}\tau)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✗
HEldpRFs [50]	$O(h)$	$O(\kappa\bar{\jmath}\tau)$	$O(\kappa\bar{\jmath}\tau)$	$O(\kappa\bar{\jmath}\tau n)$	$O(1)$	$O(\kappa)$	$O(\kappa)$	$O(\kappa)$	✓
Our work	$O(h)$	$O(\kappa\bar{\jmath})$	$O(\kappa)$	$O(\kappa\bar{\jmath}\tau n)$	$O(1)$	$O(1)$	$O(1)$	$O(h)$	✓

This work takes one step on data encryption from some crucial ingredients of learning algorithm, and main contributions can be summarized as follows:

- We present a new encryption to preserve data's Gini impurity, and the basic idea is to modify the structure of binary search trees to maintain several samples on each node, and encrypt data's features by incorporating label and order information. Our scheme could change the data frequencies, which is also beneficial for data security.
- Theoretically, we prove the preservation of minimum Gini impurity in ciphertexts without decryption, which plays an important role on the construction of random forests. Our scheme also satisfies the security against Gini-impurity-preserving chosen plaintext attack.
- We present privacy-preserving training and predicting for random forests in popular client-server protocol. We take our Gini-impurity-preserving encryption for data's features, and adopt the homomorphic encryption CKKS to encrypt data's labels. Our encrypted decision tree takes smaller communication and computational complexities, as shown in Table 1.
- Extensive experiments show that our encrypted random forests take significantly better performance than prior privacy random forests via encryption, anonymization and differential privacy, and are comparable to original (plaintexts) random forests without encryption. Our encrypted random forests make a good balance between computational cost and data security.

2 Preliminaries

Let $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} = \{1, 2, \cdots, \tau\}$ $(\tau \geq 2)$ denote the feature and label space, respectively. Let $\mathcal{D}$ be an underlying distribution over the producted space $\mathcal{X} \times \mathcal{Y}$. Note that distribution $\mathcal{D}$ is unknown in practice, and what we can observe is a training data $S_n = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, where each element is drawn i.i.d. from $\mathcal{D}$. Let $|A|$ be the cardinality of set A, and $[\![\cdot]\!]$ denotes the corresponding encrypted value. Let $\mathcal{N}(\mu, \sigma^2)$ be a normal distribution of mean μ and variance σ^2.

Homomorphic Encryption (HE) is a cryptosystem, which allows operations on encrypted data without access to a secret key [39]. Given encryption function $E(\cdot)$ and decryption function $D(\cdot)$, HE scheme provides two operators $\oplus$ and $\otimes$ such that, for every pair of plaintexts x_1 and x_2,

$$D\left(E(x_1) \oplus E(x_2)\right) = x_1 + x_2 \quad \text{and} \quad D\left(E(x_1) \otimes E(x_2)\right) = x_1 \times x_2 \ ,$$

where $+$ and $\times$ denote standard addition and multiplication operations, respectively.

3 An Encryption for Gini Impurity

This section presents the first encryption to preserve the minimum Gini impurity over encrypted data without decryption. For simplicity, we give the detailed encryption on one-dimensional feature by incorporating label information, and could make similar considerations for other dimensions.

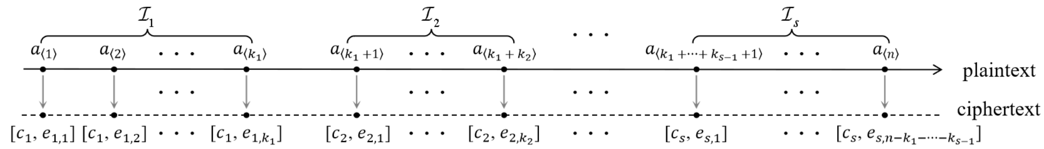

Figure 1: A simple illustration for our encryption: each plaintext is encrypted into a ciphertext vector $[c_i, e_{i,j}]$. Here, random numbers $c_1 < c_2 < \cdots < c_s$ are introduced to preserve the Gini impurity for random forests, and we take homomorphic encryption scheme for $e_{i,j} = \text{Enc}(k_{\text{pub}}, j)$ in Eqn. (5), which is helpful for decryption.

3.1 Theoretical Analysis for Gini Impurity

Let $A = \{(a_1, y_1), \cdots, (a_n, y_n)\}$ be a dataset with labels $y_i \in [\tau]$, and define the Gini value as

$$\text{Gini}(A) = 1 - \sum\nolimits_{y\in[\tau]} p_y^2 \,,$$

where p_y denotes the proportion of the label y. Let $A_a^l = \{(a_i, y_i)\colon a_i \leq a, (a_i, y_i) \in A\}$ and $A_a^r = \{(a_i, y_i)\colon a_i > a, (a_i, y_i) \in A\}$ be the left and right subsets of A w.r.t. a splitting point a, respectively. We define the Gini impurity w.r.t. dataset A and splitting point a as

$$I_G(A, a) = w_l \cdot \text{Gini}(A_a^l) + w_r \cdot \text{Gini}(A_a^r) \,, \tag{1}$$

where $w_l = |A_a^l|/n$ and $w_r = |A_a^r|/n$. Let $I_G^*(A)$ be the minimum Gini impurity of dataset A, i.e.,

$$I_G^*(A) = \min\nolimits_{a\in\mathbb{R}}\{I_G(A, a)\} \,. \tag{2}$$

The minimum Gini impurity plays a crucial role on nodes splitting during the construction of random forests. We could re-sort dataset A with a non-decreasing order for $a_1, a_2, \cdots, a_n$ as follows:

$$A = \left\{(a_{\langle 1\rangle}, y_{\langle 1\rangle}), (a_{\langle 2\rangle}, y_{\langle 2\rangle}), \cdots, (a_{\langle n\rangle}, y_{\langle n\rangle})\right\} \,, \tag{3}$$

where $a_{\langle 1\rangle} \leq a_{\langle 2\rangle} \leq \cdots \leq a_{\langle n\rangle}$, and $y_{\langle 1\rangle}, y_{\langle 2\rangle}, \cdots, y_{\langle n\rangle}$ denote their corresponding labels. By incorporating label information, we partition dataset A into several datasets $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_s$ as follows:

$$\begin{aligned} \mathcal{I}_1 &= \left\{(a_{\langle 1\rangle}, y_{\langle 1\rangle}), \cdots, (a_{\langle k_1\rangle}, y_{\langle k_1\rangle})\right\} \,, \\ \mathcal{I}_2 &= \left\{(a_{\langle k_1+1\rangle}, y_{\langle k_1+1\rangle}), , \cdots, (a_{\langle k_1+k_2\rangle}, y_{\langle k_1+k_2\rangle})\right\} \,, \\ &\cdots \\ \mathcal{I}_s &= \left\{(a_{\langle k_1+k_2+\cdots+k_{s-1}+1\rangle}, y_{\langle k_1+k_2+\cdots+k_{s-1}+1\rangle}), \cdots, (a_{\langle n\rangle}, y_{\langle n\rangle})\right\} \,. \end{aligned} \tag{4}$$

Here, any two adjacent datasets have different labels, and all samples have an identical label in one dataset $\mathcal{I}_j$, i.e., $y_{\langle i\rangle} = y_{\langle i'\rangle}$ for every $(a_{\langle i\rangle}, y_{\langle i\rangle}) \in \mathcal{I}_j$ and $(a_{\langle i'\rangle}, y_{\langle i'\rangle}) \in \mathcal{I}_j$.

We consider two important factors in encryption: i) preservation of the minimum Gini impurity $I_G^*(A)$ over the encrypted data, and ii) a cryptosystem for encoding and decoding data. Based on such recognition, we introduce the following encryption, for every example $(a_{\langle i\rangle}, y_{\langle i\rangle}) \in \mathcal{I}_j$,

$$[\![a_{\langle i\rangle}]\!] = ([\![a_{\langle i\rangle}]\!]_1, [\![a_{\langle i\rangle}]\!]_2) = \begin{cases} (c_1, \text{Enc}(k_{\text{pub}}, i)) & \text{for } \; j = 1 \,, \\ (c_j, \text{Enc}(k_{\text{pub}}, i - k_1 - \cdots - k_{j-1})) & \text{for } \; 2 \leq j \leq s \,. \end{cases} \tag{5}$$

Here, $c_1, c_2, \cdots, c_s$ are random numbers s.t. $c_1 < c_2 < \cdots < c_s$, which aim to preserve the minimum Gini impurity. We take the homomorphic encryption scheme CKKS with a public key k_{pub} for $[\![a_{\langle i\rangle}]\!]_2 = \text{Enc}(k_{\text{pub}}, i - k_1 - \cdots - k_{j-1})$ in Eqn. (5), and it is helpful for decryption. Figure 1 presents a simple illustration for our encryption, and the detailed decryption is given in Appendix A.

We now present our main theorem as follows:

Theorem 1. *We have $I_G^*(A) = I_G^*(A')$, for re-sort dataset A by Eqn. (3) and for the corresponding encrypted dataset $A' = \{([\![a_{\langle 1\rangle}]\!]_1, y_{\langle 1\rangle}), \cdots, ([\![a_{\langle n\rangle}]\!]_1, y_{\langle n\rangle})\}$ from Eqns. (4)-(5).*

This theorem shows that our encryption could preserve the minimum Gini impurity over encrypted data. The detailed proof is presented in Appendix B, which involves the proof of piecewise monotonicity of $I_G(A, a)$ w.r.t. splitting point a, and then solves the minimum splitting point on plaintexts, as well as the corresponding point on encrypted data.

Algorithm 1 The Gini-impurity-preserving encryption

Input: Dataset $A = \{(a_1, y_1), \cdots, (a_n, y_n)\}$
Output: Binary search tree $\mathcal{BT}$, ciphertexts $\{[\![a_1]\!], \cdots, [\![a_n]\!]\}$
Initialize: Tree $\mathcal{BT} = \emptyset$ with its $\mathit{cipher}_1 = c_{\text{max}}/2$, where c_{max} is a random number with $c_{\text{max}} > n$

for $i = 1, \cdots, n$ **do**
  Set $t =$ root of $\mathcal{BT}$, $t_{\text{min}} = 0, t_{\text{max}} = c_{\text{max}}$ and index $= 1$ %% Step-I
  while t is an internal node **and** index==1 **do**
    index$= 0$
    if $t.\mathit{left} \neq \emptyset$ **and** $a_i < \max\{a_j : (a_j, y_j) \in t.\mathit{left}.\mathit{samples}\}$ **then**
      $t = t.\mathit{left}$, $t_{\text{max}} = t.\mathit{cipher}_1$, index $= 1$
    else if $t.\mathit{right} \neq \emptyset$ **and** $a_i > \min\{a_j : (a_j, y_j) \in t.\mathit{right}.\mathit{samples}\}$ **then**
      $t = t.\mathit{right}$, $t_{\text{min}} = t.\mathit{cipher}_1$, index $= 1$
    end if
  end while
  Update $t = t.\mathit{left}$ if Eqn. (6) is true, and update $t = t.\mathit{right}$ if Eqn. (7) is true
  if $y_i = y_j$ for every $(a_j, y_j) \in t.\mathit{samples}$ **then** %% Step-II
    Split node t by Algorithm 2 with inputs of (a_i, y_i) and the corresponding interval $[t_{\text{min}}, t_{\text{max}}]$
  end if
  Append example (a_i, y_i) into $t.\mathit{samples}$ and update $t.\mathit{cipher}_2 = \text{Enc}(k_{\text{pub}}, |t.\mathit{samples}|)$
  Encrypt $[\![a_i]\!] = (t.\mathit{cipher}_1, t.\mathit{cipher}_2)$
end for

3.2 Binary Search Tree for Encryption

This section presents new binary search tree to encrypt $a_1, \cdots, a_n$ dynamically, and it is helpful for un-ordered dataset $A = \{(a_1, y_1), \cdots, (a_n, y_n)\}$, or when example (a_i, y_i) arrives in a streaming data. We begin with an alternative structure for binary search tree to maintain several samples on a node from Eqns. (4)-(5), rather than previous only one sample [72, 73]. Our new structure is given by

$$\textbf{Struct Tree}\ \{\text{Plaintext } \mathit{samples};\quad \text{Ciphertext } \mathit{cipher}_1, \mathit{cipher}_2;\quad \text{Tree } \mathit{left}, \mathit{right}\}\ .$$

The *samples* stores one or multiple samples from A, and cipher_1 and cipher_2 are the first and second ciphertext in Eqn. (5), and *left* and *right* denote left and right child of the current node, respectively.

We initialize an empty tree $\mathcal{BT} = \emptyset$, and construct binary search tree iteratively. We maintain an interval $[t_{\text{min}}, t_{\text{max}}]$ in each iteration so as to keep the increasing order of ciphertexts $c_1, c_2, \cdots, c_s$ in Eqn. (5). During the i-th iteration, we receive a sample (a_i, y_i), and then take two steps as follows:

Step-I: Search a node for sample (a_i, y_i) in binary search tree $\mathcal{BT}$

Let t be a node pointer with the initialization of the root of $\mathcal{BT}$. We search a path downward in $\mathcal{BT}$ by comparing with a_i, and the search will terminate when t is a leaf node or an empty node.

For an internal node t, the search continues to its left child and updates $t_{\text{max}} = t.\mathit{cipher}_1$ if the left child $t.\mathit{left} \neq \emptyset$ and $a_i < \max\{a_j : (a_j, y_j) \in t.\mathit{left}.\mathit{samples}\}$; and the search continues to its right child and updates $t_{\text{min}} = t.\mathit{cipher}_1$ if the right child $t.\mathit{right} \neq \emptyset$ and $a_i > \min\{a_j : (a_j, y_j) \in t.\mathit{right}.\mathit{samples}\}$; otherwise, the search terminates. This procedure of iterative searches can be easily implemented with a while loop.

It is necessary to consider two special cases after the above search. We update $t = t.\mathit{left}$ if

$$t.\mathit{left} \neq \emptyset,\ a_i < \min\{a_j : (a_j, y_j) \in t.\mathit{samples}\} \text{ and } y_i = y_j \text{ for all } (a_j, y_j) \in t.\mathit{left}.\mathit{samples}. \quad (6)$$

In a similar manner, we update $t = t.\mathit{right}$ if

$$t.\mathit{right} \neq \emptyset,\ a_i > \max\{a_j : (a_j, y_j) \in t.\mathit{samples}\} \text{ and } y_i = y_j \text{ for all } (a_j, y_j) \in t.\mathit{right}.\mathit{samples}. \quad (7)$$

Step-II: Update the binary search tree $\mathcal{BT}$

After Step-I, we could find a node t for sample (a_i, y_i) and the corresponding interval $[t_{\text{min}}, t_{\text{max}}]$. We directly append the example (a_i, y_i) into $t.\mathit{samples}$ if $y_i = y_j$ for every $(a_j, y_j) \in t.\mathit{samples}$; otherwise, it is necessary to split the node t according to a_i.

Algorithm 2 Splitting a node for encryption

Input: Example (a_i, y_i), node t of binary search tree $\mathcal{BT}$, and interval $[t_{\min}, t_{\max}]$
Output: Updated node t

Initialize an empty node l with $l.samples = \{(a_j, y_j) \in t.samples\colon a_j < a_i\}$
if $l.samples \neq \emptyset$ **then**
  if $t.left \neq \emptyset$ **then**
    Set $l.cipher_1$ according to Eqn. (8), and update $l.left = t.left$, $t.left = l$
  else
    Set $l.cipher_1$ according to Eqn. (9), and update $t.left = l$
  end if
end if
Initialize an empty node r with $r.samples = \{(a_j, y_j) \in t.samples\colon a_j > a_i\}$
if $r.samples \neq \emptyset$ **then**
  if $t.right \neq \emptyset$ **then**
    Set $r.cipher_1$ according to Eqn. (10), and update $r.right = t.right$, $t.right = r$
  else
    Set $r.cipher_1$ according to Eqn. (11), and update $t.right = r$
  end if
end if
Update $t.samples = t.samples \setminus l.samples \setminus r.samples$

We initialize an empty node l with $l.samples = \{(a_j, y_j) \in t.samples\colon a_j < a_i\}$, and it is sufficient to consider $l.samples \neq \emptyset$. If $t.left \neq \emptyset$, then we set

$$l.cipher_1 = (t.left.cipher_1 + t.cipher_1)/2 + \xi \quad \text{s.t.} \quad t.left.cipher_1 < l.cipher_1 < t.cipher_1 \ , \tag{8}$$

and update $l.left = t.left$, $t.left = l$; otherwise, we set

$$l.cipher_1 = (t_{\min} + t.cipher_1)/2 + \xi \quad \text{s.t.} \quad l.cipher_1 \in (t_{\min}, t.cipher_1) \ , \tag{9}$$

and update $t.left = l$. Here, ξ is random number sampled from $\mathcal{N}(0, 1)$, and notice that we may randomly sample ξ multiple times so that the condition holds in Eqns (8)-(9), respectively.

We make similar update for the right child of node t: initialize an empty node r with $r.samples = \{(a_j, y_j) \in t.samples\colon a_j > a_i\}$, and consider $r.samples \neq \emptyset$. If $t.right \neq \emptyset$, then we set

$$r.cipher_1 = (t.cipher_1 + t.right.cipher_1)/2 + \xi \ \text{ s.t. } \ t.cipher_1 < r.cipher_1 < t.right.cipher_1 \ , \tag{10}$$

and update $r.right = t.right$, $t.right = r$; otherwise, we set

$$r.cipher_1 = (t.cipher_1 + t_{\max})/2 + \xi \quad \text{s.t.} \quad r.cipher_1 \in (t.cipher_1, t_{\max}) \ , \tag{11}$$

and update $t.right = r$. Algorithm 2 presents the detailed descriptions on the splitting of node t.

Algorithm 1 presents an overview of our Gini-impurity-preserving encryption, and the decryption is given in Appendix A. Our scheme does not only keep the minimum Gini impurity, but also change frequencies to prevent decryption from frequencies, which is beneficial for encryption as in [74]. Our scheme takes an average of $O(n \log n)$ computational complexity, since it requires $O(\log n)$ and $O(1)$ computational complexities to search and update a node in each iteration, respectively.

3.3 Security Analysis

For ciphertext vector $[\![a]\!] = ([\![a]\!]_1, [\![a]\!]_2)$ in Eqn. (5), it suffices to discuss the first ciphertext $[\![a]\!]_1$, since the security of $[\![a]\!]_2$ has been analyzed in homomorphic encryption CKKS [41]. Following semantic security against chosen plaintext attacks [73, 75], we define a security game $\text{Game}_{\text{GIPCPA}}$:

- An adversary chooses two sequences with distinct plaintexts $\{a_1^0, \cdots, a_n^0\}$ and $\{a_1^1, \cdots, a_n^1\}$, and sends them to a challenger;
- The challenger flips an unbiased coin $b \in \{0, 1\}$ to select $\{a_1^b, \cdots, a_n^b\}$, and randomly sets their corresponding labels $\{y_1^b, \cdots, y_n^b\}$ with each y_i^b drawn independently and uniformly over $[\tau]$. The challenger encrypts $\{a_1^b, \cdots, a_n^b\}$ by Eqns. (4) and (5), and sends the ciphertexts to the adversary;
- The adversary outputs a guess of b, i.e., which sequence is selected for encryption.

We then introduce the security against Gini-impurity-preserving chosen plaintext attack.

Algorithm 3 Finding the best splitting feature and position

Input: Encrypted datasets $[\![S_n^t]\!]$, available splitting feature and position $[\![s]\!]_{i=1}^{\jmath}$, and secret key k_{sec}
Output: index i^*

%% Server:
 for $i \in [\jmath]$ **do**
  Calculate Gini impurity $I_G([\![S_n^t]\!], [\![s]\!]_i)$ from Eqn. (12) w.r.t splitting feature and position $[\![s]\!]_i$
 end for
 Send ciphertexts $\{I_G([\![S_n^t]\!], [\![s]\!]_i)\}_{i\in[\jmath]}$ to the client

%% Client:
 Get the decrypted $\{\text{Dec}(k_{\text{sec}}, I_G([\![S_n^t]\!], [\![s]\!]_i))\}_{i\in[\jmath]}$
 Set $i^* = -1$ if $\text{Dec}(k_{\text{sec}}, I_G([\![S_n^t]\!], [\![s]\!]_i)) = 0$ for every $i \in [\jmath]$; otherwise, set i^* by Eqn. (13)
 Send i^* to the server

Definition 2. A scheme is said to be indistinguishable under Gini-impurity-preserving chosen plaintext attack if the probability of outputs with the correct guess b is negligible for the adversary $\mathcal{A}$ in $\text{Game}_{\text{GIPCPA}}$, that is, $\Pr[\mathcal{A}(\text{Game}_{\text{GIPCPA}}) = b] < 1/2{+}\text{small constant}$.

The following theorem shows that our encrypted plaintexts sequences are indistinguishable, and the detailed proof is presented in Appendix C.

Theorem 3. *Our scheme for the first ciphertexts $[\![a_1]\!]_1, [\![a_2]\!]_1, \cdots, [\![a_n]\!]_1$ in Section 3.2 is security against Gini-impurity-preserving chosen plaintext attack.*

4 Encrypted Random Forests

For encrypted random forests, we follow the popular client-server protocols [50, 72, 76, 77]. A client encrypts training and testing data, and transfers encrypted data to an honest-but-curious server. The server trains random forests from the encrypted data with the aid of client, and finally returns predictions on encrypted testing data.

Encryption for training and testing datasets

Recall training data $S_n = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$ with $\boldsymbol{x}_i = (x_{i,1}, \cdots, x_{i,d})$. The client constructs d binary search trees $\mathcal{BT}_1, \mathcal{BT}_2, \cdots, \mathcal{BT}_d$ according to Algorithm 1 over different dimensional features and labels in S_n, where $\mathcal{BT}_j$ is used to encrypt features $\{x_{1,j}, \cdots, x_{n,j}\}$ for $j \in [d]$.

We take the homomorphic encryption CKKS [41] to encrypt training labels $y_1, \cdots, y_n$. Each label y_i is encoded with a vector of size τ by one-hot method, and we encrypt the vector by homomorphic encryption scheme CKKS with a public key k_{pub}. The final ciphertexts $[\![y_i]\!] = [[\![y_{i,1}]\!], \cdots, [\![y_{i,\tau}]\!]]$ is given by $[\![y_{i,j}]\!] = \text{Enc}(k_{\text{pub}}, 1)$ if $j = y_i$; otherwise $[\![y_{i,j}]\!] = \text{Enc}(k_{\text{pub}}, 0)$. We obtain the final training data $[\![S_n]\!] = \{([\![\boldsymbol{x}_1]\!], [\![y_1]\!]), \cdots, ([\![\boldsymbol{x}_n]\!], [\![y_n]\!])\}$.

Let $\tilde{S}_{n'} = \{\tilde{\boldsymbol{x}}_1, \cdots, \tilde{\boldsymbol{x}}_{n'}\}$ be a testing data with instance $\tilde{\boldsymbol{x}}_i = (\tilde{x}_{i,1}, \cdots, \tilde{x}_{i,d})$. For every plaintext $\tilde{x}_{i,j}$ with $i \in [n']$ and $j \in [d]$, we search a node t in the binary search tree $\mathcal{BT}_j$, similarly to the node search (Step-I) in Section 3.2, and obtain its ciphertext $[\![\tilde{x}_{i,j}]\!] = [t.\mathit{cipher}, \text{Enc}(k_{\text{pub}}, i)]$. We have the encrypted testing data $[\![\tilde{S}_{n'}]\!] = \{[\![\tilde{\boldsymbol{x}}_1]\!], \cdots, [\![\tilde{\boldsymbol{x}}_{n'}]\!]\}$.

Construction on encrypted random forests

Encrypted random forests consist of individual decision trees $\mathcal{DT}_1, \cdots, \mathcal{DT}_m$, where each tree $\mathcal{DT}_i$ is constructed as follows. We first take a bootstrap sample $[\![S'_n]\!]$ from $[\![S_n]\!]$, and initialize $\mathcal{DT}_i$ with one node of data $[\![S'_n]\!]$. We repeat the following procedure recursively for each leaf node, until the number of training samples is smaller than α, or all instances have the same label in the leaf node:

- Select a k-subset B from d available features randomly without replacement;
- Find the best splitting feature in B and position by Gini impurity from the encrypted data;
- Split the current node into left and right children via the best splitting position and feature.

Such construction is essentially similar to original random forests [1], whereas we require a different way to find the best splitting feature and position based on Gini impurity from the encrypted data.

Table 2: Datasets

Datasets	#Inst	#Feat	Datasets	#Inst	#Feat	Datasets	#Inst	#Feat	Datasets	#Inst	#Feat
wdbc	569	30	adver	3,279	1,558	ailerons	13,750	41	adult	48,842	14
cancer	569	31	bibtex	7,396	1,836	house	22,784	16	mnist	70,000	780
breast	699	9	phpB0	7,797	617	a9a	32,563	123	miniboone	72,998	51
diabetes	768	8	pendigits	10,992	16	amazon	32,769	9	runwalk	88,588	6
german	1,000	24	phish	11,055	30	bank	45,211	17	covtype	581,012	54

Let t be the current leaf node for further splitting with the encrypted training data $\llbracket S_n^t \rrbracket \subseteq \llbracket S_n \rrbracket$, and $\llbracket s \rrbracket_1, \cdots, \llbracket s \rrbracket_\jmath$ denote all possible splitting features and positions in the scope of the corresponding feature subset B from $\llbracket S_n^t \rrbracket$. Here, the information of feature and position can be derived from the corresponding index $i \in [\jmath]$ and subset B.

For each $i \in [\jmath]$, the server partitions the current encrypted training data $\llbracket S_n^t \rrbracket$ into left and right subsets, i.e., $\llbracket S_n^t \rrbracket_i^l$ and $\llbracket S_n^t \rrbracket_i^r$, according to the splitting feature and position $\llbracket s \rrbracket_i$. Let n_l and n_r be the number of training examples in $\llbracket S_n^t \rrbracket_i^l$ and $\llbracket S_n^t \rrbracket_i^r$, respectively, and denote by

$$\llbracket S_n^t \rrbracket_i^l = \{(\llbracket \boldsymbol{x}_1^l \rrbracket, \llbracket y_1^l \rrbracket), \cdots, (\llbracket \boldsymbol{x}_{n_l}^l \rrbracket, \llbracket y_{n_l}^l \rrbracket)\} \quad \text{and} \quad \llbracket S_n^t \rrbracket_i^r = \{(\llbracket \boldsymbol{x}_1^r \rrbracket, \llbracket y_1^r \rrbracket), \cdots, (\llbracket \boldsymbol{x}_{n_r}^r \rrbracket, \llbracket y_{n_r}^r \rrbracket)\} \, .$$

From Eqn. (1), we have Gini impurity $I_G(\llbracket S_n^t \rrbracket, \llbracket s \rrbracket_i)$ as

$$\left[\frac{n_l}{n_l + n_r} \otimes I_G(\llbracket S_n^t \rrbracket_i^l)\right] \oplus \left[\frac{n_r}{n_l + n_r} \otimes I_G(\llbracket S_n^t \rrbracket_i^r)\right] , \tag{12}$$

where $I_G(\llbracket S_n^t \rrbracket_i^l) = 1 \ominus p_l \odot p_l$ and $I_G(\llbracket S_n^t \rrbracket_i^r) = 1 \ominus p_r \odot p_r$, with $p_l = (1/n_l) \otimes (\llbracket y_1^l \rrbracket \oplus, \cdots, \oplus \llbracket y_{n_l}^l \rrbracket)$ and $p_r = (1/n_r) \otimes (\llbracket y_1^r \rrbracket \oplus, \cdots, \oplus \llbracket y_{n_r}^r \rrbracket)$. Here, $\oplus$, $\ominus$, $\otimes$ and $\odot$ denote the CKKS element-wise homomorphic addition, subtraction, multiplication and dot functions, respectively, as in [41].

The client gets plaintexts $\{\mathrm{Dec}(k_{\text{sec}}, I_G(\llbracket S_n^t \rrbracket, \llbracket s \rrbracket_i))\}_{i=1}^{\jmath}$ by decrypting with the secret key k_{sec}, when the server sends ciphertexts $\{I_G(\llbracket S_n^t \rrbracket, \llbracket s \rrbracket_i)\}_{i=1}^{\jmath}$. If all instances have the same label in $\llbracket S_n^t \rrbracket$, then we have $\mathrm{Dec}(k_{\text{sec}}, I_G(\llbracket S_n^t \rrbracket, \llbracket s \rrbracket_i)) = 0$ for each $i \in [\jmath]$, and we set $i^* = -1$; otherwise, we set i^* as

$$i^* \in \arg\min_{i \in [\jmath]} \left\{\mathrm{Dec}(k_{\text{sec}}, I_G(\llbracket S_n^t \rrbracket, \llbracket s \rrbracket_i))\right\} . \tag{13}$$

The client sends index i^* to the server for further splitting. Algorithm 3 presents the detailed descriptions on finding the best splitting feature and position.

For encrypted decision tree, the client requires the $O(\kappa)$ computational complexity with κ leaves nodes, since the client performs constant basic operations for each node. The server takes the $O(\kappa \bar{\jmath} \tau n)$ computational complexity for Eqn. (12), where $\bar{\jmath}$ is an average of number of possible splitting features and positions, and τ and n are the number of labels and training examples, respectively.

Our method takes $O(h)$ communication rounds of $O(\kappa \bar{\jmath})$ communication bandwidth to train an encrypted decision tree of height h. This is because we consider the breadth-first search and aggregate all nodes in the same height and send to the client with a single message at one time.

We do not require bootstrapping for homomorphic encryption in 3-depth homomorphic multiplicative, since we independently compute the splitting feature and position for each node from Eqn. (12). This is different from previous encrypted decision trees [50, 69], which could take expensive computational complexity for bootstrapping [39, 78].

Prediction on encrypted testing dataset

After getting decision trees $\mathcal{DT}_1, \cdots, \mathcal{DT}_m$, we predict label $\llbracket \tilde{y}_i \rrbracket = \mathcal{DT}_1(\llbracket \tilde{\boldsymbol{x}}_i \rrbracket) \oplus \cdots \oplus \mathcal{DT}_m(\llbracket \tilde{\boldsymbol{x}}_i \rrbracket)$ for test instance $\llbracket \tilde{\boldsymbol{x}}_i \rrbracket \in \llbracket \tilde{S}_{n'} \rrbracket$. The server sends ciphertexts $\{\llbracket \tilde{y}_1 \rrbracket, \cdots, \llbracket \tilde{y}_{n'} \rrbracket\}$ to the client, and the client decrypts those ciphertexts, and gets the final plaintext label by $\tilde{y}_i = \arg\max_{j \in [\tau]} \{\mathrm{Dec}\llbracket \tilde{y}_{i,j} \rrbracket\}$.

During such prediction process, the server requires the $O(h)$ computational complexity, since we search from the root to leaf node of tree. The client takes $O(1)$ rounds of communication and communication bandwidth to transfer the testing data and predicting ciphertext without interaction.

Table 3: Comparisons of prediction accuracies (mean±std). •/∘ indicates that our encrypted random forests are significantly better/worse than other compared random forests (pairwise t-tests at 95% significance level). 'NA' means that no results were obtained after running out 10^6 seconds (about 11.6 days).

Dataset	Our encrypted RFs	Original RFs	AnonyRFs	DiffPrivRFs	PPD-ERTs	PivotRFs	MulPRFs	HEldpRFs
wdbc	.9525±.0141	.9617±.0018	.9091±.0205•	.8998±.0024•	.9222±.0037•	.9609±.0101	.9510±.0114	.9195±.0029•
cancer	.9766±.0082	.9824±.0143	.9271±.0016•	.9034±.0578•	.9600±.0022•	.9510±.0130•	.9656±.0102	.9823±.0024
breast	.9855±.0012	.9881±.0011	.9657±.0021•	.9271±.0515•	.9678±.0129•	.9806±.0086	.9769±.0107	.9275±.0023•
german	.7939±.0124	.8033±.0205	.7300±.0214•	.7400±.0141•	.7610±.0168•	.7533±.0122•	.7823±.0154	.7043±.0027•
diabetes	.7641±.0093	.7677±.0309	.7193±.0023•	.7328±.0124•	.7448±.0193	.7419±.0061•	.7611±.0035	.7478±.0193•
adver	.9851±.0011	.9888±.0014	.9278±.0018•	.9390±.0051•	NA	.9664±.0043•	NA	NA
bibtex	.7907±.0054	.7749±.0027•	.7425±.0009•	.7200±.0130•	NA	.7461±.0193•	NA	NA
phpB0	.9380±.0024	.9585±.0043∘	.8641±.0009•	.8920±.0031•	NA	NA	NA	NA
pendigits	.9917±.0024	.9906±.0016	.9072±.0104•	.9154±.0126•	.9639±.0048•	.9070±.0130•	NA	NA
phish	.9798±.0026	.9716±.0018	.9032±.0014•	.9318±.0089•	.9555±.0125•	.9454±.0067•	.9401±.0102•	NA
ailerons	.8795±.0027	.8819±.0015	.8104±.0105•	.8322±.0091•	.8589±.0043•	.8571±.0082•	.8766±.0025	NA
house	.8794±.0007	.8913±.0039∘	.8255±.0011•	.8475±.0025•	.8541±.0149•	.8508±.0016•	.8742±.0023	NA
a9a	.8321±.0011	.8303±.0012	.8046±.0027•	.7909±.0084•	.8345±.0144	.8314±.0071	.8051±.0102•	NA
amazon	.9491±.0109	.9478±.0060	.9193±.0024•	.9104±.0035•	.9221±.0024•	.9401±.0128	.9400±.0032	NA
bank	.8992±.0118	.9029±.0104	.8499±.0089•	.8517±.0064•	.8940±.0147	.8940±.0091	.8827±.0108	NA
adult	.8663±.0019	.8691±.0018	.8206±.0032•	.8355±.0053•	.8452±.0106•	.8243±.0076•	.8594±.0103	NA
mnist	.9674±.0105	.9763±.0101	.9362±.0006•	.9059±.0157•	NA	NA	NA	NA
miniboone	.9497±.0018	.9518 ±.0013	.8977±.0101•	.9111±.0104•	.9301±.00021•	.9501±.0011	NA	NA
runwalk	.9784±.0014	.9798±.0032	.9523±.0024•	.9401±.0040•	.9572±.0074•	.9511±.0071•	NA	NA
covtype	.9787±.0042	.9650±.0104•	.9112±.0015•	.9407±.0018•	.9569±.0134•	NA	NA	NA
win/tie/loss		**2/16/2**	**20/0/0**	**20/0/0**	**17/3/0**	**14/6/0**	**10/10/0**	**19/1/0**

5 Experiment

We conduct experiments on 20 datasets[1] as summarized in Table 2. Most datasets have been well-studied in previous random forests. Besides the original (plaintexts) random forests [1], we compare with six state-of-the-art privacy-preserving random forests in recent years: anonymization random forests **AnonyRFs** [66]; differential-privacy random forests **DiffPrivRFs** [79]; distributed extremely privacy randomized forests **PPD-ERTs** [71]; random forests by partially HE and secure multiparty computation **PivotRFs** [69]; secure-multi-party-computation random forests **MulPRFs** [80]; random forests of fully HE on low-degree polynomial approximations **HEldpRFs** [50].

For all random forests, we train 100 individual decision trees, and randomly select $\lfloor\sqrt{d}\rfloor$ candidate features during node splitting. We set $\alpha = 10$ for datasets of size smaller than 20,000 for our encrypted random forests; otherwise, set $\alpha = 100$, following [81]. For multi-class datasets, we take the one-vs-all method for MulPRFs, since it is limited to binary classification. Other parameters are set according to their respective references, and more details can be found in Appendix D.

Experimental Comparisons

The performance is evaluated by five trials of 5-fold cross validation, and final prediction accuracies are obtained by averaging over these 25 runs, as summarized in Table 3. It is evident that our encrypted random forests take comparable performance with original random forests [1] on plaintexts, and this could well support our Theorem 1 on the preservation of minimum Gini impurity in the construction of random forests. Our encrypted random forests are also comparable to MulPRFs if they can obtain results within 10^6 seconds, since MulPRFs are essentially similar to original random forests, yet with different implementation of secure multi-party computation.

From Table 3, our random forests are better than AnonyRFs and DiffPrivRFs, since the win/tie/loss counts show that our random forests win for most times and never lose. This is because AnonyRFs combine features by anonymization, while DiffPrivRFs add perturbations to features via differential privacy; therefore, both of them cause information lost in privacy process. Our random forests also achieve better performance than PivotRFs, since PivotRFs have to limit trees' depth for random forests due to heavy computations for HE and communications for secure multi-party computation.

Our random forests also outperform PPD-ERTs and HEldpRFs if results are obtained in 10^6 seconds, since PPD-ERTs adopt completely-random splitting, rather than selecting the minimum Gini impurity, while HEldpRFs take homomorphic encryption on features and employ low-degree polynomial approximation. Those approaches have modified the structures of original random forests.

[1]Downloaded from *www.openml.org*

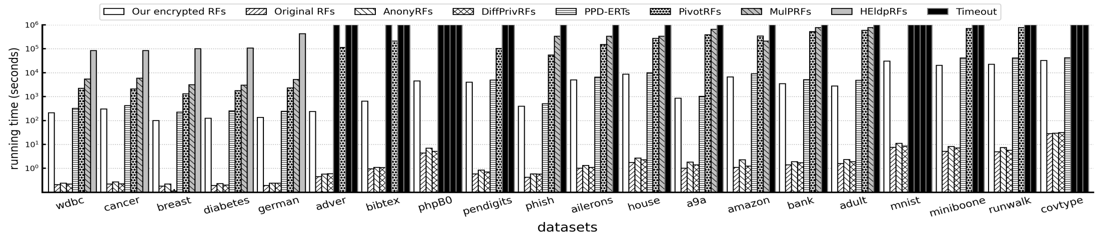

Figure 2: Comparisons of training running time on different random forests. Notice that the y-axis is in log-scale, and full black columns imply that no result was obtained after running out 10^6 seconds (about 11.6 days).

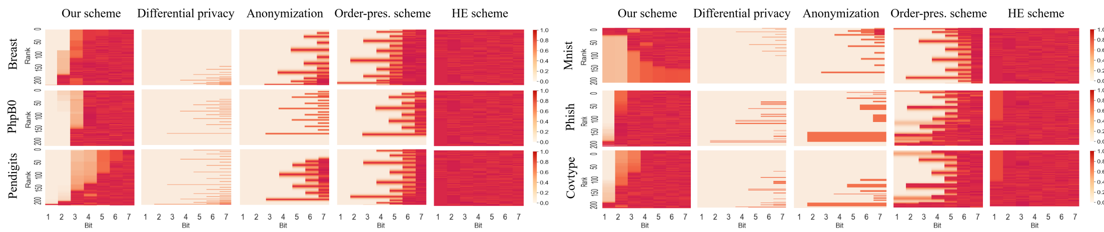

Figure 3: Security comparisons for different schemes: the more red the area, the higher the security.

Running Time

All experiments are performed by c++ on the Ubuntu with 256GB main memory (AMD Ryzen Threadripper 3970X). We compare the training running time of our encrypted random forests and others, and the average CPU time (in seconds) is shown in Figure 2.

As expected, original random forests take the least running time over raw datasets without privacy preservation. Our encrypted random forests take larger running time than AnonyRFs and DiffPrivRFs because they are essentially similar to original random forests, yet with some simple modifications or perturbations on features. Our encrypted random forests take better performance and higher security.

Our encrypted random forests take smaller running time than PPD-ERTs, PivotRFs, MulPRFs and HEldpRFs, in particular for large datasets or high-dimensional datasets, where no results are obtained even after running out 10^6 seconds (almost 11.6 days). Because PPD-ERTs, PivotRFs and MulPRFs require expensive communication cost for multi-parity computation, while PivotRFs and HEldpRFs take heavy computation costs on HE scheme.

Security Analysis

We present security analysis for the first ciphertext $[\![a]\!]_1$ in ciphertext vector $[\![a]\!] = ([\![a]\!]_1, [\![a]\!]_2)$, and the second ciphertext $[\![a]\!]_2$ can be ensured by HE scheme. We compare with four state-of-the-art encryptions: differential privacy [79], anonymization [66], order-preserving scheme [82] and HE scheme [41]. Here, we present results of six datasets and randomly selecting one feature, and trends are similar on other dimensions and datasets. More results can be found in Appendix D.

Figure 3 shows the comparison results, and we take the bitwise leakage matrices to measure the security as in [83]: the more red the area, the higher the security. As expected, HE scheme presents the highest security, yet with heavy computational costs, for example, no results are obtained for datasets of size exceeding 3000 even after running out 10^6 seconds. It is also observed that our scheme presents higher security than the other three schemes, since those schemes simply present perturbations, compression or preserve the entire order information regardless of learning ingredients. In comparison, our scheme could make a good balance between security and computational cost.

6 Conclusion

This work takes one step on data encryption from some crucial ingredients of learning algorithm. We present a new encryption to preserve data's Gini impurity, which plays a crucial role during the construction of random forests. For random forests, we encrypt data features based on our Gini-impurity-preserving scheme, and take the homomorphic encryption scheme CKKS to encrypt data labels. Both theoretically and empirically, we validate the effectiveness, efficiency and security of our proposed method. An interesting work is to exploit other learning ingredients, such as gini index and information gain, for data encryption in the future.

References

[1] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[2] G. Biau and E. Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

[3] L. Mentch and S. Zhou. Randomization as regularization: A degrees of freedom explanation for random forest success. *Journal of Machine Learning Research*, 21:1–36, 2020.

[4] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[5] D. Cutler, T. Edwards Jr, K. Beard, A. Cutler, K. Hess, J. Gibson, and J. Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

[6] Y. Qi. Random forest for bioinformatics. In *Ensemble Machine Learning*, pages 307–323. Springer, 2012.

[7] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[8] M. Belgiu and L. Dragut. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:24–31, 2016.

[9] A. Criminisi and J. Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

[10] S. Basu, K. Kumbier, J. Brown, and Bin B. Yu. Iterative random forests to discover predictive and stable high-order interactions. *Proceedings of the National Academy of Sciences*, 115(8): 1943–1948, 2018.

[11] M. Denil, D. Matheson, and N. Freitas. Consistency of online random forests. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1256–1264, 2013.

[12] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. *Advances in Neural Information Processing Systems*, 26, 2013.

[13] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1): 3–42, 2006.

[14] B. Lakshminarayanan, D. Roy, and Y. Teh. Mondrian forests: Efficient online random forests. In *Advances in Neural Information Processing Systems 27*, pages 3140–3148, 2014.

[15] X. Li, Y. Wang, S. Basu, K. Kumbier, and B. Yu. A debiased mdi feature importance measure for random forests. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[17] B. Menze, B. Kelm, D. Splitthoff, U. Koethe, and F. Hamprecht. On oblique random forests. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 453–469, 2011.

[18] J. Rodriguez, L. Kuncheva, and C. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

[19] S. Wager, T. Hastie, and B. Efron. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *Journal of Machine Learning Research*, 15(1):1625–1651, 2014.

[20] Z.-H. Zhou and J. Feng. Deep forest. *National Science Review*, 6(1):74–86, 2019.

[21] W. Gao, F. Xu, and Z.-H. Zhou. Towards convergence rate analysis of random forests for classification. *Artificial Intelligence*, 313:103788, 2022.

[22] G. Biau, L. Devroye, and G. Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9:2015–2033, 2008.

[23] G. Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, 13(1): 1063–1095, 2012.

[24] E. Scornet, G. Biau, and J. Vert. Consistency of random forests. *Annals of Statistics*, 43(4): 1716–1741, 2015.

[25] J. Mourtada, S. Gaïffas, and E. Scornet. Universal consistency and minimax rates for online mondrian forests. In *Advances in Neural Information Processing Systems 30*, pages 3758–3767, 2017.

[26] C. Tang, D. Garreau, and U. von Luxburg. When do random forests fail. In *Advances in Neural Information Processing Systems 31*, pages 2987–2997, 2018.

[27] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC Press, 2012.

[28] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming*, pages 1–12, 2006.

[29] A. Patil and S. Singh. Differential private random forest. In *Proceedings of the 3rd International Conference on Advances in Computing, Communications and Informatics*, pages 2623–2630, 2014.

[30] X. Li, B. Qin, Y.-Y. Luo, and D. Zheng. A differential privacy budget allocation algorithm based on out-of-bag estimation in random forest. *Mathematics*, 10(22):4338, 2022.

[31] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The sulq framework. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 128–138, 2005.

[32] A. Friedman and A. Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 493–502, 2010.

[33] S. Fletcher and M. Islam. Decision tree classification with differential privacy: A survey. *ACM Computing Surveys*, 52(4):1–33, 2019.

[34] S. Samet and A. Miri. Privacy preserving ID3 using gini index over horizontally partitioned data. In *Proceedings of the 6th International Conference on Computer Systems and Applications*, pages 645–651, 2008.

[35] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. Patterson. Privacy-preserving decision trees over vertically partitioned data. *ACM Transactions on Knowledge Discovery from Data*, 2(3):1–27, 2008.

[36] S. Hoogh, B. Schoenmakers, and P. Chen. Practical secure decision tree learning in a teletreatment application. In *Proceedings of the 18th International Conference on Financial Cryptography and Data Security*, pages 179–194, 2014.

[37] M. Joye and F. Salehi. Private yet efficient decision tree evaluation. In *Proceedings of the 32nd Annual IFIP Conference on Data and Applications Security and Privacy*, pages 243–259, 2018.

[38] Y. Li, Z. Jiang, L. Yao, X. Wang, S. Yiu, and Z. Huang. Outsourced privacy-preserving C4.5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties. *Cluster Computing*, 22(1):1581–1593, 2019.

[39] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[40] L. Ducas and D. Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640, 2015.

[41] J. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of the 23rd International Conference on Theory and Application of Cryptology and Information Security*, pages 409–437, 2017.

[42] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[43] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 201–210, 2016.

[44] Q. Lou, W. Lu, C. Hong, and L. Jiang. Falcon: Fast spectral inference on encrypted data. In *Advances in Neural Information Processing Systems 33*, pages 2364–2374, 2020.

[45] Z. Ghodsi, N. Jha, B. Reagen, and S. Garg. Circa: Stochastic ReLUs for private deep learning. In *Advances in Neural Information Processing Systems 34*, pages 2241–2252, 2021.

[46] X. Li, R. Dowsley, and M. Cock. Privacy-preserving feature selection with secure multiparty computation. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6326–6336, 2021.

[47] A. Aloufi, P. Hu, H. Wong, and S. Chow. Blindfolded evaluation of random forests with multi-key homomorphic encryption. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1821–1835, 2019.

[48] J. Li, X. Kuang, S. Lin, X. Ma, and Y. Tang. Privacy preservation for machine learning training and classification based on homomorphic encryption schemes. *Information Sciences*, 526: 166–179, 2020.

[49] L. Pulido-Gaytan, A. Tchernykh, J. Cortés-Mendoza, M. Babenko, and G. Radchenko. A survey on privacy-preserving machine learning with fully homomorphic encryption. In *Proceedings of the 7th Latin American Conference on High Performance Computing*, pages 115–129, 2021.

[50] A. Akavia, M. Leibovich, Y. Resheff, R. Ron, M. Shahar, and M. Vald. Privacy-preserving decision trees training and prediction. *ACM Transactions on Privacy and Security*, 25(3):1–30, 2022.

[51] K. Cong, D. Das, J. Park, and H. Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security*, pages 563–577, 2022.

[52] J. Brickell, D. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM SIGSAC Conference on Computer and Communications Security*, pages 498–507, 2007.

[53] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *Proceedings of the 17th European Symposium on Research in Computer Security*, pages 424–439, 2009.

[54] R. Tai, J. Ma, Y. Zhao, and S. Chow. Privacy-preserving decision trees evaluation via linear functions. In *Proceedings of the 22nd European Symposium on Research in Computer Security*, pages 494–512, 2017.

[55] A. Tueno, F. Kerschbaum, and S. Katzenbeisser. Private evaluation of decision trees using sublinear cost. *Proceedings on Privacy Enhancing Technologies*, 2019(1):266–286, 2019.

[56] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider. Sok: Modular and efficient private decision tree evaluation. In *Proceedings of Privacy Enhancing Technologies*, pages 187–208, 2019.

[57] K. Sarpatwar, N. Ratha, K. Nandakumar, K. Shanmugam, J. Rayfield, S. Pankanti, and R. Vaculín. Privacy enhanced decision tree inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 154–159, 2020.

[58] K. Han, S. Hong, J. Cheon, and D. Park. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 9466–9471, 2019.

[59] P. Fenner and E. Pyzer-Knapp. Privacy-preserving Gaussian process regression - A modular approach to the application of homomorphic encryption. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 3866–3873, 2020.

[60] A. Sanyal, M. Kusner, A. Gascón, and V. Kanade. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4497–4506, 2018.

[61] Q. Lou, B. Feng, G. Fox, and L. Jiang. Glyph: Fast and accurately training deep neural networks on encrypted data. In *Advances in Neural Information Processing Systems 33*, pages 9193–9202, 2020.

[62] Q. Lou and L. Jiang. HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *Proceedings of the 38th International Conference on Machine Learning*, pages 7102–7110, 2021.

[63] S. Pentyala, R. Dowsley, and M. Cock. Privacy-preserving video classification with convolutional neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8487–8499, 2021.

[64] E. Lee, J. Lee, J. Lee, Y. Kim, Y. Kim, J. No, and W. Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *Proceedings of the 39th International Conference on Machine Learning*, pages 12403–12422, 2022.

[65] J. Wang, Q. Tang, A. Arriaga, and P. Ryan. Novel collaborative filtering recommender friendly to privacy protection. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4809–4815, 2019.

[66] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, pages 25–25, 2006.

[67] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 10(05):557–570, 2002.

[68] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Proceedings of the 14th IEEE International Conference on Privacy, Security and Data Mining*, pages 1–8, 2002.

[69] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. Ooi. Privacy preserving vertical federated learning for tree-based models. *CoRR/abstract*, 2008.06170, 2020.

[70] K. Hamada, D. Ikarashi, R. Kikuchi, and K. Chida. Efficient decision tree training with new data structure for secure multi-party computation. *CoRR/abstract*, 2112.12906, 2021.

[71] A. Aminifar, F. Rabbi, K. Pun, and Y. Lamo. Privacy preserving distributed extremely randomized trees. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1102–1105, 2021.

[72] R. Popa, F. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *Proceedings of the 34th IEEE Conference on Symposium on Security and Privacy*, pages 463–477, 2013.

[73] K. Florian. Frequency hiding order preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667, 2015.

[74] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, page 12, 2012.

[75] S. Goldwasser and S. Micali. Probabilistic encryption how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Conference on Symposium on Theory of Computing*, pages 365–377, 1982.

[76] M. De Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W. Poon, and S. Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2): 217–230, 2017.

[77] A. Tueno, Y. Boev, and F. Kerschbaum. Non-interactive private decision tree evaluation. In *Proceedings of the 34th Annual IFIP Conference on Data and Applications Security and Privacy*, pages 174–194, 2020.

[78] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Proceedings of the 22nd International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33, 2016.

[79] N. Holohan, S. Braghin, P. Mac Aonghusa, and K. Levacher. Diffprivlib: The IBM differential privacy library. *CoRR/abstract*, 1907.02444, 2019.

[80] K. Marcel. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590, 2020.

[81] P. Probst, M. Wright, and A. Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):1301, 2019.

[82] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[83] C. Roy, B. Ding, S. Jha, W. Liu, and J. Zhou. Strengthening order preserving encryption with differential privacy. In *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security*, pages 2519–2533, 2022.

[84] M. Tschantz, S. Sen, and A. Datta. Sok: Differential privacy as a causal property. In *Proceedings of the 41st IEEE Conference on Symposium on Security and Privacy*, pages 354–371, 2020.

[85] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. Mitchell, and R. Cunningham. Sok: Cryptographically protected database search. In *Proceedings of the 38th IEEE Conference on Symposium on Security and Privacy*, pages 172–191, 2017.

A The Decryption for the Our Encryption Method

(i) The decryption for the encryption in Section 3.1

Here, we give the detailed decryption methods for our encryption motivation in Section 3.1. To decrypt the ciphertext $[\![a_{\langle i \rangle}]\!]$ which is defined by Eqn. (5) as follows:

$$[\![a_{\langle i \rangle}]\!] = ([\![a_{\langle i \rangle}]\!]_1, [\![a_{\langle i \rangle}]\!]_2) = \begin{cases} (c_1, \text{Enc}(k_{\text{pub}}, i)) & \text{for } j = 1 \ , \\ (c_j, \text{Enc}(k_{\text{pub}}, i - k_1 - \cdots - k_{j-1})) & \text{for } 2 \leq j \leq s \ . \end{cases} \tag{14}$$

We can get its corresponding plaintext $a_{\langle i \rangle}$ through the following steps:

(1) Find the partition dataset $\mathcal{I}_j$ according to the $[\![a_{\langle i \rangle}]\!]_1$.

(2) Decrypt the ciphertext $[\![a_{\langle i \rangle}]\!]_2$ by the secret key k_{sec} of homomorphic encryption CKKS, and obtain the index $k = \text{Dec}(k_{\text{sec}}, [\![a_{\langle i \rangle}]\!]_2]$ in partition dataset $\mathcal{I}_j$.

(3) Get the plaintext $a_{\langle i \rangle}$ as the k-th sample in partition dataset $\mathcal{I}_j$ and complete the decryption.

Similarly, we give the detailed decryption for our Gini-impurity-preserving encryption in Section 3.2 as follows.

(ii) The Decryption for Our Gini-impurity-preserving Encryption in Section 3.2

As shown in Section 3.2, we present new binary search tree structure to encrypt $a_1, \cdots, a_n$ dynamically, and it is helpful for un-ordered dataset $A = \{(a_1, y_1), \cdots, (a_n, y_n)\}$, or when example (a_i, y_i) arrives in a streaming data, the detailed method can be shown in Algorithm 1. Thus, in order to decrypt the ciphertext $[\![a]\!]$, we use the built binary search tree $\mathcal{BT}$ in encryption phase and the secret key k_{sec} of CKKS to get its corresponding plaintext a as follows:

(1) Let t be a node pointer with the initialization of the root of the built binary search tree $\mathcal{BT}$. Then we search a path downward in binary search tree $\mathcal{BT}$ by comparing with $[\![a]\!]_1$. The search continues to its left child if $[\![a]\!]_1 < t.\textit{cipher}_1$; and the search continues to its right child if $[\![a]\!]_1 > t.\textit{cipher}_1$.

(2) When $[\![a]\!]_1 = t.\textit{cipher}_1$, we obtain the index i of samples which stores in $t.\textit{samples}$ by the secret key k_{sec} of CKKS as: $i = \text{Dec}(k_{\text{sec}}, [\![a]\!]_2)$, and use the index i to get the plaintext a which corresponding to the ciphertext $[\![a]\!]$ as $a = t.\textit{samples}[\text{Dec}(k_{\text{sec}}, [\![a]\!]_2])$.

The detailed decryption method can be shown in Algorithm 4.

The Formal Definition for Our Gini-impurity-preserving Encryption

Here, we give a more formal definition of our Gini-impurity preserving encryption which consists of the following three algorithms:

- $S \leftarrow \text{KeyGen}(t_{\max})$: Generates the secret state S through initializing an empty binary search tree $\mathcal{BT} = \emptyset$,and a security parameter $t_{\max}$, where $c_{\max}$ is a random number with $c_{\max} > n$. Besides, we maintain the interval $[t_{\min}, t_{\max}]$ in each secret state S so as to keep the increasing order of ciphertexts $c_1, c_2, \cdots, c_s$ in Eqn. (5) with $t_{\min} = 0$ in the initial stage. In this way, the ciphertexts are random numbers only containing semi-order information of the plaintexts, and the ciphertext will differ when the same plaintext is encrpyted twice.
- $S', [\![x_i]\!] \leftarrow \text{Encrypt}(S, x_i)$: When sample (x_i, y_i) arrives, we will take three steps to encrypt the feature x_i and update the the secret state to S' as follows:
  (1) **Step-I**: Search a node for sample (x_i, y_i) in binary search tree $\mathcal{BT}$ as shown in Algorithm 1. Let t be a node pointer with the initialization of the root of $\mathcal{BT}$. We search a path downward in $\mathcal{BT}$ by comparing with x_i. The search will terminate when t is a leaf or an empty node.
  (2) **Step-II**: Update the binary search tree $\mathcal{BT}$. After Step-I, we could find a node t for sample (x_i, y_i) and the corresponding interval $[t_{\min}, t_{\max}]$. We directly append the example (x_i, y_i) into $t.\textit{samples}$ if $y_i = y_j$ for every $(x_j, y_j) \in t.\textit{samples}$; otherwise, it is necessary to split the node t according to x_i. Algorithm 2 presents the detailed descriptions on the splitting of node t.

Algorithm 4 Decryption

Input: Tree node t of $\mathcal{BT}$, ciphertext $[\![a]\!]$
Output: plaintext a

```
if [[a]]_1 > t.cipher_1 then
    return Decryption(t.right, [[a]])
else if [[a]]_1 < t.cipher_1 then
    return Decryption(t.left, [[a]])
else
    Return a = t.samples[Dec(k_sec, [[a]]_2)]
end if
```

(3) **Step-III**: Computes a ciphertext $[\![x_i]\!]$ and update the state from S to S'. Append example (a_i, y_i) into $t.\textit{samples}$ and update $t.\textit{cipher}_2 = \text{Enc}(k_{\text{pub}}, |t.\textit{samples}|)$. Then we compute the ciphertext $[\![x_i]\!] = (t.\textit{cipher}_1, t.\textit{cipher}_2)$, and updates the state from S to S' through our updated $\mathcal{BT}$.

- $x_i \leftarrow \text{Decrypt}(S', [\![x_i]\!])$: Computes the plaintext x_i for ciphertext $[\![x_i]\!]$ based on state S' with the built binary search tree $\mathcal{BT}$ in encryption phase and the secret key k_{sec} of CKKS as follows:

(1) Let t be a node pointer with the initialization of the root of the built binary search tree $\mathcal{BT}$. Then we search a path downward in binary search tree $\mathcal{BT}$ by comparing with $[\![x_i]\!]_1$. The search continues to its left child if $[\![x_i]\!]_1 < t.\textit{cipher}_1$; and the search continues to its right child if $[\![x_i]\!]_1 > t.\textit{cipher}_1$.

(2) When $[\![x_i]\!]_1 = t.\textit{cipher}_1$, we obtain the index i of samples which stores in $t.\textit{samples}$ by the secret key k_{sec} of CKKS as: $i = \text{Dec}(k_{\text{sec}}, [\![x_i]\!]_2)$, and use the index i to get the plaintext x_i which corresponding to the ciphertext $[\![x_i]\!]$ as $x_i = t.\textit{samples}[\text{Dec}(k_{\text{sec}}, [\![x_i]\!]_2])$.

B Proof of Theorem 1

Lemma 4. *For dataset $A = \{(a_1, y_1), \cdots, (a_n, y_n)\}$, let $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_s$ be the corresponding partitions as defined by Eqn. (4). There exists a splitting point a^* such that $I_G(A, a^*) = I_G^*(A)$ and*

$$a^* \in \bigcup_{i \in [s-1]} \{\max\{a_k \colon (a_k, y_k) \in \mathcal{I}_i\}/2 + \min\{a_k \colon (a_k, y_k) \in \mathcal{I}_{i+1}\}/2\} \ ,$$

where $I_G(A, a^*)$ and $I_G^*(A)$ are defined by Eqns. (1) and (2), respectively.

Proof. Without loss of generality, we assume that $a_1, a_2, \cdots, a_n$ are distinct elements. Our goal is to solve the optimal splitting point $a^* \in \arg\min_{a \in \mathbb{R}} \{I_G(A, a)\}$, and we begin with some notations used in our proof. For every label $j \in [\tau]$, we denote by

$$\nu_j = |\{i \in [n] \colon y_i = j\}| \ ,$$

i.e., the number of the label j in dataset A. Let a be a splitting point, which splits A into left and right datasets A_a^l and A_a^r, that is,

$$\begin{aligned} A_a^l &= \{(a_i, y_i) \colon a_i \le a, (a_i, y_i) \in A\} \ , \\ A_a^r &= \{(a_i, y_i) \colon a_i > a, (a_i, y_i) \in A\} \ . \end{aligned}$$

For any given $a \in \mathbb{R}$ and $j \in [\tau]$, we further denote by

$$\nu_j^l = |\{i \in [n] \colon y_i = j, a_i \le a\}| \ ,$$

i.e., the number of label j in subsets A_a^l. This follows that

$$I_G(A, a) = w_l - w_l \sum_{j \in [\tau]} \frac{(\nu_j^l)^2}{|A_a^l|^2} + w_r - w_r \sum_{j \in [\tau]} \frac{(\nu_j - \nu_j^l)^2}{(n - |A_a^l|)^2} \ ,$$

where $w_l = |A_a^l|/n$, and $w_r = 1 - w_l$. In the following, we will explore the monotonicity of function $I_G(A, a)$ when

$$\begin{aligned} a &\geq \max\{a_k : (a_k, y_k) \in \mathcal{I}_{i-1}\}/2 + \min\{a_k : (a_k, y_k) \in \mathcal{I}_i\}/2 \\ a &\leq \max\{a_k : (a_k, y_k) \in \mathcal{I}_i\}/2 + \min\{a_k : (a_k, y_k) \in \mathcal{I}_{i+1}\}/2 \ , \end{aligned}$$

for $i = 2, 3, \cdots, s-1$. It is easy to observe that ν_j and ν_j^l keep constants except for $\nu_{j_*}^l$, where j_* denotes the label of instances in $\mathcal{I}_i$. It remains to discuss the variable $\nu_{j_*}^l$, and we have

$$\begin{aligned} & n^2 \frac{\partial I_G(A,a)}{\partial \nu_{j_*}^l} = \frac{1}{n} \sum_{j \in [\tau]} \frac{(\nu_j^l)^2}{(w_l)^2} - 2\frac{\nu_{j_*}^l}{w_l} - \frac{1}{n} \sum_{j \in [\tau]} \frac{(\nu_j - \nu_j^l)^2}{(w_r)^2} + 2\frac{(\nu_{j_*} - \nu_{j_*}^l)}{w_r} \\ = & \ \frac{1}{n} \sum_{j \in [\tau]} \left(\left(\frac{\nu_j^l}{w_l} \right)^2 - \left(\frac{\nu_j - \nu_j^l}{w_r} \right)^2 \right) + 2 \left(\frac{\nu_{j_*} - \nu_{j_*}^l}{w_r} - \frac{\nu_{j_*}^l}{w_l} \right) \\ = & \ \frac{1}{n} \sum_{j \in [\tau], j \neq j_*} \left(\left(\frac{\nu_j^l}{w_l} \right)^2 - \left(\frac{\nu_j - \nu_j^l}{w_r} \right)^2 \right) \\ & + \frac{1}{n} \left(\left(\frac{\nu_{j_*}^l}{w_l} \right)^2 - \left(\frac{\nu_{j_*} - \nu_{j_*}^l}{w_r)} \right)^2 \right) + 2 \left(\frac{\nu_{j_*} - \nu_{j_*}^l}{w_r} - \frac{\nu_{j_*}^l}{w_l} \right) \\ = & \ \frac{1}{n} \sum_{j \in [\tau], j \neq j_*} \left(\left(\frac{\nu_j^l}{w_l} \right)^2 - \left(\frac{\nu_j - \nu_j^l}{w_r} \right)^2 \right) + \left(\frac{\nu_{j_*} - \nu_{j_*}^l}{w_r} - \frac{\nu_{j_*}^l}{w_l} \right) \left(2 - \frac{\nu_{j_*} - \nu_{j_*}^l}{n w_r} - \frac{\nu_{j_*}^l}{n w_l}) \right) \ . \end{aligned}$$

It is easy to observe that

$$0 \leq \frac{\nu_j - \nu_j^l}{w_r} \leq n \quad \text{and} \quad 0 \leq \frac{\nu_j^l}{w_l} \leq n \ \text{ for each } \ j \in [\tau] \ . \tag{15}$$

It is sufficient to consider two cases as follows:

- We consider the first case

$$\sum_{j \in [\tau], j \neq j_*} \left(\left(\frac{\nu_j^l}{w_l} \right)^2 - \left(\frac{\nu_j - \nu_j^l}{w_r} \right)^2 \right) \geq 0 \ ,$$

  and this follows that

$$\begin{aligned} 0 & \leq \sum_{j \in [\tau], j \neq j_*} \left(\left(\frac{\nu_j^l}{w_l} \right)^2 - \left(\frac{\nu_j - \nu_j^l}{w_r} \right)^2 \right) \\ & = \sum_{j \in [\tau], j \neq j_*} \left(\frac{\nu_j^l}{w_l} + \frac{\nu_j - \nu_j^l}{w_r} \right) \left(\frac{\nu_j^l}{w_l} - \frac{\nu_j - \nu_j^l}{w_r} \right) \\ & \leq \sum_{j \in [\tau], j \neq j_*} 2n \left(\frac{\nu_j^l}{w_l} - \frac{\nu_j - \nu_j^l}{w_r} \right) = 2n \sum_{j \in [\tau], j \neq j_*} \left(\frac{\nu_j^l}{w_l} - \frac{\nu_j - \nu_j^l}{w_r} \right) \ . \end{aligned}$$

  We have

$$n - \sum_{j \in [\tau], j \neq j_*} \frac{\nu_j - \nu_j^l}{w_r} \geq n - \sum_{j \in [\tau], j \neq j_*} \frac{\nu_j^l}{w_l} \ , \tag{16}$$

  and it holds that

$$\frac{\nu_{j_*} - \nu_{j_*}^l}{w_r} \geq \frac{\nu_{j_*}^l}{w_l} \ . \tag{17}$$

Combining with Eqns. (15)-(17), we have

$$\frac{\partial I_G(A,a)}{\partial \nu^l_{j*}} \geq 0 \ ,$$

which proves the increasing function of $I_G(A,a)$.

- We now consider the second case

$$\sum_{j\in[\tau],j\neq j_*}\left(\left(\frac{\nu_j^l}{w_l}\right)^2 - \left(\frac{\nu_j-\nu_j^l}{w_r}\right)^2\right) < 0 \ ,$$

and this follows that

$$\begin{aligned}\sum_{j\in[\tau]}\left(\left(\frac{\nu_j^l}{w_l}\right)^2 - \left(\frac{\nu_j-\nu_j^l}{w_r}\right)^2\right) &< \left(\frac{\nu_{j_*}^l}{w_l}\right)^2 - \left(\frac{\nu_{j_*}-\nu_{j_*}^l}{w_r}\right)^2 \\ &= \left(\frac{\nu_{j_*}^l}{w_l} + \frac{\nu_{j_*}-\nu_{j_*}^l}{w_r}\right)\left(\frac{\nu_{j_*}^l}{w_l} - \frac{\nu_{j_*}-\nu_{j_*}^l}{w_r}\right) < 2n\left(\frac{\nu_{j_*}^l}{w_l} - \frac{\nu_{j_*}-\nu_{j_*}^l}{w_r}\right) \ .\end{aligned}$$

We have

$$\begin{aligned} n^2\frac{\partial I_G(A,a)}{\partial \nu^l_{j_*}} &= \frac{1}{n}\sum\nolimits_{j\in[\tau]}\left(\left(\frac{\nu_j^l}{w_l}\right)^2 - \left(\frac{\nu_j-\nu_j^l}{w_r)}\right)^2\right) + 2\left(\frac{\nu_{j_*}-\nu_{j_*}^l}{w_r} - \frac{\nu_{j_*}^l}{w_l}\right) \\ &< 2\left(\frac{\nu_{j_*}^l}{w_l} - \frac{\nu_{j_*}-\nu_{j_*}^l}{w_r}\right) + 2\left(\frac{\nu_{j_*}-\nu_{j_*}^l}{w_r} - \frac{\nu_{j_*}^l}{w_l}\right) = 0 \ , \end{aligned}$$

which proves the decreasing function of $I_G(A,a)$.

In a summary, we prove the piecewise monotonicity of $I_G(A,a)$ for

$$\begin{aligned} a &\geq \max\{a_k\colon (a_k,y_k)\in\mathcal{I}_{i-1}\}/2 + \min\{a_k\colon (a_k,y_k)\in\mathcal{I}_i\}/2 \\ a &\leq \max\{a_k\colon (a_k,y_k)\in\mathcal{I}_i\}/2 + \min\{a_k : (a_k,y_k)\in\mathcal{I}_{i+1}\}/2 \ , \end{aligned}$$

with $i = 2,3,\cdots,s-1$. Moreover, it is easy to observe the monotonicity of $I_G(A,a)$ from $\nu_j^l = 0 (j\neq j_*)$ when

$$a \in (-\infty, (\max\{a_k : (a_k,y_k)\in\mathcal{I}_1\} + \min\{a_k : (a_k,y_k)\in\mathcal{I}_2\})/2] \ ;$$

and from $\nu_j - \nu_j^l = 0 \ (j\neq j_*)$ when

$$a \in [(\max\{a_k : (a_k,y_k)\in\mathcal{I}_{s-1}\} + \min\{a_k : (a_k,y_k)\in\mathcal{I}_s\})/2, +\infty) \ .$$

It is not necessary to consider the splitting point $a^* > \max\{a_k : (a_k,y_k)\in\mathcal{I}_s\}$ with $|A_a^r| = 0$, as well as the splitting point $a^* < \min\{a_k : (a_k,y_k)\in\mathcal{I}_1\}$ with $|A_a^l| = 0$, i.e., without splitting dataset A. This completes the proof. □

Proof of Theorem 1

According to Lemma 4, we could find an optimal splitting point a^* such that

$$a^* \in \bigcup_{i\in[s-1]}\left\{\frac{\max\{a_k : (a_k,y_k)\in\mathcal{I}_i\} + \min\{a_k : (a_k,y_k)\in\mathcal{I}_{i+1}\}}{2}\right\} \ .$$

It is easy to observe that, for $i\in[s-1]$

$$I_G(A, (\max\{a_k : (a_k,y_k)\in\mathcal{I}_i\} + \min\{a_k : (a_k,y_k)\in\mathcal{I}_{i+1}\}/\,2) = I_G(A,(c_i+c_{i+1})/2) \ ,$$

where c_i is the identical ciphertext for those elements in $\in\mathcal{I}_i$, and we complete the proof. □

We also give the following theorem to show that our encryption method in Algorithm 1 could also preserve the minimum Gini impurity over encrypted data.

Theorem 5. *We have $I_G^*(A) = I_G^*(\hat{A})$, for re-sort dataset A by Eqn. (3) and for the corresponding encrypted dataset $\hat{A} = \{(\llbracket a_{\langle 1\rangle}\rrbracket_1, y_{\langle 1\rangle}), \cdots, (\llbracket a_{\langle n\rangle}\rrbracket_1, y_{\langle n\rangle})\}$ accroding to Algorithm 1.*

Proof. The alternative structure $\mathcal{BT}$ for binary search tree to maintain several samples on a node as shown in Section 3.2 keeps the property that the values of all nodes on the left subtree are less than the values of their root nodes while the values of all nodes on the right subtree are larger than the values of their root nodes. In this way, we can obtain a monotone increasing sequence $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_s$ by inorder traversing the built Tree $\mathcal{BT}$ in Algorithm 1. Each $\mathcal{I}_i$ for $j \in [s]$ contains several samples as follows:

$$
\begin{aligned}
\mathcal{I}_1 &= \{(a_{\langle 1\rangle}, y_{\langle 1\rangle}), \cdots, (a_{\langle k_1\rangle}, y_{\langle k_1\rangle})\} \\
\mathcal{I}_2 &= \{(a_{\langle k_1+1\rangle}, y_{\langle k_1+1\rangle}), , \cdots, (a_{\langle k_2\rangle}, y_{\langle k_2\rangle})\} \\
&\cdots \\
\mathcal{I}_s &= \{(a_{\langle k_{s-1}+1\rangle}, y_{\langle k_{s-1}+1\rangle}), \cdots, (a_{\langle n\rangle}, y_{\langle n\rangle})\} \ ,
\end{aligned}
\tag{18}
$$

where $a_{\langle i'\rangle} < a_{\langle j'\rangle}$ when $(a_{\langle i'\rangle}, y_{\langle i'\rangle}) \in \mathcal{I}_i, (a_{\langle j'\rangle}, y_{\langle j'\rangle}) \in \mathcal{I}_j$ and $i < j$. What's more, the data in each sequence $\mathcal{I}_i$ have only the following two cases:

(1) all samples have an identical label in one sequence $\mathcal{I}_j$ ($j \in [s]$), i.e., $y_{\langle i\rangle} = y_{\langle i'\rangle}$ for every $(a_{\langle i\rangle}, y_{\langle i\rangle}), (a_{\langle i'\rangle}, y_{\langle i'\rangle}) \in \mathcal{I}_j$.

(2) all samples have an identical value in one sequence $\mathcal{I}_j$ ($j \in [s]$), i.e., $a_{\langle i\rangle} = a_{\langle i'\rangle}$ for every $(a_{\langle i\rangle}, y_{\langle i\rangle}), (a_{\langle i'\rangle}, y_{\langle i'\rangle}) \in \mathcal{I}_j$.

Thus, we can use Theorem 1 to proof the $I_G^*(A) = I_G^*(\hat{A})$ for the case(1) which includes this situation. Then for the case(2), as the value in each $\mathcal{I}_j$ ($j \in [s]$) is identical, this split value has been preserved and will not change the minimum Gini-impurity of random forests, thus $I_G^*(A) = I_G^*(\hat{A})$ as well.

□

C Proof of Theorem 3

Here, we give a proof of security against Gini-impurity-preserving chosen plaintext attack of our encryption scheme on the $\llbracket a\rrbracket_1$ of Section 3.3 , while the security of homomorphic encryption CKKS on the $\llbracket a\rrbracket_2$ can be found in [41] and has a higher security. We prove by constructing a simulator of the encryption that produces identical outputs for each of the two challenge sequences following [73].

Our simulator proceeds as follows. The adversary sends two sequences of distinct plaintexts $\{a_1^0, a_2^0, \cdots, a_n^0\}$ and $\{a_1^1, a_2^1, \cdots, a_n^1\}$ to a challenger. The simulator simulates the unbiased coin by the random source, i.e. the random source could be replaced by hash functions (a random oracle), to select sequence $\{a_1^b, a_2^b, \cdots, a_n^b\}$ and randomly set their corresponding labels $\{y_1^b, y_2^b, \cdots, y_n^b\}$ with each y_i^b drawn independently and uniformly over $[\tau]$.

Given the sequence $\{a_1^b, a_2^b, \cdots, a_n^b\}$ and the corresponding labels $\{y_1^b, y_2^b, \cdots, y_n^b\}$, the simulator needs to compute the ciphertexts. The simulator first re-sorts the sequence $\{a_1^b, a_2^b, \cdots, a_n^b\}$ and gets a non-decreasing order $\{a_{\langle 1\rangle}^b, a_{\langle 2\rangle}^b, \cdots, a_{\langle n\rangle}^b\}$ where $a_{\langle 1\rangle}^b \le a_{\langle 2\rangle}^b \le \cdots \le a_{\langle n\rangle}^b$ according to Eqns. (3). Then, the simulator gets the partitions $\tilde{\mathcal{I}}^b = \{\mathcal{I}_1^b, \mathcal{I}_2^b, \cdots, \mathcal{I}_s^b\}$ according to Eqns. (4) with

$$
\begin{aligned}
\mathcal{I}_1^b &= \left\{(a_{\langle 1\rangle}^b, y_{\langle 1\rangle}^b), \cdots, (a_{\langle i_1\rangle}^b, y_{\langle i_1\rangle}^b)\right\} \\
\mathcal{I}_2^b &= \left\{(a_{\langle i_1+1\rangle}^b, y_{\langle i_1+1\rangle}^b), , \cdots, (a_{\langle i_2\rangle}^b, y_{\langle i_2\rangle}^b)\right\} \\
&\cdots \\
\mathcal{I}_s^b &= \left\{(a_{\langle i_{s-1}+1\rangle}^b, y_{\langle i_{s-1}+1\rangle}^b), \cdots, (a_{\langle n\rangle}^b, y_{\langle n\rangle}^b)\right\} \ .
\end{aligned}
$$

It is easy to see that the results of dataset partitions $\tilde{\mathcal{I}}^b$ for $\{a_1^0, a_2^0, \cdots, a_n^0\}$ and $\{a_1^1, a_2^1, \cdots, a_n^1\}$ with the selected $\{y_1^b, y_2^b, \cdots, y_n^b\}$ are consistent. Once the dataset partition $\tilde{\mathcal{I}}^b$ has been determined, the simulator encrypts the selected plaintext sequences $\{a_1^b, a_2^b, \cdots, a_n^b\}$ with the dataset partitions

Algorithm 5 Simulator encryption

Input: $\{a_1^b, \cdots, a_n^b\}$ and $\tilde{\mathcal{I}}^b = \{\mathcal{I}_1^b, \cdots, \mathcal{I}_s^b\}$
Output: Built tree $\mathcal{BT}$ and ciphertexts $\{[\![a_1^b]\!], [\![a_2^b]\!], \cdots, [\![a_n^b]\!]\}$

for $i \in [n]$ **do**
  t= root of $\mathcal{BT}$, index=1
  while t is an internal node **and** index==1 **do**
    index=0
    if $\mathcal{I}_t^b > \mathcal{I}_j^b(a_i^b \in \mathcal{I}_j^b)$ **then**
      $t = t$.left, index=1
    else if $\mathcal{I}_t^b < \mathcal{I}_j^b(a_i^b \in \mathcal{I}_j^b)$ **then**
      $t = t$.right, index=1
    end if
  end while
  Update $t.cipher_2 = \text{Enc}(k_{\text{pub}}, |t.samples|)$, and set $[\![a_i^b]\!] = (t.cipher_1, t.cipher_2)$
end for

$\tilde{\mathcal{I}}^b$ by the binary search tree $\mathcal{BT}$ as shown in Algorithm 5. While it encrypts a_i^b and finally stores a_i^b in $\mathcal{BT}$, it also stores $\mathcal{I}_j^b(j \in [s])$ in node of $\mathcal{BT}$, i.e. for each node t in the tree $\mathcal{BT}$ we also know $\mathcal{I}_t^b$.

While the output of Algorithm 5 is deterministic with the dataset partitions $\tilde{\mathcal{I}}^b$, when we set the same random seed in Algorithm 1, the output between Algorithm 5 with $\tilde{\mathcal{I}}^b$ and Algorithm 1 with corresponding $\{y_1^b, y_2^b, \cdots, y_n^b\}$ is indistinguishable. Thus, the simulator produces the same output for both sequences $\{a_1^0, a_2^0, \cdots, a_n^0\}$ and $\{a_1^1, a_2^1, \cdots, a_n^1\}$, and the probability that the adversary wins $\text{Game}_{\text{GIPCPA}}$ against our encryption is negligible larger than $1/2$. When we set different random seeds in Algorithm 1, the probability that the adversary wins $\text{Game}_{\text{GIPCPA}}$ against our encryption will also negligible larger than $1/2$, while it has higher security. $\square$

D Experimental Details

Experimental Settings

Here we give the address of the the comparison methods we used in Section 5. The method without given the address is that the code is not published, and we have reproduced the code according to the content of its paper.

- PPD-ERTs [2] : The PPD-ERTs method is based on the extremely randomized trees algorithm for learning from distributed structured data. The data is assumed to be horizontally partitioned. To share partial information with the mediator, parties employ a secure multiparty computation layer on top of distributed ERT, which is robust to k colluding parties;
- PivotRFs [3] : The PivotRFs method is a private and efficient solution for tree-based models which under the vertical federated learning setting. The solution is based on a hybrid of threshold partially homomorphic encryption and secure multiparty computation techniques;
- MulPRFs[4]: The MulPRFs method is based on the original random forest [1] with the secure multiparty computation library MP-SPDZ, and we take the sh2 protocol which supports semi-honest two-party computation to run this method;
- AnonyRFs [5]: The AnonyRFs method trains the random forests based on anonymization library Mondrian which is a top-down greedy data anonymization algorithm for relational dataset, and proposed by LeFevre et al. [66];
- DiffPrivRFs [6] : The DiffPrivRFs method adopts random forests based on differential privacy library Diffprivlib which is a general-purpose library for experimenting with, investigating and developing applications in, differential privacy;

[2]The code is taken from `https://github.com/AminAminifar/kPPDERT_cloud`.
[3]The code is taken from `https://github.com/nusdbsystem/pivot`.
[4]The code is taken from `https://github.com/csiro-mlai/decision-tree-mpc`.
[5]The code is taken from `https://github.com/qiyuangong/Mondrian`.
[6]The code is taken from `https://github.com/IBM/differential-privacy-library`.

Table 4: Hyperparameters of all tree ensemble models used in our experiments. 'NP' means that no applicable parameters in corresponding method(max_bin denotes the maximun splitting point of each feature).

Parameter	Our Work	PPD-ERTs	HEldpRFs	PivotRFs	MulPRFs	AnonyRFs	DiffPrivRFs	original RFs
max_depth	None	None	5	4	None	None	None	None
n_estimators	100	100	100	100	100	100	100	100
max_features	$\lfloor\sqrt{d}$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$	$\lfloor\sqrt{d}\rfloor$
differentia privacy level ϵ	NP	NP	NP	NP	NP	NP	1	NP
anonymization parameter k	NP	NP	NP	NP	NP	10	NP	NP
multi-party size p	2	2	2	2	2	NP	NP	NP
max_bin	NP	NP	NP	16	NP	NP	NP	NP

Table 5: The hyperparameters of minimum samples α to split a leaf of all tree ensemble models for all datasets used in our experiments.

Parameter	wdbc	cancer	breast	diabetes	german	adver	bibtex	phpB0	pendigits	phish
α	10	10	10	10	10	10	10	10	10	10
Parameter	ailerons	house	a9a	amazon	bank	adult	mnist	miniboone	runwalk	covtype
α	10	100	100	100	100	100	100	100	100	100

- original RFs [7]: The orignal plaintext random forests [1] implemented by sklearn.

Then we give the hyperparameters used in our experiments which are summarized in Table 4 and Table 5. Except for n_estimators and minimum samples α to split a leaf, the other values were copied from their original works [50, 66, 70, 71, 79].

Average Compression Ratio

Here, we focuses on the evaluation of the effectiveness of Gini-impurity preserving encryption in terms of compression ratio, which is a key factor in data security and privacy. The compression ratio measures the extent to which the size of the feature space can be reduced without significant information loss. In Table 2, we provide details of the datasets used in our experiments, which include a diverse range of data types and sizes. We assess the compression performance of our method by comparing the feature space size before and after encryption, as illustrated in Figure 4.

The results show that the average compression ratio achieved across the 20 datasets is 66%, indicating a significant reduction in feature space size while retaining the minimal Gini-impurity of the original data. Notably, our method achieves a compression rate of 0.6% on the runwalk dataset, which is a substantial improvement in terms of data security and privacy. These findings highlight the potential of our approach to enable efficient and effective encryption of sensitive data in various applications.

For further observation, we aimed to test the efficacy of our Gini-impurity preserving encryption method on the UCI iris dataset. The iris dataset is a widely used benchmark dataset in machine learning and consists of 4 attributes: sepal length, sepal width, petal length, and petal width, each with a varying number of distinct values. To evaluate the impact of our encryption method, we applied it to the iris dataset and compared the number of distinct values before and after encryption for each attribute.

Our results in Table 6 indicate that the number of distinct values decreases significantly after encryption, with the largest reduction observed for sepal width. This reduction in the number of distinct values is a direct result of our many-to-one mapping approach, which compresses the range of plaintext values. The minimum compression rate observed in the iris dataset is 21.43%, demonstrating the effectiveness of our encryption algorithm in reducing the size of the plaintext dataset while preserving the Gini impurity. Overall, these findings suggest that our encryption method may be a useful tool for protecting sensitive data in machine learning applications.

[7] The code is taken from `https://github.com/scikit-learn/scikit-learn`.

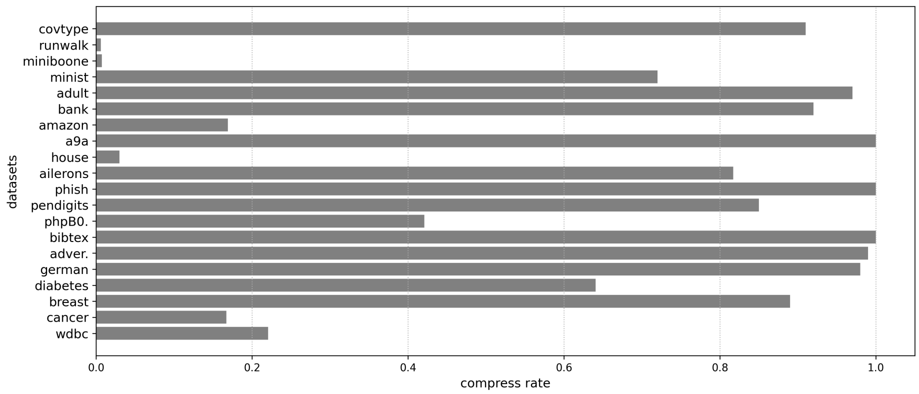

Figure 4: Average compression ratio of the feature space for datasets in Table 2 before and after our Gini-impurity preserving encryption.

Table 6: Number of attribute values in Iris datasets.

Attribute	Size(plaintext)	Size(ciphertext)	compression ration
sepal length	34	24	70.58%
sepal width	22	19	86.36%
petal length	42	9	21.43%
petal width	21	7	33.33%

Running Time

To provide a clear comparison of the efficiency of our method in Section 4 with other existing methods, we have conducted a comprehensive analysis of the computational time required for training. In particular, we have compared the orders of magnitude improvement in runtime of training which achieved by our approach with other state-of-the-art methods, which is presented Table 7.

For prediction inference, we also give the running time comparisons(in seconds) for different methods as shown in Figure 5. The results show that our encrypted random forests could take comparable running time with original random forests, AnonyRFs and DiffPrivRFs, as for our Gini-impurity preserving encryption method only requires $O(h)$ time complexity without other additional operations, where $O(h)$ denotes the height of binary search tree $\mathcal{BT}$.

Furthermore, when considering the computational time required for the trained model obtained in 10^6 seconds (almost 11.6 days), our encrypted random forests show superior efficiency compared to other methods, such as MulPRFs, PPD-ERTs, PivotRFs, and HEldpRFs. This is due to the fact that MulPRFs, PivotRFs, and PPD-ERTs require expensive communication costs for multi-parity computation, while HEldpRFs incur heavy computation costs on the HE scheme. We have also provided a detailed analysis of the orders of magnitude improvement achieved by our approach compared to other methods for prediction inference in Table 8.

Security

In this supplementary experiment, we aim to further investigate the security of the our Gini-impurity preserving encryption method, and present the rest results of the fourteen datasets with random selection of one-dimensional features, and trends are similar on other dimensions, as depicted in Figure 6. We compare our Gini-impurity-preserving scheme with other four encryption methods: differential privacy [79], anonymization [66], order-preserving scheme [82] and HE scheme [41].

To achieve this, we take the bitwise leakage matrices to measure the security as in [83], and we first discretize and scale the feature space values to integers within the range of $[0, 2^7]$. Then we select 200 samples from each dataset and evaluate the security of the feature space. The primary objective

Table 7: The orders of magnitude improvement compared to other approaches in Figure 2. 'NA' means that no results were obtained after running out 10^6 seconds (about 11.6 days).

Dataset	Our encrypted RFs	Original RFs	AnonyRFs	DiffPrivRFs	PPD-ERTs	PivotRFs	MulPRFs	HEldpRFs
wdbc	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$2\times$	$10\times$	$25\times$	$400\times$
cancer	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$1.5\times$	$10\times$	$20\times$	$300\times$
breast	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$2\times$	$13\times$	$30\times$	$10^3\times$
german	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$2\times$	$18\times$	$40\times$	$3000\times$
diabetes	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$2\times$	$15\times$	$25\times$	$850\times$
adver	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	NA	$475\times$	NA	NA
bibtex	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	NA	$328\times$	NA	NA
phpB0	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	NA	NA	NA	NA
pendigits	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$2\times$	$25\times$	NA	NA
phish	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$1\times$	$139\times$	$848\times$	NA
ailerons	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$1\times$	$31\times$	$40\times$	NA
house	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$1\times$	$31\times$	$38\times$	NA
a9a	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$1\times$	$453\times$	$762\times$	NA
amazon	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$1\times$	$51\times$	$31\times$	NA
bank	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$1.5\times$	$149\times$	$220\times$	NA
adult	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$2\times$	$211\times$	$276\times$	NA
mnist	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	NA	NA	NA	NA
miniboone	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$2\times$	$35\times$	NA	NA
runwalk	1	$10^{-4}\times$	$10^{-4}\times$	$10^{-4}\times$	$2\times$	$35\times$	NA	NA
covtype	1	$10^{-3}\times$	$10^{-3}\times$	$10^{-3}\times$	$1\times$	NA	NA	NA

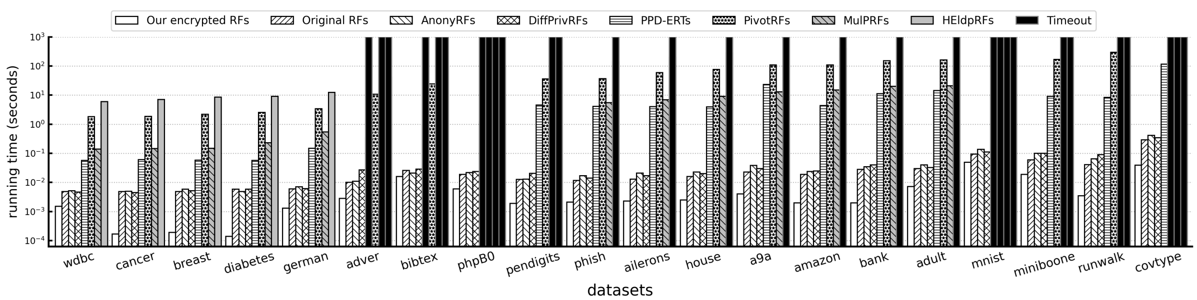

Figure 5: Comparisons of the prediction phase's running time (in seconds) for different methods. Notice that the y-axis is in log-scale.

of this experiment is to safeguard as many bits of the plaintexts as possible. The security analysis is conducted by plotting the results on a color map, while the x-axis represents the individual bits from 1 to 7, while the y-axis indicates the rank of the 200 sampled datasets.

The color gradient, ranging from white to red, represents the degree of security, where a lower security degree is associated with white, and the highest security degree is represented by red. To ensure consistency in our results, we have scaled the security degree values to the range of [0,1]. For instance, a security degree of 0 with color white indicates that there is no security, while a security degree of 1 with color red suggests the highest level of security.

As expected, the HE scheme presents the highest security, yet with heavy computational costs, for example, no results are obtained for datasets of size exceeding 3000 even after running out 10^6 seconds. It is also observe that our scheme presents higher security than the other three schemes, since those schemes simply present perturbations, compression or preserve the entire order information regardless of learning ingredients. In comparison, our scheme could make a good balance between security and computational cost. Through this experiment, we aim tow provide a more comprehensive understanding of the security of our method and identify areas for further improvement.

Table 8: The orders of magnitude improvement compared to other approaches in Figure 5. 'NA' means that no results were obtained after running out 10^6 seconds (about 11.6 days).

Dataset	Our encrypted RFs	Original RFs	AnonyRFs	DiffPrivRFs	PPD-ERTs	PivotRFs	MulPRFs	HEldpRFs
wdbc	1	$3\times$	$3\times$	$3\times$	$38\times$	$1,220\times$	$93\times$	$4,000\times$
cancer	1	$28\times$	$29\times$	$25\times$	$360\times$	$11,052\times$	$851\times$	$41,911\times$
breast	1	$25\times$	$31\times$	$27\times$	$308\times$	$11,631\times$	$776\times$	$44,736\times$
german	1	$4\times$	$5\times$	$4\times$	$115\times$	$2,615\times$	$421\times$	$9,615\times$
diabetes	1	$42\times$	$35\times$	$41\times$	$411\times$	$18,142\times$	$1,642\times$	$64,285\times$
adver	1	$3\times$	$4\times$	$10\times$	NA	$3,821\times$	NA	NA
bibtex	1	$1\times$	$1\times$	$1\times$	NA	$1,528\times$	NA	NA
phpB0	1	$4\times$	$4\times$	$4\times$	NA	NA	NA	NA
pendigits	1	$6\times$	$6\times$	$10\times$	$2384\times$	$18,947\times$	NA	NA
phish	1	$5\times$	$8\times$	$6\times$	$1,966\times$	$1,7619\times$	$2,604\times$	NA
ailerons	1	$6\times$	$9\times$	$8\times$	$1,581\times$	$30,200\times$	$3,600\times$	NA
house	1	$6\times$	$9\times$	$8\times$	$1,581\times$	$30,400\times$	$3,600\times$	NA
a9a	1	$6\times$	$10\times$	$8\times$	$5,482\times$	$27,000\times$	$3,250\times$	NA
amazon	1	$10\times$	$12\times$	$12\times$	$2,208\times$	$54,500\times$	$7,500\times$NA	NA
bank	1	$14\times$	$18\times$	$20\times$	$5,637\times$	$75,500\times$	$10,000\times$	NA
adult	1	$4\times$	$5\times$	$4\times$	$1,967\times$	$22,054\times$	$2,876\times$	NA
mnist	1	$2\times$	$3\times$	$2\times$	NA	NA	NA	NA
miniboone	1	$6\times$	$9\times$	$9\times$	$1,800\times$	$75,000\times$	NA	NA
runwalk	1	$12\times$	$18\times$	$26\times$	$2,413\times$	$84,000\times$	NA	NA
covtype	1	$7\times$	$10\times$	$8\times$	$2,943\times$	NA	NA	NA

E Proof of Bitwise Leakage

In this section, we present an extensive and rigorous security analysis for various methods utilized in our study. Specifically, we provide a comprehensive evaluation of the security properties of our Gini-impurity preserving methods, full homomorphic encryption, anonymization technique, and differential privacy methods. The security analysis is conducted in feature space of datasets with the bitwise leakage matrix proposed by [83]. Our methodology involves utilizing the bitwise leakage profile of plaintexts obtained from both the revealed rank and the adversary's auxiliary knowledge.

By integrating these two sources of information, we are able to provide a more robust and accurate evaluation of the security properties of the aforementioned methods. The resulting analysis provides valuable insights into the strengths and weaknesses of each method, and enables us to make informed decisions regarding their selection and deployment in real-world scenarios.

In this context, by assuming the input domain to be discrete and finite, we are considering a well-defined set of inputs with a predetermined size. The input domain is denoted by $\mathcal{X} = [0, 2^{m-1}]$, which means that the input size is m bits, and the domain ranges from 0 to 2^{m-1}. We also assume that the true input distribution is represented by D, which captures the probability distribution of inputs in the input domain. Moreover, the dataset $X = \{x_1, \ldots, x_n\}$ contains n data points, with each point sampled independently and identically from the true input distribution D.

The adversary $\mathcal{A}$ possesses two types of knowledge to achieve its goal of recovering plaintexts:

- auxiliary knowledge about a distribution D' over the input domain $\mathcal{X}$ [84], which provides additional information to the adversary.
- ciphertexts $\mathcal{C}$ corresponding to X, which represents the snapshot of the encrypted data store, as described in Fuller et al. [85].

The adversary's objective is to recover as many bits of the plaintexts as possible. To represent the plaintexts in X, we use $X(i)$ to denote the plaintext with rank i in X, and $X(i,j)$ to represent the j-th bit of $X(i)$, with $i \in [n], j \in [m]$. The adversary's guess for $X(i,j)$ through the auxiliary knowledge distribution D' is represented by $b(i,j)$ as follows:

$$b(i,j) = \arg\max_{b \in \{0,1\}} \Pr_{D'}\left(X(i,j) = b\right) = \begin{cases} 0 & \text{for} \quad \mathbb{E}_{D'}[X(i,j)] \leq 1/2 \\ 1 & \text{for} \quad \mathbb{E}_{D'}[X(i,j)] > 1/2 \,, \end{cases}$$

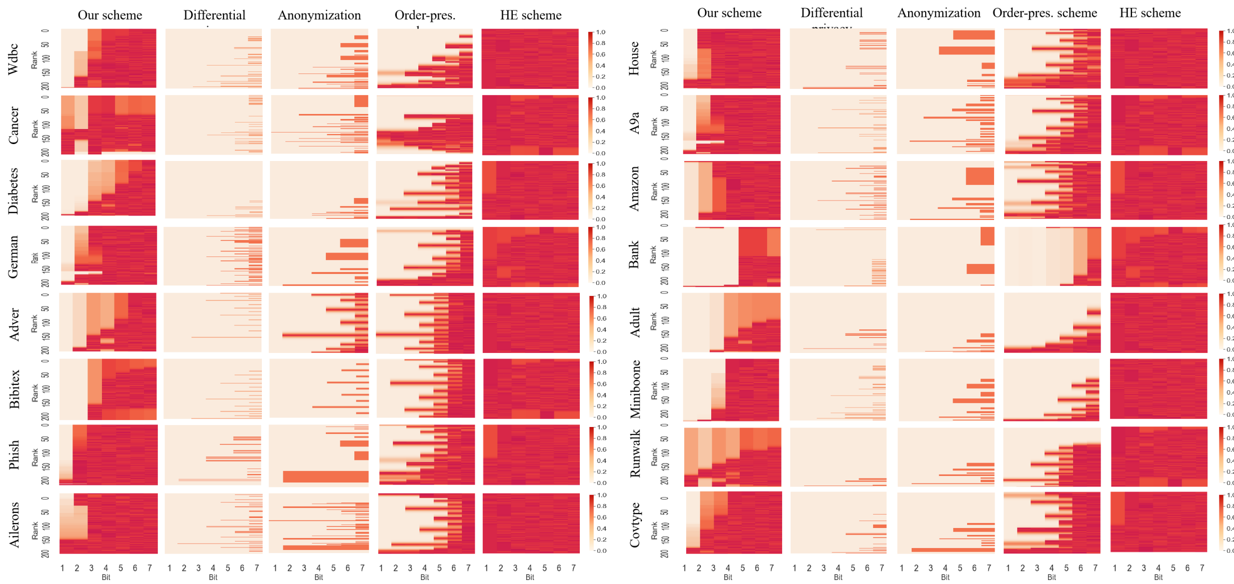

Figure 6: Comparisons of the security degree for the feature space through the bitwise leakage matrix.

for $i \in [n]$ and $j \in [m]$. The adversary aims to maximize the number of correct guesses for $X(i,j)$ using the auxiliary knowledge D'. Then we denote by $\mathcal{L}$ a $n \times m$ matrix with

$$\mathcal{L}(i,j) = \Pr\left(X(i,j) = b(i,j)|D, D'\right) ,$$

for $i \in [n]$ and $j \in [m]$, and the fact shows that (the detailed can be found in [83])

$$\Pr_D\left(X(i,j) = b\right) = \sum_{s \in S_b^j} \Pr_D\left(X(i) = s\right) ,$$

where $i \in [n], j \in [m], b \in \{0,1\}$, and s^j denotes the j-th bit of s and $S_b^j = \{s | s \in \mathcal{X} \text{ and } s^j = b\}$. Then, we have

$$\mathcal{L}(i,j) = \Pr\left(X(i,j) = b(i,j)|D, D'\right) = \sum_{s \in S_b^j} \Pr_D\left(X(i) = s\right) .$$

The variable $\mathcal{L}$ denotes the probability that an adversary can accurately determine the j-th bit of the plaintext with a given rank i. This metric can be considered as a measure of the information security of the ciphertexts $\mathcal{C}$, in the sense that a lower value of $\mathcal{L}$ signifies a higher degree of security. Specifically, the bitwise information security of $\mathcal{C}$ can be quantified as 1-$\mathcal{L}$, as demonstrated in Section 5 of the security analysis. The use of this metric allows for a rigorous and quantitative evaluation of the security properties of the encryption scheme under consideration.

The analysis of the bitwise leakage matrix $\mathcal{L}$ is of paramount importance in evaluating the security level of Gini-impurity-preserving encryption. In this regard, we present a comprehensive study of $\mathcal{L}$, where we examine its various properties and characteristics. Specifically, we investigate the correlation between the elements of $\mathcal{L}$ and the plaintext, ciphertext, and secret keys. Furthermore, we explore the impact of different encryption parameters on the structure and behavior of $\mathcal{L}$. Our analysis reveals that the leakage pattern of $\mathcal{L}$ is highly dependent on the specific encryption scheme used, and hence, it is crucial to carefully design and select the appropriate encryption scheme to minimize the risk of information leakage.

Here we provide analysis for the bitwise leakage matrix $\mathcal{L}$ of our Gini-impurity-preserving encryption as follows.

Theorem 6. *If plaintexts X are encrypted by our Gini-impurity-preserving encryption, for all $i \in [n], j \in [m]$, we have*

$$L(i,j) = P_{i,j} \sum_{s \in S_b^j} \Pr_D\left(X(i) = s\right) + negl. ,$$

where 'negl.' denotes the negligible number and

$$P_{i,j} = \sum_{m \in [i, n-k+i]} \frac{\mathbb{I}(B(X(m), j) = X(i,j))}{n-k+1} \ ,$$

and $B(x, j)$ denotes the j-th bit's value of x.

Proof. Let $\mathcal{C}(i)$ denotes the ciphertext with rank i and corresponding to the dataset $\mathcal{I}_i$, for $i \in [k]$ (defined by Eqn. (4)). For our Gini-impurity-preserving encryption, multiple plaintexts can be transferred into one ciphertext as Eqn. (5). Therefore, the i-th ciphertext $\mathcal{C}(i)$ will corresponding to multiple plaintexts. Then the adversary has to guess the true plaintext $X(i)$ of ciphertext $\mathcal{C}(i)$. Since the adversary only knows that there are $i-1$ ciphertexts smaller than $\mathcal{C}(i)$, and $k-i$ ciphertexts larger than $\mathcal{C}(i)$, the adversary will guess the plaintext $X(i)$ from

$$\{X(m) | m \in [i, n-k+i]\}$$

with the same probability. In this way, the probability of adversary correctly guessing $X(i,j)$ is

$$P_{i,j} = \sum_{m \in [i, n-k+i]} \frac{\mathbb{I}(B(X(m), j) = X(i,j))}{n-k+1} \ ,$$

where $X(m)$ denotes the m-th plaintext in X (rank m) for $m \in [n]$, and $B(x, j)$ denotes the j-th bit's value of x, and $b(i,j)$ denotes the adversary's guess for $X(i,j)$ through the auxiliary knowledge D' as follows:

$$b(i,j) = \arg \max_{b \in \{0,1\}} \Pr_{D'}\left(X(i,j) = b\right) = \begin{cases} 0 & \text{for} \quad \mathbb{E}_{D'}[X(i,j)] \leq 1/2 \\ 1 & \text{for} \quad \mathbb{E}_{D'}[X(i,j)] > 1/2 \ . \end{cases}$$

Thus the probability for the adversary correctly identifies the j-th bit of the plaintext with rank i is

$$L(i,j) = P_{i,j} \Pr_D\left(X(i,j) = b(i,j)\right) + \text{negl.} = P_{i,j} \sum_{s \in S_b^j} \Pr_D\left(X(i) = s\right) + \text{negl.} \ ,$$

where 'negl.' denotes the negligible number, $i \in [n], j \in [m], b \in \{0,1\}$, and s^j denotes the j-th bit of s and $S_b^j = \{s | s \in \mathcal{X} \text{ and } s^j = b\}$. Then we will use the computation of $\Pr_D\left(X(i) = s\right)$ in Roy et al. [83] to formalize the bitwise leakage matrix $\mathcal{L}$.

Lemma 7. *Let D denotes an input distribution and $X = \{x_1, \ldots, x_n\}$ denotes a dataset of size n with each data point sampled i.i.d. from D, then we have*

$$\Pr_D\left(X(i) = x'\right) = \sum_{j=n-i+1}^{n} \binom{n}{j} \left(\Pr_D\left(x < x'\right)\right)^{n-j} \left(\Pr_D\left(x = x'\right)\right)^{j} \ \textit{for} \ \Pr_D\left(x > x'\right) = 0 \ ,$$

and

$$\Pr_D\left(X(i) = x'\right) = \sum_{j=i}^{n} \binom{n}{j} \left(\Pr_D\left(x = x'\right)\right)^{j} \left(\Pr_D\left(x > x'\right)\right)^{n-j} \ \textit{for} \ \Pr_D\left(x < x'\right) = 0 \ ,$$

and otherwise

$$\Pr_D\left(X(i) = x'\right) = \sum_{j=1}^{n} \sum_{k=\max\{1, i-j+1\}}^{\min\{i, n-j+1\}} \binom{n}{k-1, j, n-k-j+1} \Delta_{k-1, j, n-k-j+1} \ .$$

where

$$\Delta_{k-1,j,n-k-j+1} = \left(\Pr_D\left(x < x'\right)\right)^{k-1} \cdot \left(\Pr_D\left(x = x'\right)\right)^{j} \cdot \left(\Pr_D\left(x > x'\right)\right)^{n-k-j+1}$$

We calculate the bitwise leakage matrix $\mathcal{L}$ with Theorem 6 and Lemma 7 and complete the proof. $\square$

We then provide similar analysis for bitwise leakage matrix $\mathcal{L}$ for ϵ-local differential privacy.

Theorem 8. *If plaintexts X are processed by ϵ-local differential privacy, then for all $i \in [n], j \in [m]$, we have:*

$$L(i,j) = \frac{\Pr\left(X(i,j) = b(i,j)\right) + \Pr\left(X(i,j) = X'(i,j)\right)}{2} + \textit{negl.}$$

where 'negl.' denotes the negligible number and

$$b(i,j) = \arg\max_{b\in\{0,1\}} \Pr_{D'}\left(X(i,j) = b\right) = \begin{cases} 0 & \textit{if} \quad \mathbb{E}_{D'}[X(i,j)] \le 1/2\ , \\ 1 & \textit{if} \quad \mathbb{E}_{D'}[X(i,j)] > 1/2\ , \end{cases}$$

and $X'(i,j)$ denotes the j-th bit of the ϵ-local differential privacy disturbed data with rank i in X'.

Proof. It is important to note that the ϵ-differential privacy method is not a form of data encryption. Instead, it employs a statistical technique that adds random noise to the data in order to protect the privacy of individuals in the dataset. Specifically, we are concerned with ϵ-local differential privacy, which involves adding noise to each individual value. This approach means that an adversary attempting to infer the original plaintext $X(i)$ must rely on the ϵ-differential privacy altered data $X'(i)$ and the auxiliary knowledge distribution D'.

In the case of the j-th bit of plaintext $X(i)$, the adversary will attempt to guess the i-th plaintext's j-th bit $\tilde{X}(i,j)$ through a process of deduction based on the available information as follows:

$$\tilde{X}(i,j) = \begin{cases} 1 & \text{for } b(i,j) = 1 \text{ and } X'(i,j) = 1\ , \\ 0 & \text{for } b(i,j) = 0 \text{ and } X'(i,j) = 0\ , \\ \text{randomly select from}\{0,1\} & \text{otherwise}\ . \end{cases}$$

Hence, we can calculate the bitwise leakage based on the probability $\Pr\left(X(i,j) = \tilde{X}(i,j)\right)$, and we have

$$\Pr\left(X(i,j) = \tilde{X}(i,j)\right) = \frac{\Pr\left(X(i,j) = b(i,j)\right) + \Pr\left(X(i,j) = X'(i,j)\right)}{2}\ ,$$

thus we complete the proof. It is worth emphasizing that differential privacy is a well-established framework for protecting the privacy of sensitive data, and its use is supported by a significant body of theoretical and empirical researches. □

In order to gain a deeper understanding of the security of the k-anonymous algorithm, we will conduct an analysis of the bitwise leakage matrix $\mathcal{L}$. This matrix represents the amount of information leakage that occurs when the original data X is compressed into m partitions $\mathcal{K}_1, \mathcal{K}_2, \cdots, \mathcal{K}_m$ by the k-anonymous algorithm as follows:

$$\begin{aligned} \mathcal{K}_1 &= \{X(1), X(2), \cdots, X(k_1)\} \\ \mathcal{K}_2 &= \{X(k_1+1), X(k_1+2), \cdots, X(k_2)\} \\ &\cdots \\ \mathcal{K}_m &= \{X(k_{m-1}+1), X(k_{m-1}+2), \cdots, X(n)\}\ . \end{aligned}$$

This process effectively obscures the identities of the individuals in the data set, making it more difficult for an attacker to re-identify them. However, it is important to assess the level of information leakage that occurs during this process. The bitwise leakage matrix $\mathcal{L}$ is used to quantify the amount of information that can be inferred about an individual from the partitions they belong to. By analyzing this matrix, we can determine the level of privacy that is maintained by the k-anonymous algorithm and identify any potential vulnerabilities that could be exploited by an attacker. Then we give the bitwise leakage matrix $\mathcal{L}$ analysis for k-anonymous algorithm as follows.

Theorem 9. *If plaintexts X are processed by k-anonymous algorithm, then for all $i \in [n], j \in [m]$, we have*

$$L(i,j) = \Pr\left(X(i,j) = b(i,j)\right) + \textit{negl.}\ ,$$

where 'negl.' denotes the negligible number, $X(i,j)$ denotes the j-th bit of $X(i)$ and $X(i) \in \mathcal{K}_t$ and

$$b(i,j) = \arg\max_{b\in\{0,1\}} \Big\{ \sum_{x\in\mathcal{K}_t} \mathbb{I}[B(x,j) = b] \Pr_{D'}(x) \Big\}\ ,$$

and $B(x,j)$ denotes the j-th bit's value of x under the binary representation.

Proof. The concept of k-anonymity is a privacy-preserving technique that aims to protect the identity of individuals in a dataset. It works by grouping together individuals with similar attributes and pooling their data in a larger group, thus making it difficult for an adversary to identify any specific individual in the group. The k-anonymity model ensures that each group has at least k individuals with the same attribute values, which further enhances the security of the data.

When the original data $X(i)$ is pooled in the group $\mathcal{K}_t$, the adversary attempts to guess the j-th bit of the i-th plaintext using the auxiliary knowledge distribution D' and $\mathcal{K}_t$. To achieve this, the adversary guesses $b(i, j)$, which is the value corresponding to the maximum probability of the j-th bit in group $\mathcal{K}_t$ as follows:

$$b(i,j) = \arg\max_{b \in \{0,1\}} \left\{ \sum_{x \in \mathcal{K}_t} \mathbb{I}[B(x,j) = b] \Pr_{D'}(x) \right\} .$$

Hence, the bitwise leakage can then be calculated based on the guess $b(i, j)$. Overall, k-anonymity is an effective technique for preserving the privacy of individuals in a dataset. By grouping individuals with similar attributes, it pools their data in a larger group, making it difficult for an adversary to identify any specific individual in the group. This technique has many applications in fields such as healthcare, finance, and marketing, where sensitive information must be protected. □