Privacy-Preserving Decision Trees Evaluation via Linear Functions

Raymond K.H. Tai(✉), Jack P.K. Ma, Yongjun Zhao, and Sherman S.M. Chow

Information Engineering Department,
Chinese University of Hong Kong, Shatin, Hong Kong
{tkh016,mpk016,zy113,sherman}@ie.cuhk.edu.hk

Abstract. The combination of cloud-based computing paradigm and machine learning algorithms has enabled many complex analytic services, such as face recognition in a crowd or valuation of immovable properties. Companies can charge clients who do not have the expertise or resource to build such complex models for the prediction or classification service. In this work, we focus on machine learning classification with decision tree (or random forests) as the analytic model, which is popular for its effectiveness and simplicity. We propose privacy-preserving decision tree evaluation protocols which hide the sensitive inputs (model and query) from the counterparty. Comparing with the state-of-the-art, we made a significant improvement in efficiency by cleverly exploiting the structure of decision trees, which avoids an exponential number of encryptions in the depth of the decision tree. Our experiment results show that our protocols are especially efficient for deep but sparse decision trees, which are typical for classification models trained from real datasets, ranging from cancer diagnosis to spam classification.

1 Introduction

Machine learning analyzes the pattern of past data for predicting the outcome when given new data as a query. It is widely applicable, say, to credit risk assessment, object recognition, recommendation systems, *etc.* Taking diagnosis of ischemic heart disease as an example, applying machine learning to the past client records help the symptoms evaluation and electrocardiography [21].

Typical machine learning algorithms reveal the query of client and the corresponding classification result to the server. The clients (or users) using these services may not want to reveal their sensitive information. For example, consider revealing every single email to the spam classification server, or food allergy to a diagnosis server. Leakage of sensitive information can be a life-or-death issue.

Another approach is to simply ask the server to give the model to the clients, who then perform the classification themselves. Yet, the computation for a complex classification model is time-consuming for a typical client. Moreover, the

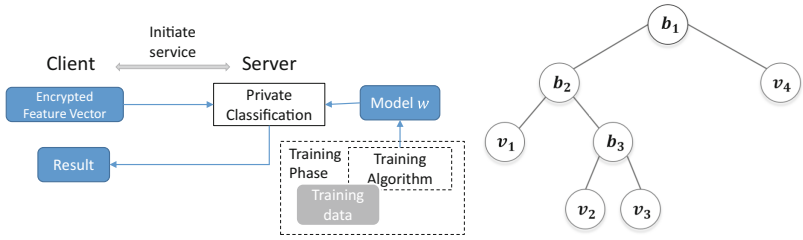
Sherman Chow is supported by the Early Career Scheme and the Early Career Award (CUHK 439713), and General Research Funds (CUHK 14201914) of the Research Grants Council, University Grant Committee of Hong Kong.

S.N. Foley et al. (Eds.): ESORICS 2017, Part II, LNCS 10493, pp. 494–512, 2017.
DOI: 10.1007/978-3-319-66399-9_27

classifier itself, as the result of dedicated research effort which spent a considerable amount of resources, is a valuable asset of the company. This means revealing such "business secret" in clear is not an option. Also, the model is built from, and hence can reveal, sensitive training data such as financial statements or medical records. Recent work by Fredrikson *et al.* [10,11] showed a model inversion attack which can recover information about the training data when given access to the model but can be avoided by taking privacy-aware model training [10]. A comprehensive survey [28] summarized attacks and defenses throughout the process of machine learning, with a focus on confidentiality of training data and differential privacy. Leaking the model not only hurts the reputation of the company due to the compromise of the collected sensitive data, but may even violate laws and regulations such as Health Insurance Portability and Accountability Act (HIPAA). This further motivates the need for the client to send the (encrypted) query for the server to apply the model on it locally.

Ideally, users do not want the server to infer anything about their data, including the classification result, while the server aims to prevent leaking any information about the model. Simply put, the users should only know the classification result[1] and the server should learn nothing.

This paper focuses on preserving privacy in classification using a decision tree, a classifier known for its effectiveness and simplicity. Comparing to deep learning approaches which are more powerful, decision tree approach is more efficient when the data has a hierarchical structure and requires less parameter tuning as well as training cost. Furthermore, in general, the more complicated the underlying (non-privacy-preserving) machine learning it is, the less efficient will be the corresponding privacy-preserving version. Figure 1 gives an overview of the supervised machine learning model. The model w is a decision tree computed from training data. In classification phase, a server holding w receives a *feature vector* from the user as a query, and returns the result by applying w on it.



(a) Private classification service (b) Decision tree example

Fig. 1. Machine learning service under the decision tree model

Decision tree, illustrated in Fig. 1b, is a binary tree structure storing a collection of decision nodes and leaf nodes. Starting at the root, the classifier compares one attribute in the feature vector with a node-specific threshold at a time

[1] Of course, all such systems require a rate-limiting mechanism on the user queries.

and outputs a bit b_i denoting which node to traverse. This process is iterated until arriving at some leaf node, which represents the classification result v_i. The result can be a fixed class or a probability distribution. Recent privacy-preserving machine learning classification protocols [4,35] are built upon decision tree.

In general, it is a secure multi-party computation problem, where one may employ garbled circuit (GC) [16,19,20,23,25] and fully homomorphic encryption (FHE) [14] to implement different kinds of classifiers. However, this approach typically incurs a high cost even for cloud servers. A comprehensive discussion on generic methods can be found in [35]. Tailor-made schemes for specific classifier can be far more efficient [4,12,17,27,35].

1.1 Related Work

Earlier works in the privacy-preserving machine learning mostly focus on protecting data in the training phase [9,15,18,24,31,34]. Some of them use cryptographic techniques including somewhat homomorphic encryption [14], and some leverage differential privacy. Recently, big data analytics and cloud services are gaining popularity. There are many privacy-preserving protocols for cloud-based computation (*e.g.*, feature extraction from encrypted images [29,33]). Following the trend, privacy-preserving machine learning classification is getting more and more attention [2–4,35]. Mohassel and Niksefat [26] proposed protocols that evaluate decision program obliviously, but with the assumption that clients already knew the comparison results, *i.e.*, the comparison nodes are public.

Bost *et al.* [4] build privacy-preserving protocols for hyperplane decision, naïve Bayes, and decision tree classifiers. They first identify what are the core operations of these classifiers, including addition, multiplication, dot products, argmax, and comparison over encrypted data. Many of these can be achieved by semi-homomorphic encryption. However, their construction treats a decision tree as a high-degree polynomial, evaluation thus requires using FHE.

Wu *et al.* [35] propose an improved protocol for decision tree classification. They make use of *oblivious transfer* (OT) and replace FHE by (much more efficient) *additively homomorphic encryption* (AHE), while preserving both functionality and privacy. They also show that their protocols outperform garbled circuit based private evaluation protocols of branching program (which cover decision tree as a special case) proposed by Brickell *et al.* [5] and Barni *et al.* [1].

1.2 Our Contribution

Bost *et al.* [4] treat a decision tree as a high-degree polynomial such that the server can evaluate the result by homomorphic operation on the client's FHE encrypted input. To avoid using heavy FHE, Wu *et al.* [35] require the server to send the decision tree to the client. For security, the server needs to transform it into a randomized and complete tree before sending it to the client. However, this results in the server complexity growing exponentially in the depth of the tree.

Instead of representing a decision tree as a high-degree polynomial, we represent it in the form of linear functions. We exploit the structure of the decision

tree and leverage the concept of path cost. Specifically, we compute the path cost of each leaf node by a linear function and use it to determinate whether a leaf node contains the classification result. This is to avoid multiplications between encrypted messages. In this way, we require neither heavy FHE nor sending randomized complete tree to the client, and achieve by far the most efficient privacy-preserving decision-tree evaluation protocols while keeping the communication cost minimal. The overall performance beats the state-of-the-art asymptotically and empirically. Our basic construction is secure under the semi-honest model which only requires AHE. Moreover, it only requires 4 *communication rounds*, where one sending/receiving action is considered as one round. Let n and t be the *dimension* and the *bit-size* for *each feature* of a feature vector respectively, and m be the *number of decision nodes*. The complexity of our protocol is $O((n+m)t)$ for clients and $O(mt)$ for the server.

We extend our basic construction to achieve one-sided security against malicious client. The only existing one-sided secure protocol is from Wu *et al.* [35], which requires the server to send a randomized complete tree to the client for achieving one-sided security, *even for sparse trees*. Its complexity thus grows exponentially in the *depth of the tree* denoted by d. For the first time, with our new way of decision tree evaluation, we obtain a one-sided secure protocol which does not require this exponential blow-up in both time and space complexities. Notably, it achieves the same asymptotic complexity as the semi-honest one.

Depending only on m, our protocols work well for deep but sparse trees. Table 1 shows a comparison. In practice, the differences between m and 2^d can be huge. Table 2 demonstrates this according to the parameters of UCI dataset [22] considered by Wu *et al.* [35]. The ratios of $2^d/m$ for the listed datasets are $1.6, 3.2, 21.3, 89.0$, and 2259.9. It is practically relevant to consider sparse trees.

Table 1. Summary (t/n: size/number of feature, d/m: depth/number of nodes)

Protocol	Complexity		Number of rounds	Encryption scheme	Leakage	
	Client	Server			Client	Server
Bost *et al.* [4] (semi-honest)	$O((n+m)t)$	$O(mt)$	≥ 6	Leveled-FHE	None	m
Wu *et al.* [35] (semi-honest)	$O((n+m)t+d)$	$O(mt+2^d)$	6	AHE	None	m, d
This work (semi-honest)	$\boldsymbol{O((n+m)t)}$	$\boldsymbol{O(mt)}$	**4**	AHE	None	$\boldsymbol{m}$
Wu *et al.* [35] (One-sided)	$O((n+d)t)$	$O(2^d t)$	2	AHE	None	d
This work (one-sided)	$\boldsymbol{O((n+m)t)}$	$\boldsymbol{O(mt)}$	**4**	AHE	None	$\boldsymbol{m}$

2 Preliminaries

2.1 Decision Tree Classifiers and Important Notations

Let the user input be in the form of an n-dimensional feature vector $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, and the number of decision nodes of the tree be m. Without loss of generality, we assume the decision tree is a *full binary tree*, namely, every node has either 0 or 2 children. For a full binary tree with m non-leaf nodes, the number of leaf nodes is $m + 1$. Let $\mathcal{T} : \mathbb{Z}^n \longmapsto \{v_1, \ldots, v_{m+1}\}$ be the decision tree evaluation function with m decision nodes. Its output $v = \mathcal{T}(x)$ is the classification result which represents the class that x belongs. Each non-leaf node denotes a test on the input attributes. Evaluation starts from the root, descends to the left or right branch based on the test on the current node, and continues until arriving at some leaf node storing $\mathcal{T}(x)$.

2.2 Building Blocks

Homomorphic Encryption. Our protocols use an *additively homomorphic encryption* (AHE) called lifted ElGamal encryption [13] which is like ElGamal but encodes the messages as exponents. Our particular choice of this encryption scheme is for a fairer and easier comparison with prior work [35].

We denote by $[m]$ an encryption of the plaintext m. Lifted ElGamal consists of the following PPT algorithms: Let g be a generator of $\mathbb{G}$. KGen takes in security parameter λ and outputs public key pk and secret key sk. Enc outputs ciphertext $[m]$ when given plaintext (as an exponent) m while DEC outputs g^m when given $[m]$. Of course, one can also encrypt $V = g^v$ without knowing v as regular ElGamal. Add takes in ciphertexts $[m_1]$, $[m_2]$ and outputs a ciphertext $[m_1 + m_2]$. ScalarMul takes in $[m]$ and a scalar n, and outputs $[n \cdot m]$. Again, ScalarMul also works for a regular ElGamal ciphertext of $V = g^v$ without knowing v.

We stress that we do not require recovering the plaintext from the exponents as we only use it to encrypt either a bit or we use the group element as is. Thus our usage does not require solving any discrete logarithm problem. In particular, our constructions simply encrypts key k or classification v using regular ElGamal. Ciphertext $[k']$ of lifted ElGamal is equivalent to a regular ElGamal encryption of the key $k = g^{k'}$. Likewise, classification result can be directly represented by $g^{v'}$ while neither the server nor the client needs to know v'. To avoid clumsy notation, our protocol description simply treats the implicit exponent as the classification result to be encrypted. For actual operation, the server can simply encrypt the group element and multiply its ElGamal ciphertext with another.

Comparison Protocols. Here we review the functionality of the *private comparison protocol* PvtCmp [32] and the private comparison protocol *with conditional key transfer* PvtCmpOT [35] used by our schemes. The main idea of the protocol is that, $x = \sum_{i=1}^{t} 2^{t-i} \cdot x_i > y = \sum_{i=1}^{t} 2^{t-i} \cdot y_i$ if and only if there exist i such that $x_i - y_i - 1 + 3 \cdot \sum_{j<i} x_j \oplus y_j = 0$ where t is a number larger than the length of both x and y. It is easy to see that the equation can be computed over

x encrypted with additively homomorphic encryption and plaintext y. Therefore it can be used to achieve private comparison. More formally, let $[\mathbf{x}]$ be an (additively homomorphic) encryption of x in binary form, $\mathbf{y}$ denote y in binary form, b_1 is a bit chosen randomly as part of the input (which also serves as a secret share), and (k_0, k_1) are the two secret keys (for the protocol with key transfer). The functionalities of these protocols are:

$$\mathsf{PvtCmp}([\mathbf{x}], (\mathbf{y}, b_1)) \longmapsto (b_2, \perp),$$
$$\mathsf{PvtCmpOT}([\mathbf{x}], (\mathbf{y}, b_1, (k_0, k_1)) \longmapsto ((b_2, k_{b_2}), \perp)$$

The bit b_2 is set such that $b_1 \oplus b_2 = (x < y)$, *e.g.*, when $x < y$ and $b_1 = 1$, we have $(x < y) = 1$, so $b_2 = 0$.

Looking ahead, our schemes make black-box use of these protocols. We iterate them over i to perform comparison at each node D_i to decide the traversal. We use $\mathsf{PvtCmp_s}$ and $\mathsf{PvtCmp_c}$ to denote the two stages of the protocol, which is respectively initiated by the server and executed by the client upon receiving the output of the server. Similar notation will be adopted for $\mathsf{PvtCmpOT}$. Figure 2 gives the constructions of the above protocols. For their correctness and security proofs, we refer to [35, Sects. 3.2, 4.2].

Proof of Knowledge. Zero-knowledge proofs can protect the privacy of some inputs while we need to assert a certain property about them. We use the notion of Camenisch and Stadler [6] to represent a zero-knowledge proof of knowledge (PoK). For example, $\mathsf{PoK}\{(\alpha) : c = g^\alpha\}$ denotes a PoK to prove that $c = g^\alpha$ holds for a secret α. Everything else in the equation (c and g here) are public.

Our schemes require proving certain equality for lifted ElGamal and the following disjunctive (DisJ) proof [7]. $\mathsf{PfDisj_{pk}}$ takes in a ciphertext $[m_b]$ and the corresponding randomness used to encrypt, the bit b, $M_0 = g^{m_0}$ and $M_1 = g^{m_1}$, and outputs $\pi = (c_0, f_0, c_1, f_1)$ as a proof that $[m_b]$ encrypted m_0 or m_1. $\mathsf{VerDisj_{pk}}$ takes $[m]$, π, M_0 and M_1, and outputs a bit indicating if π is valid. To turn it into a non-interactive proof, we use a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$.

With input $[\mathbf{x}] = ([x_1], \cdots, [x_t])$, $\mathsf{PfDisj_{pk}}$ is run t times, each with input $[x_q]$ and output π_q, $q \in \{1, \cdots, t\}$. With input $[\boldsymbol{\pi}]$, $\mathsf{VerDisj_{pk}}$ is run t times, each with input $([x_q], \pi_q)$. Figure 3 shows the instantiation of $\mathsf{PfDisj_{pk}}$ and $\mathsf{VerDisj_{pk}}$.

3 Proposed Main Construction

Our main construction is secure under semi-honest adversary. If both parties follow the protocol, it guarantees the client only learns the number of decision nodes and the correct classification result while the server learns nothing.

In the rest of paper, we denote n as dimension of the feature space, t as bits needed to represent one feature, m as number of decision nodes in a decision tree, d as depth of a decision tree. D_i indicates i^{th} decision node, $\mathrm{E}_{i,0}$ ($\mathrm{E}_{i,1}$) indicates left (right) edge of D_i, $\mathsf{ec}_{i,j}$ indicates edge cost of $\mathrm{E}_{i,j}$, L_k indicates k^{th} leaf node, P_k indicates path of L_k from the root, and pc_k indicates the

$\mathsf{PvtCmp_s}([\mathbf{x}], (\mathbf{y}, b_1)), \mathbf{y} = (y_0, \ldots, y_{t-1}) \in \{0,1\}^t$

$[\mathbf{x}] = ([x_1], \cdots [x_t])$

$y = \sum_{i=1}^{t} 2^{t-i} \cdot y_i$

$s \leftarrow 1 - 2 \cdot b_1$

For $i \in \{1, \ldots, t\}$

$r_i \leftarrow \mathbb{Z}_p^*$

$[c_i] \leftarrow [r_i \cdot (x_i - y_i + s + 3 \cdot \sum_{j<i} x_j \oplus y_j)]$

return $C = ([c_1], \cdots, [c_t])$

$\mathsf{PvtCmp_c}(C)$

For $i \in \{1, \ldots, t\}$

If $\mathsf{Dec_{sk}}([c_i]) = 0$

return $b_2 = 1$

return $b_2 = 0$

(a) Details of $\mathsf{PvtCmp_s}$ and $\mathsf{PvtCmp_c}$

$\mathsf{PvtCmpOT_s}([\mathbf{x}], (\mathbf{y}, b_1, (k_0, k_1))$

$[\mathbf{x}] = ([x_1], \cdots [x_t])$

$y = \sum_{i=1}^{t} 2^{t-i} \cdot y_i, \ s \leftarrow 1 - 2 \cdot b_1$

For $i \in \{1, \ldots, t\}$

$r_i \leftarrow \mathbb{Z}_p^*, \ K_0 \leftarrow g^{k_0}, K_1 \leftarrow g^{k_1}$

If $(-y_i - s = 0)$ or $(-y_i - s = -1)$

$[c_i] \leftarrow [k_0 + r_i \cdot (x_i - y_i - s + 2 \cdot \sum_{j<i} x_j \oplus y_j)]$

Else

$[c_i] \leftarrow [k_1 + r_i \cdot (x_i - y_i + s + 2 \cdot \sum_{j<i} x_j \oplus y_j)]$

return $C = ([c_1], \cdots, [c_t], K_0, K_1)$

$\mathsf{PvtCmpOT_c}(C)$

For $i \in \{1, \ldots, t\}$

$k \leftarrow \mathsf{Dec_{sk}}([c_i])$

$K \leftarrow g^k$

If $K = K_0$

return $(b_2 = 0, k_0 = k)$

If $K = K_1$

return $(b_2 = 1, k_1 = k)$

(b) Details of $\mathsf{PvtCmpOT_s}$ and $\mathsf{PvtCmpOT_c}$

Fig. 2. Comparison protocols

corresponding path cost, while v_k indicates classification result belongs to L_k. $[m]$ denotes encryption of message m, $k_{i,j}$ denotes key belongs to $\mathrm{E}_{i,j}$ and $\mathbf{x}$ denotes binary representation of $x = x_1, \cdots, x_t$. $b \leftarrow_{\$} \{0,1\}$ means randomly picking a bit and assign it to b, $c \leftarrow f(b)$ means assigning the output of $f(b)$ to c and $(x < y)$ is a bit equals 1 if the predicate $x < y$ is true, 0 otherwise.

3.1 Intuition

When using a binary decision tree to do classification, each decision node D_i outputs a boolean value $b_i = (x_i < y_i)$ by comparing a given attribute x_i in

$\mathsf{PfDisj}_{\mathsf{pk}}((\mathsf{ct}_0, \mathsf{ct}_1), b, M_0, M_1)$

$\mathsf{pk} = (g, g^s), (\mathsf{ct}_0, \mathsf{ct}_1) = (g^r, g^{sr+m}; r)$

$a_{1-b}, c_b, f_b \leftarrow_{\$} \mathbb{Z}_p$

$A_{1-b} \leftarrow g^{a_{1-b}}, B_{1-b} \leftarrow (g^s)^{a_{1-b}},$

$A_b \leftarrow \dfrac{g^{f_b}}{\mathsf{ct}_0^{c_b}}, B_b \leftarrow \dfrac{(g^s)^{f_b}}{(\mathsf{ct}_1/M_b)^{c_b}}$

$c \leftarrow H(A_{1-b}, B_{1-b}, A_b, B_b)$

$c_{1-b} \leftarrow c - c_b,\ f_{1-b} \leftarrow a_{1-b} + c_{1-b} r$

return $\pi = (c_0, f_0, c_1, f_1)$

$\mathsf{VerDisj}_{\mathsf{pk}}((\mathsf{ct}_0, \mathsf{ct}_1), (c_0, f_0, c_1, f_1), M_0, M_1)$

$\mathsf{pk} = (g, g^s), (\mathsf{ct}_0, \mathsf{ct}_1) = (g^r, g^{sr+m})$

$c \leftarrow c_0 + c_1$

$A_0 \leftarrow \dfrac{g^{f_0}}{\mathsf{ct}_0^{c_0}},\ B_0 \leftarrow \dfrac{(g^s)^{f_0}}{(\mathsf{ct}_1/M_0)^{c_0}},$

$A_1 \leftarrow \dfrac{g^{f_1}}{\mathsf{ct}_0^{c_1}},\ B_1 \leftarrow \dfrac{(g^s)^{f_1}}{(\mathsf{ct}_1/M_1)^{c_1}}$

$c' \leftarrow H(A_0, B_0, A_1, B_1)$

return $(c \stackrel{?}{=} c')$

Fig. 3. Proof of knowledge for lifted ElGamal encryption

the query and the threshold y_i stored in D_i. The boolean value $b_i = 0$ (or 1) indicates the classification result v is in the left (or right) subtree of D_i. After eliminating all impossible leaf nodes, we will get the unique classification result.

Each leaf node L_k has a unique path P_k from the root of the decision tree. This path consists of a unique collection of edges. Observing that each of the leaf node L_k in the right (left) subtree of D_i has path P_k containing the right (left) outgoing edge $\mathrm{E}_{i,1}$ ($\mathrm{E}_{i,0}$) of D_i.

Combining the above two observations, we can get the unique classification result by using each b_i to eliminate leaf nodes that have path containing $\mathrm{E}_{i,1-b_i}$.

In our constructions, we introduce *edge cost* $\mathsf{ec}_{i,j}$ for each edge $\mathrm{E}_{i,j}$ and define *path cost* pc_k as the summation of all $\mathsf{ec}_{i,j}$'s along the path P_k. By setting edge cost ec_{i,b_i} of edge E_{i,b_i} to be zero and edge cost $\mathsf{ec}_{i,1-b_i}$ of edge $\mathrm{E}_{i,1-b_i}$ to be non-zero, path P_k that contains edge $\mathrm{E}_{i,1-b_i}$ will have path cost pc_k being non-zero. Finally, we have v_k being the classification result if and only if $\mathsf{pc}_k = 0$.

Referring to the example in Fig. 1b, setting the edge costs $\mathsf{ec}_{i,0} = b_i$ and $\mathsf{ec}_{i,1} = 1 - b_i$ for every node D_i, the path cost pc_k for each leaf node L_k are: $\mathsf{pc}_1 = b_1 + b_2$; $\mathsf{pc}_2 = b_1 + (1 - b_2) + b_3$; $\mathsf{pc}_3 = b_1 + (1 - b_2) + (1 - b_3)$; $\mathsf{pc}_4 = 1 - b_1$.

By using ABY framework [8], one can switch the underlying primitives to achieve better performance on different operations. However, in our construction, the server needs to hide from the client which attribute is being compared in each node as well as how the path costs are added up. To the best of our knowledge, AHE is the best primitive to achieve our purpose.

3.2 Details of Algorithms

Let $(\mathsf{pk}, \mathsf{sk})$ be a key pair for lifted ElGamal over $\mathbb{G}$ of prime order p. The client holds secret key sk. A feature vector is defined as $(x_1, \ldots, x_n) \in \mathbb{Z}^n$. Let t be the bit-length of x_i, and $\mathbf{x}_i$ denotes the binary representation of x_i.

In Steps 1–3 in Fig. 4, the server and the client interact to derive the comparison results b_i of *every* decision node. The client outputs a bit $b_{i,2}$ such that $b_{i,1} \oplus b_{i,2} = b_i = (x_i < y_i)$ where $b_{i,1}$ is chosen by the server.

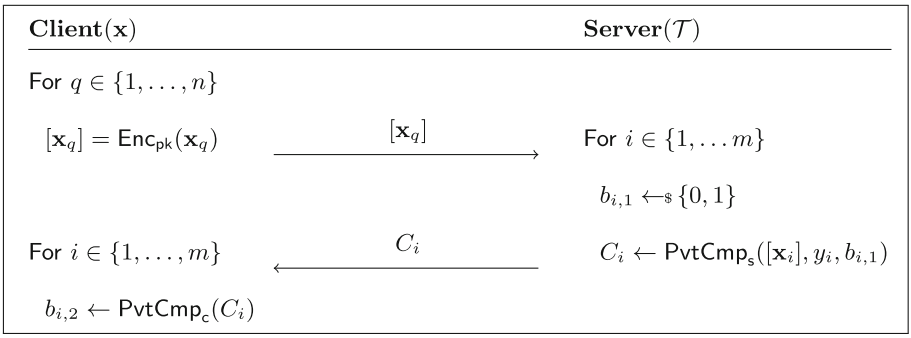


Fig. 4. Private decision tree evaluation in the honest-but-curious model (Steps 1–3)

1. **Client:** Encrypts each component of the feature vector $x_1, \ldots, x_n$ in bits then sends the ciphertexts $[\mathbf{x}_1]$, ..., $[\mathbf{x}_n]$ to the server.
2. **Server:** For each decision node D_i, $i \in \{1, \cdots, m\}$: chooses a random bit $b_{i,1} \leftarrow_{\$} \{0, 1\}$, applies PvtCmp (in Sect. 2.2) on attribute $[\mathbf{x}_i]$, threshold y_i, and bit $b_{i,1}$. After the loop, sends the results of all comparisons to the client.
3. **Client:** For $i \in \{1, \cdots, m\}$: obtains bit $b_{i,2}$, a share of the comparison result.

In Steps 4–6 in Fig. 5, the server returns an encryption of the classification result according to the comparison done in Steps 1–3. Firstly, the client encrypts its share bit $b_{i,2}$ and sends $[b_{i,2}]$ to the server. The server then computes $[b_i] = [b_{i,1} \oplus b_{i,2}]$. We set $\mathsf{ec}_{i,0} = b_i$ to be the edge cost of $\mathrm{E}_{i,0}$ and $\mathsf{ec}_{i,1} = 1 - b_i$ to be the edge cost of $\mathrm{E}_{i,1}$. Then we compute the path cost $\mathsf{pc}_k = \sum_{\mathrm{E}_{i,j} \in \mathrm{P}_k} \mathsf{ec}_{i,j}$ for each leaf node L_k. Finally, the server sends the randomized path cost $[\tilde{\mathsf{pc}}_k] = [r_k \cdot \mathsf{pc}_k]$ with randomized classification result $[\tilde{v}_k] = [r'_k \cdot \mathsf{pc}_k + v_k]$ to the client such that the client can only check whether the path cost $[\mathsf{pc}_k]$ equals zero and can only get the corresponding classification result v_k when pc_k equals zero.

4. **Client:** Sends encryptions of comparison results $[b_{1,2}], \ldots, [b_{m,2}]$ to the server.
5. **Server:**
   For $i \in \{1, \cdots, m\}$: computes $[b_i] \leftarrow [b_{i,1} \oplus b_{i,2}]$.
   For $k \in \{1, \cdots, m+1\}$: computes path cost pc_k of leaf node L_k by taking $([b_1], \ldots, [b_m])$ and the decision tree as input. Chooses $r_k, r'_k \leftarrow_{\$} \mathbb{Z}_p^*$. Then computes randomized path cost $\tilde{\mathsf{pc}}_k \leftarrow [r_k \cdot \mathsf{pc}_k]$ and randomized classification result $[\tilde{v}_k] \leftarrow [r'_k \cdot \mathsf{pc}_k + v_k]$ for leaf node L_k.
   After the loop, chooses a random permutation P over $\{1, \ldots, m+1\}$ and sends $([\tilde{\mathsf{pc}}_{P(1)}], [\tilde{v}_{P(1)}])$, ..., $([\tilde{\mathsf{pc}}_{P(m+1)}], [\tilde{v}_{P(m+1)}])$ to the client.
6. **Client:** For $k' \in \{1, \cdots m+1\}$: checks if $[\tilde{\mathsf{pc}}_{k'}] = [0]$. If so, outputs $v \leftarrow \mathsf{DEC}_{\mathsf{sk}}([\tilde{v}_{k'}])$.

The following lemma shows that our protocol is correct.

Lemma 1. *If both client and server follow our protocol, the client learns the classification results $\mathcal{T}(x)$ at the end.*

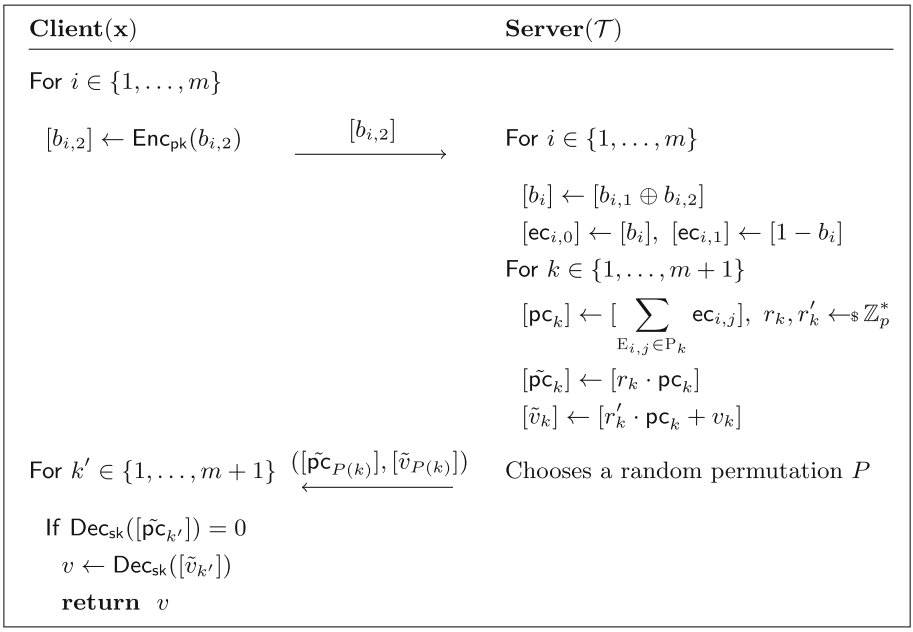


Fig. 5. Private decision tree evaluation in the honest-but-curious model (Steps 4–6)

Proof. By the tree construction, $\mathrm{E}_{i,0} \in \mathrm{P}_k$ indicates $x_i < y_i$ is a constraint of getting v_k, while $\mathrm{E}_{i,1} \in \mathrm{P}_k$ indicates $x_i \geq y_i$ is a constraint of getting v_k. The server obtains $[b_i]$ from the comparison protocol where $b_i = (x_i < y_i)$. The edge costs $\mathsf{ec}_{i,0}$, $\mathsf{ec}_{i,1}$ are defined as b_i and $1 - b_i$ respectively, so that $\mathsf{ec}_{i,0} = (x_i < y_i)$, and $\mathsf{ec}_{i,1} = (x_i \geq y_i)$. The path cost pc_k of classification v_k is defined to be $\sum_{\mathrm{E}_{i,j} \in \mathrm{P}_k} \mathsf{ec}_{i,j}$. Thus, we have v_k is the classification result if and only if $\forall \mathrm{E}_{i,j} \in \mathrm{P}_k$, $\mathsf{ec}_{i,j} = 0$, that is $\mathsf{pc}_k = 0$. Moreover, we have $\tilde{\mathsf{pc}}_k = r_k \cdot \mathsf{pc}_k = 0 \iff \mathsf{pc}_k = 0$ and $\tilde{v}_k = r'_k \cdot \mathsf{pc}_k + v_k = v_k \iff \mathsf{pc}_k = 0$. Therefore the protocol is correct.

3.3 Random Forest Extension

To extend our constructions to random forest, the simplest way is asking the server to send the comparison results of all trees in the forest in Step 2 and likewise all outputs in Step 5 to the client. In this way, the client only knows the total number of decision nodes of all trees, but does not know the number of decision nodes of an individual tree. In addition, we can use the trick of additive secret sharing [35] to further hide each output value v from a tree.

To handle numeric attributes, one can multiply the numeric attributes with a large number to make it an integer. For categorical attributes C_i, we require the client to send encryption of its category $[C]$ to the server. In malicious setting, the client is also required to prove that $[C] = [C_i]$ for some i. Then the server chooses $r_i \leftarrow_{\$} \mathbb{Z}_p^*$ and sets the edge cost ec_i as $r_i \cdot (C_i - C)$.

4 One-Sided Secure Extension

A client that does not follow the protocol specification can learn some information about the threshold or structure of the model in the semi-honest construction. For example, the client can send feature vector not in binary form or send false responses in the comparison protocol.

In the one-sided secure extension, similar to the existing protocol [35], we use *proof of knowledge* and *conditional oblivious transfer* to protect against malicious clients. In particular, the client needs to prove that the encrypted feature vector consists of encryption of either 0 or 1. To ensure the client sends true responses in the comparison protocol, the server uses conditional OT to transfer the keys, such that the client gets either key k_0 or k_1 at each comparison depending on the comparison result. The client needs to prove the response is encryption of either k_0 or k_1. We only require the input attribute values to be in encrypted binary form while the range of attribute values are not restricted (except the bound 2^t) as inputting abnormal attribute values only leads to a corrupted classification result. For this extension, a malicious server can only give corrupted result but learns nothing, while a malicious client can only learn the classification result and the number of decision nodes m.

Figures 6 and 7 show the details of our extension. In Steps 1–3, PoKs are sent along with encrypted inputs to ensure that they are encryption of bits. The

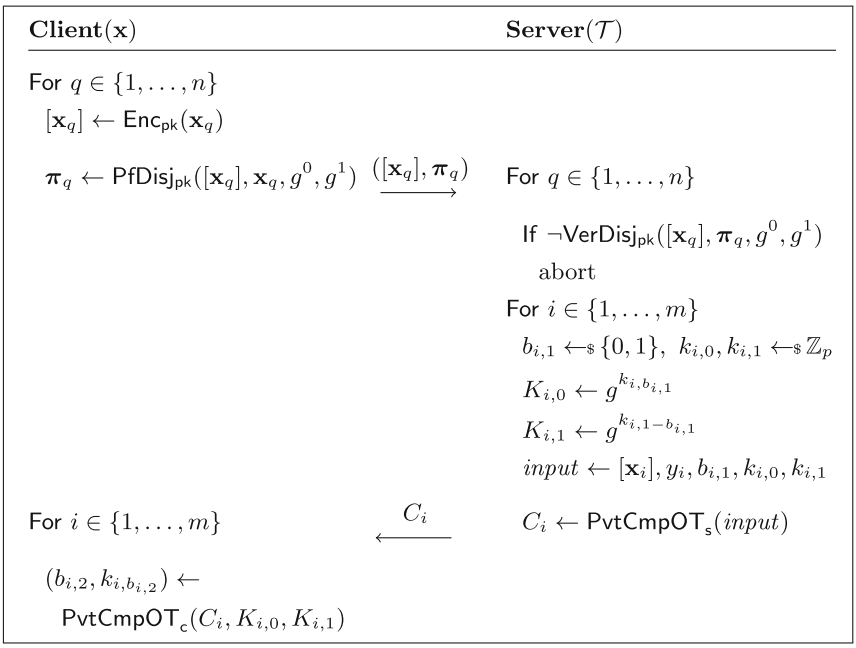


Fig. 6. One-sided secure decision tree evaluation (Steps 1–3)

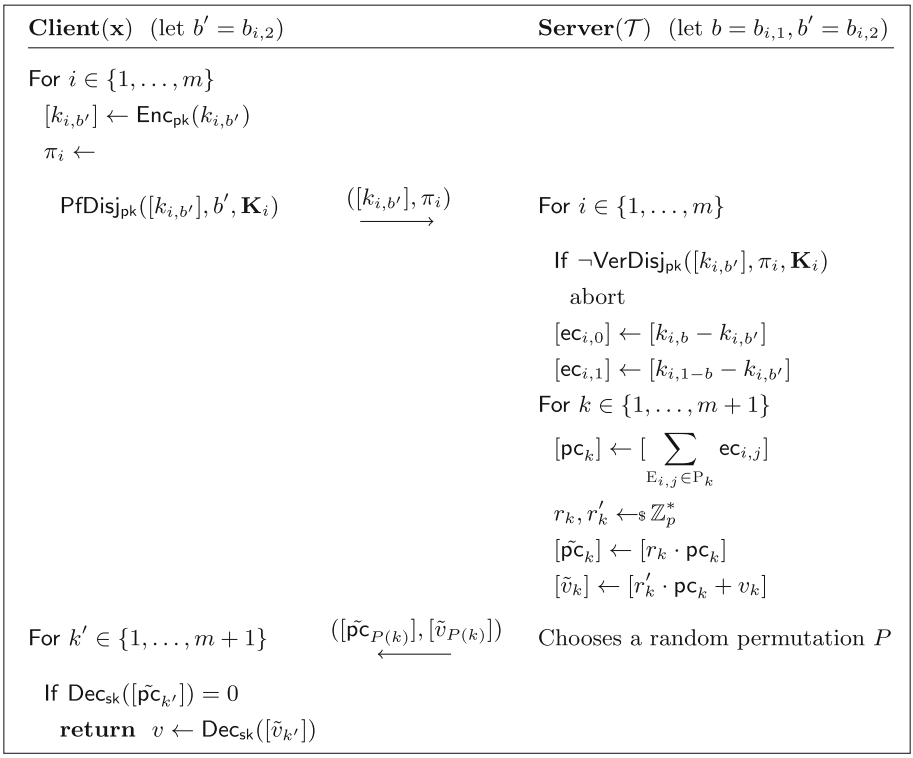


Fig. 7. One-sided secure decision tree evaluation (Steps 4–6)

server and the client involve in comparison protocol with OT. The client outputs $b_{i,2}$ such that $b_{i,1} \oplus b_{i,2} = (x_i < y_i)$ and key $k_{i,b_{i,2}}$ where $b_{i,1}$ is chosen by server.

1. **Client:** Encrypts the feature vector in bits and computes proofs showing the ciphertexts are encryption of 0 or 1. Then sends encrypted feature vector in bits with proofs $([\mathbf{x}_1], \boldsymbol{\pi}_1), \cdots, ([\mathbf{x}_n], \boldsymbol{\pi}_n)$ to the server.
2. **Server:** Verifies all proofs. Aborts if any proofs fail.
   For $i \in \{1, \cdots, m\}$: chooses $b_{i,1} \leftarrow_{\$} \{0, 1\}$ and keys $k_{i,0}$, $k_{i,1} \leftarrow_{\$} \mathbb{Z}_p$.
   Computes $K_{i,0} \leftarrow g^{k_{i,b_{i,1}}}$, $K_{i,1} \leftarrow g^{k_{i,1-b_{i,1}}}$. Then applies $\mathsf{PvtCmpOT}$ (in Sect. 2.2) on attribute $[x_i]$, threshold y_i, bit $b_{i,1}$, and keys $k_{i,0}$, $k_{i,1}$.
   After the loop, sends $(K_{1,0}, K_{1,1}), \ldots, (K_{m,0}, K_{m,1})$ and all messages for m comparisons with OT to the client.
3. **Client:** For $i \in \{1, \cdots, m\}$: computes the comparison result $(b_{i,2}, k_{i,b_{i,2}})$.

In Steps 4–6 in Fig. 7, PoKs are sent along with encrypted keys $[k_{i,b_{i,2}}]$ to ensure that the client sends the correct comparison results. Instead of using $b_{i,2}$, the server uses $k_{b_{i,2}}$ to define the edge cost and compute path cost pc_k. The edge cost $\mathsf{ec}_{i,j}$ is defined as $\mathsf{ec}_{i,j} \leftarrow k_{i,j} - k_{i,b_i}$. As mentioned in Sect. 2.2, the client does not need to solve discrete logarithm to get the keys or the result.

4. **Client:** For $i \in \{1, \cdots, m\}$: encrypts comparison result $k_{i,b_{i,2}}$ and produces proof π_i showing $[k_{i,b_{i,2}}]$ encrypted either one element in K_i where $\mathbf{K}_i = \{g^{k_{i,0}}, g^{k_{i,1}}\}$. Then sends $(([k_{1,b_{1,2}}], \pi_1), \ldots, ([k_{m,b_{m,2}}], \pi_m))$ to server.
5. **Server:** Let $\mathbf{K}_i = \{g^{k_{i,0}}, g^{k_{i,1}}\}$. Verifies all the proofs, aborts if any one fails. For $k \in \{1, \cdots, m+1\}$: computes path cost pc_k of leaf node L_k by taking $([k_{1,b_{1,2}}], \ldots, [k_{m,b_{m,2}}])$, $(k_{1,b_{1,1}}, \ldots, k_{m,b_{m,1}})$, $(k_{1,b_{1,2}}, \ldots, k_{m,b_{m,2}})$ and tree $\mathcal{T}$ as input.
   Chooses $r_k, r'_k \leftarrow_{\$} \mathbb{Z}_p^*$. Then computes $\tilde{\mathsf{pc}}_k \leftarrow [r_k \cdot \mathsf{pc}_k]$ and $[\tilde{v}_k] \leftarrow [r'_k \cdot \mathsf{pc}_k + v_k]$. After the loop, chooses a random permutation P over $\{1, \ldots, m+1\}$ and sends $([\tilde{\mathsf{pc}}_{P(1)}], [\tilde{v}_{P(1)}]), \ldots, ([\tilde{\mathsf{pc}}_{P(m+1)}], [\tilde{v}_{P(m+1)}])$ to the client.
6. **Client:** For $k' \in \{1, \cdots, m+1\}$: checks if $[\tilde{\mathsf{pc}}_{k'}] = [0]$. If so, outputs $v \leftarrow \mathsf{DEC}_{\mathsf{sk}}([\tilde{v}_{k'}])$.

5 Performance Analysis

5.1 Complexity

In the semi-honest construction, the client needs to encrypt its feature vector in binary form, which results in nt ciphertext to be sent to the server. When computing the comparison results, the server computes mt ciphertexts and sends to the client. The client decrypts at most mt ciphertexts and encrypts m responses to the server. Finally, the server computes $2(m+1)$ ciphertexts and sends to the client. The client outputs the result by decrypting at most $(m+2)$ ciphertexts.

The one-sided secure construction requires the client to do additional PoKs. The client sends nt ciphertexts and nt PoKs in the first round. The server verifies all nt PoKs. When computing the comparison results, the server computes mt ciphertexts and sends to the client. The server also computes $2m$ exponentiation (for the g^k term) and sends the results to the client. The client decrypts at most mt ciphertexts and mt exponentiations to get the comparison results. The client then encrypts m responses to the server with m PoKs. The server verifies all m PoKs. Finally, the server computes $2(m+1)$ ciphertexts and sends to the client. The client outputs the result by decrypting at most $(m+2)$ ciphertexts.

5.2 Experiment Setup

We also evaluate our protocols empirically. We implement the lifted ElGamal over elliptic curve secp256k1 with key size 256 bits using mcl library[2] which contains an implementation of lifted ElGamal cryptosystem [30].

For the comparison protocol, we instantiate it with an AHE-based one. While one can easily change the AHE-based comparison protocol to one based on garbled circuits (GC) [19,20,23]; however, if we adapt GC in a trivial way, the client will know what attribute is utilized in each comparison. More concretely, in a decision tree, one attribute may be reused in comparison nodes or not being used (if it is a dummy one), revealing which attribute is used to compare will

[2] https://github.com/herumi/mcl/.

leak information of the decision tree to the client. One can, again, prevent such leakage by utilizing AHE, but this defeats the purpose of replacing it with GC. In addition, the experiment done by Wu *et al.* [35] is based on AHE, so we only consider comparison protocol in AHE for a fairer comparison.

We run our tests on a commodity desktop computer equipped with Intel Core i7-6700 CPU (3.40 GHz) running Ubuntu 16.04 on VMware Workstation allocated with one core and 4 GB of RAM. The times reported are an average over 10 trials. For an easier comparison, we use the Nursery dataset from UCI machine learning repository [22] as in the previous benchmarks [4,35]. We set $t = 64$ as the bit-size for representing a single feature following [35].

5.3 Comparison

Table 2 shows the comparison between our protocols and the existing works. The timing figures for [4,35], marked with "()", are from the experiments performed by Bost *et al.* [4] and Wu *et al.* [35]. Those marked with "(∼)", *e.g.*, (∼290) are read off from the chart which cannot be precise due to the scale. The comparison below used those numbers as is. While we used a similar platform and same security parameter for the experiment, those numbers are *for references only.*

Table 2. Computation time comparison (n: vector dim., d: depth, m: no. of nodes)

Dataset	n	d	m	Protocol		Computation (s)		Bandwidth (MB)
						Client	Server	
Nursery	8	4	4	Semi-honest	Bost *et al.* [4]	(1.58)	(0.80)	(2.58)
					Wu *et al.* [35]	(0.11)	(0.13)	(0.10)
					This work	**0.11**	**0.06**	**0.10**
				One-sided	Wu *et al.* [35]	(0.22)	(0.94)	(0.70)
					This work	**0.40**	**0.51**	**0.25**
(Sparse) tree	16	20	500	Semi-honest	Wu *et al.* [35]	(∼2)	(∼102)	(∼145)
					This work	**2.54**	**7.88**	**4.15**
	16	12	300	One-sided	Wu *et al.* [35]	(∼0.5)	(∼290)	(∼130)
					This work	**2.35**	**10.01**	**5.06**

For tree with $m \approx 2^d$, *e.g.*, nursery data with $d = m = 4$, ours perform similarly to Wu *et al.* [35]. For a *sparse* tree with $m \ll 2^d$, which are abundant as we argued in the introduction, our protocols perform much better. Note that one-sided secure protocol of Wu *et al.* [35] *has to transform a non-complete tree* to a complete tree, resulting in $O(2^d t)$ complexity for the server (see Table 1). While the server is more powerful than the clients in general, yet it is serving multiple clients. Since all these protocols are interactive, a client still needs to wait for the server to complete its computation before getting the final results, the running time of the server unavoidably affects the user experiences.

For concrete benchmark, we consider a sparse tree with $m = 25d$ (following [35]). For $d = 20$, our semi-honest protocol takes 7.88 s for the server which is 13 times better. The total bandwidth required by our protocol is only 4.15 MB, which is only 2.86% of [35]. For a sparse tree of depth 12 with 300 nodes, the one-sided secure protocol of Wu *et al.* [35] operates on a complete tree of $2^d = 4096$ nodes. Our protocol takes 10.01 s for the server which is 29 times better. For both cases, the client takes less than 3 s. The total bandwidth required by our protocol is 5.06 MB, which is only 3.9% of [35].

In general, our protocol greatly reduces the computation time for the server when $m \ll 2^d$ while maintaining similar performance for clients. More importantly, we avoid the exponential (in the tree depth) bandwidth required by the one-sided protocol of Wu *et al.* [35]. It is desirable to save both the local storage and the downloading bandwidth requirement for the client. In favor of existing works, the above figures exclude our saving in network communication time.

5.4 Benchmark on Real Datasets

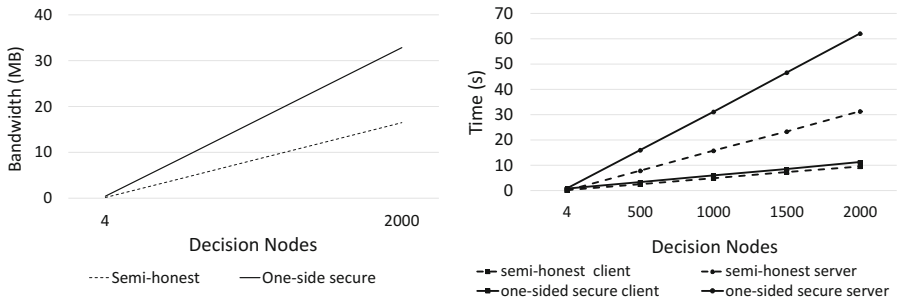
Table 3 shows that our protocols give good performance in various real datasets. Even for housing data which introduces a large number of decision nodes, or spambase date which has high dimension feature vectors and introduces a deep tree, our semi-honest protocol requires less than 2.5 s to complete the classification, and the bandwidth required is less than 1 MB. Our semi-honest protocol outperforms the semi-honest protocol of Wu *et al.* [35] in all datasets. Although the performance of one-sided secure protocol of Wu *et al.* [35] is not provided, by referring to Table 2, we can see that it requires more than 5 min and 130 MB for housing and spambase data due to the great depth, which is not practical. For our one-sided secure protocol, the computation time required is less than 8 s while the bandwidth required is less than 2.5 MB.

Table 3. Performance of semi-honest and one-sided secure protocol on UCL datasets

Dataset	n	d	m	Computation (s)				Bandwidth (MB)			
				Semi-honest			One-sided	Semi-honest			One-sided
				This work	[35]	Difference	This work	This work	[35]	Difference	This work
Heart-disease	13	3	5	0.25	(0.37)	−33%	1.42	0.14	(0.11)	+25%	0.39
Credit-screening	15	4	5	0.27	(0.55)	−50%	1.59	0.16	(0.09)	+70%	0.43
Breast-cancer	9	8	12	0.34	(0.55)	−47%	1.30	0.17	(0.20)	−16%	0.41
Housing	13	13	92	1.98	(4.08)	−51%	4.56	0.85	(1.87)	−54%	1.80
spambase	57	17	58	1.80	(16.60)	−89%	7.47	0.92	(17.41)	−95%	2.28

In both protocols, the bandwidth and computation required by the server grows linearly with the number of decision nodes m. When $m \gg n$ (*e.g.*, for spam), the computation required by clients also grows linearly with m.

Figure 8 shows the bandwidth required and the performance of our protocol.



(a) Total bandwidth vs. number of nodes (b) Computation time vs. number of nodes

Fig. 8. Performance of semi-honest and one sided secure protocol

6 Conclusion

We proposed new privacy-preserving protocols for decision tree classifier. The complexity of the state-of-the-art [35] is exponential in the depth of the decision tree. The major improvement is that the complexity of our protocols grows only linearly with the number of decision nodes. Many models in the form of a decision tree are deep but sparse [22,35]. This makes our protocols more desirable.

Our experiment results show a significant improvement for our semi-honest protocol and one-sided secure protocol. The total bandwidth and the server computation are greatly reduced which makes the one-sided protocol practical. We hope our techniques of exploiting the structure of decision tree will spark future improvement on the efficiency while maintaining security.

A Outline of Security Analysis

Security of Client. In the semi-honest protocol, all the client sends to the server are encrypted feature vector in Step 1 and encrypted share of comparison results in Step 4. The server can thus learn nothing from the client. In the one-sided secure protocol, the server additionally receives PoKs along with ciphertexts in Step 1 and Step 4. By the zero-knowledge property of the PoK, the PoKs leak no information about the client input.

Security of Server. In the semi-honest protocol, all the server sends to the client are secret shares of comparison results in Step 2, and randomized path costs and randomized classifications in Step 5. Without the knowledge of another share (hidden by the server), the share of comparison results appears to be random.

In Step 5, by the correctness of our protocol, only the path cost corresponding to the classification result equals 0 and others equal to non-zero numbers. After randomization, only the one corresponding to the classification result equals 0, while others equal to random numbers. Except for the classification result with 0 path cost, all other classifications are randomized by random path costs. The client can only learn the classification result. Since the number of comparisons equals the number of decision nodes m, and the number of path costs, as well as classifications, equals the number of leaf nodes $m + 1$, our protocol leaks m.

The one-sided secure protocol additionally needs to ensure the client follows the protocol. In Step 1, PoKs are sent with ciphertexts to ensure the client input is encryption of bits. In Step 4, by the security of the comparison protocol, the client only learns one out of two keys according to the share of the comparison result. The client has to encrypt and send the key along with PoK showing the encrypted key is one of the two keys corresponding to the comparison. This ensures the comparison response sent by the client is correct as the client only learns one key and it has to send back either key as the response. As long as the correctness of the client input is ensured, the messages sent in Step 2 and Step 5 leak no information about the server input as the semi-honest setting.

References

1. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04444-1_26
2. Barni, M., Failla, P., Lazzeretti, R., Sadeghi, A., Schneider, T.: Privacy-preserving ECG classification with branching programs and neural networks. Trans. Inf. Forensics Secur. **6**(2), 452–468 (2011)
3. Bos, J.W., Lauter, K.E., Naehrig, M.: Private predictive analysis on encrypted medical data. J. Biomed. Inform. **50**, 234–243 (2014)
4. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS (2015)
5. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS (2007)
6. Camenisch J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: CRYPTO (1997)
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). doi:10.1007/3-540-48071-4_7
8. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
9. Du, W., Han, Y.S. Chen, S.: Privacy-preserving multivariate statistical analysis: linear regression and classification. In: SDM (2004)
10. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: ACM CCS (2015)
11. Fredrikson, M., Lantz, E., Jha, S., Lin, D., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: an end-to-end case study of personalized Warfarin dosing. In: USENIX Security (2014)

12. Frikken, K.B.: Practical private DNA string searching and matching through efficient oblivious automata evaluation. In: DBSec (2009)
13. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). doi:10.1007/3-540-39568-7_2
14. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, Stanford, CA, USA, AAI3382729 (2009)
15. Graepel, T., Lauter, K., Naehrig, M.: ML confidential: machine learning on encrypted data. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 1–21. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37682-5_1
16. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols - Techniques and Constructions. Information Security and Cryptography. Springer, Heidelberg (2010)
17. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: TCC (2007)
18. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.N.: A practical differentially private random decision tree classifier. Trans. Data Priv. **5**(1), 273–295 (2012)
19. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR Gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014 Part II. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_25
20. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008 Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). doi:10.1007/978-3-540-70583-3_40
21. Kononenko, I.: Machine learning for medical diagnosis: history, state of the art and perspective. Artif. Intell. Med. **23**(1), 89–109 (2001)
22. Lichman, M.: UCI machine learning repository. School of Information and Computer Sciences, University of California, Irvine (2013). http://archive.ics.uci.edu/ml
23. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013 Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_1
24. Lindell, Y., Pinkas, B.: Privacy Preserving Data Mining. Springer, Heidelberg (2000)
25. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. J. Cryptol. **28**(2), 312–350 (2015)
26. Mohassel, P., Niksefat, S.: Oblivious decision programs from oblivious transfer: efficient reductions. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 269–284. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32946-3_20
27. Mohassel, P., Niksefat, S., Sadeghian, S., Sadeghiyan, B.: An efficient protocol for oblivious DFA evaluation and applications. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 398–415. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27954-6_25
28. Papernot, N., McDaniel, P.D., Sinha, A., Wellman, M.P.: Towards the science of security and privacy in machine learning. CoRR, abs/1611.03814 (2016)
29. Qin, Z., Yan, K. Ren, K., Chen, C.W., Wang, C.: Towards efficient privacy-preserving image feature extraction in cloud computing. In: ACM Multimedia (2014)
30. Sakai, Y., Emura, K., Hanaoka, G., Kawai, Y., Omote, K.: Methods for restricting message space in public-key encryption. IEICE Trans. **96**(6), 156–1168 (2013)
31. Vaidya, J., Kantarcioglu, M., Clifton, C.: Privacy-preserving naïve Bayes classification. VLDB J. **17**(4), 879–898 (2008)

32. Veugen, T.: Improving the DGK comparison protocol. In: WIFS (2012)
33. Wang, Q., He, M., Du, M., Chow, S.S.M., Lai, R.W.F., Zou, Q.: Searchable encryption over feature-rich data. IEEE Trans. Dependable Sec. Comput. (2017)
34. Wright, R.N., Yang, Z.: Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In: SIGKDD (2004)
35. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. PoPETs **4**, 335–355 (2016)