An Efficient Private Evaluation of a Decision Graph*

Hiroki Sudo[1,2][0000−0003−2222−3371], Koji Nuida[3][0000−0001−8259−9958], and Kana Shimizu[1(✉)][0000−0001−6452−9091]

[1] Waseda University, Tokyo, Japan
hsudo108@ruri.waseda.jp, shimizu.kana@waseda.jp
[2] AIST-Waseda University CBBD-OIL, Tokyo, Japan
[3] The University of Tokyo, Tokyo, Japan
nuida@mist.i.u-tokyo.ac.jp

Abstract. A decision graph is a well-studied classifier and has been used to solve many real-world problems. We assumed a typical scenario between two parties in this study, in which one holds a decision graph and the other wants to know the class label of his/her query without disclosing the graph and query to the other. We propose a novel protocol for this scenario that can obliviously evaluate a graph that is designed by an efficient data structure called the graph level order unary degree sequence (GLOUDS). The time and communication complexities of this protocol are linear to the number of nodes in the graph and do not include any exponential factors. The experiment results revealed that the actual runtime and communication size were well concordant with theoretical complexities. Our method can process a graph with approximately 500 nodes in only 11 s on a standard laptop computer. We also compared the runtime of our method with that of previous methods and confirmed that it was one order of magnitude faster than the previous methods.

Keywords: Decision graph· Homomorphic encryption· GLOUDS.

1 Introduction

Classification is a central topic in machine learning (ML), which is aimed at training a classifier on a set of labeled samples so that the trained classifier can correctly assign one of the class labels to an input query, and has been successfully applied to various real-world problems such as credit scoring, drug discovery, and disease diagnostics [17, 20, 22].

One of the typical online services using ML is a classification service where a service provider has a trained classifier and a user can obtain classification results for his/her data. In fact, software platforms that easily achieve such scenario are already available [1–3], which enables service providers to publish the application programming interfaces (APIs) of the trained classifiers on the cloud server.

*This is the full version of ICISC2018 paper.

Although both service providers and service users can benefit from such classification services, there are certain *privacy* concerns about the data. A natural scenario for an online classification service is for a user to send his/her query (an input to a classifier) to a server and the server to return the classification results based on the classifier. Suppose the online service involves disease diagnostics, where the input to the classifier includes the user's private information such as health records and genetic information. The server's classifier also includes data on donors' private information because the classifier was trained on private data. Various model-inversion attacks are possible [15, 16] in this scenario; they can infer sensitive information being used for training by accessing the trained classifier. Therefore, it is necessary to conceal both the user's query and the server's classifier.

We focused on a decision graph (DG) as a classifier and tackled the problem of private evaluations of the decision graphs. A decision graph is an efficient data structure for the classification rules. It is also described as a decision diagram [7] in the logic synthesis literature and as a branching program (BP) [23] in computer science theory. Compared to complex models such as neural networks, the decision graph is easier to interpret and is therefore often preferred for problems like clinical diagnosis where the interpretation of decision-making is important. We assumed the underlying graph was a binary graph and defined a binary decision graph (BDG) as follows. BDG is a rooted directed acyclic graph (DAG) that consists of a set of nodes of in-degree ≥ 1 and the out-degree of two or zero. A node with the out-degree of zero is called a leaf and has a class label. Each internal node contains a split function that decides whether a query that reaches the node should visit a node connected to the right edge or the node connected to the left edge, depending on the corresponding attribute of the query. We assumed in our study that each split function computed whether or not the input was greater than a threshold t.

The problem setup for this study was as follows: one party (Server) has a BDG and the other party (Client) wants to obtain a classification result. Client's input is a private attribute vector, $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$. The length of the vector and the ordering of the attributes are common information between Server and Client. Client only knows common information and the height of the graph (maximum path length from a root node to a leaf). After computation, Client learns the classification result (a class label); he/she does not learn anything more than what he/she already knows.

1.1 Related Works

Many studies have addressed the problem of private evaluation of classifiers [13, 21, 32]. Brickell et al. [6] and Barni et al. [4] respectively proposed methods which combine Yao's garbled circuit [36] and additively homomorphic encryption (AHE) for private evaluation of the BDG. We will present a detailed comparison of our method with those approaches in Section 4.3. Mohassel et al. [25] have also proposed a method of private evaluation of BDG; however, they assumed

that a user knew all the outputs of all split functions of a server's internal nodes, which differs from our scenario.

Since the decision graph is regarded as a generalized form of a decision tree (DT), we also describe a series of studies for a private evaluation of DTs. Bost et al. [5] proposed a secure decision tree evaluation protocol as part of their work. Their method evaluated a decision tree as a polynomial of Boolean variables using leveled fully homomorphic encryption. Although this method improved efficiency compared to other conventional methods, it still suffered from the problem of computation and communication costs. A recent work by Wu et al. [35] achieved more practical computational time and communication size. Their method was only based on AHE and performed efficiently for shallow trees; however, it did not perform well for the evaluation of deep trees because of its exponential time and communication complexity for the height of the trees. Cock et al. [10] proposed a protocol that achieved time complexity that was similar to Wu et al.'s algorithm and improved runtime by using arithmetic sharing to avoid heavy modular exponentiation. However, their protocol assumed a different problem setup where a trusted initializer participated in the protocol to generate multiplication triplets. The trusted initializer could be removed, but the additional costs of generating the multiplication triplets by the two parties was exponential to the height of the tree, which greatly deteriorated the runtime. Tai et al. [30] formulated a decision tree evaluation as a compact linear function to attain a protocol in which time complexity was only dependent on the number of internal nodes and was independent of the exponential of the tree height.

Protocols for DTs can theoretically be applied to private evaluations of BDG if the underlying graph is transformed into a tree. However, the number of nodes in the tree, that is equivalent to the graph, becomes very large. As we will discuss in Subsection 2.4, two in-coming edges to an internal node cause a copy of all the subordinates of the node on transformation, which leads to the exponential growth in total tree size.

We also noted that a BDG achieved accurate predictions while it achieved lower model complexity than DT [19, 26, 27, 29], and it even achieved considerably improved generalization [29]. A BDG with 3,000 nodes achieved the same accuracy as a DT with 22,000 nodes in the classification of a Kinect dataset in a study by Shotton et al. [29].

1.2 Our Contribution

The five main contributions of this paper are summarized below:

- We propose an efficient protocol for the oblivious evaluation of a BDG. More precisely, the protocol allows two parties, one holding a BDG T, and the other holding an attribute vector, $\boldsymbol{x}$, to determine the class label of $\boldsymbol{x}$, without revealing T and $\boldsymbol{x}$ to the other party in a semi-honest setting.
- The time and communication complexities of our protocol are linear to the number of nodes and the height of T and exclude any exponential factors.

- The DAG of the BDG in our protocol is represented by a look-up vector V, and the other party obliviously refers to V. We demonstrate how the length of V is reduced by using a succinct data structure called GLOUDS to achieve linear complexity. We also propose an efficient design principle for V by exploiting the GLOUDS algorithm to further decrease the length of V.
- An oblivious evaluation of a split function in each internal node is conducted before graph traversal. We propose a novel protocol called eROT that enables the correct edges to obliviously be chosen during traversal.
- We implemented our protocol and tested it on BDGs of various sizes; we found that its actual runtime and communication size were concordant with the theoretical complexities. We also compared the runtime and communication size of our protocol to those in previous studies [4,6] to confirm that our protocol was an order of magnitude faster.

The rest of the paper is organized as follows. Section 2 describes important building blocks for the proposed method and the security model that was assumed for this study. We detail our method in Section 3, evaluate it on various datasets in Section 4, and compare it with the previous methods in Section 4.3. Section 5 concludes the paper.

2 Preliminary

2.1 Notation

We denote vector $\boldsymbol{v}$ as $(v_0, \ldots, v_{n-1})$ and the i-th element of $\boldsymbol{v}$ as $\boldsymbol{v}[i]$. The $\{a_0, \ldots, a_{n-1}\}$ represents a set of size n. The $\{a_i\}_{i=0}^{n-1}$ stands for $\{a_0, \ldots, a_{n-1}\}$. We define the "rotation" of a vector as: given n dimensional vector $\boldsymbol{v}$, the r-rotation of $\boldsymbol{v}$ results in vector $\hat{\boldsymbol{v}}$, each of whose elements is $\hat{\boldsymbol{v}}[(i+r)_{\mathsf{mod}\ n}] = \boldsymbol{v}[i]$. The $\langle P(x) \rangle$ returns 1 if predicate $P(x)$ is true given x, otherwise 0. The notation, $r \in_R A$, means r is a uniformly chosen random value from a set A. We define λ-bit unary representation of $x \in \{0, \ldots, \lambda - 1\}$ as a λ-bit vector that has 1 at x-th least significant bit and has 0 at the other bits, and denote it as $\mathsf{UNARY}_\lambda(x)$.

2.2 Additively Homomorphic Encryption

We used a semantically secure additively homomorphic public-key encryption scheme in our protocol and especially assumed a lifted-ElGamal cryptosystem [11] with plaintext space $\mathbb{Z}_p$ whose message in a ciphertext is located in the exponent. The public-key encryption scheme is equipped with three algorithms:

1. KeyGen**:** outputs a public/private key pair $(\mathsf{pk}, \mathsf{sk})$.
2. $\mathsf{Enc}_{\mathsf{pk}}(m)$**:** outputs a ciphertext $[\![m]\!]$, by encrypting a plaintext m, with pk.
3. $\mathsf{Dec}_{\mathsf{sk}}([\![m]\!])$**:** outputs a plaintext m, by decrypting a ciphertext $[\![m]\!]$, with sk.

$[\![m]\!]$ represents a ciphertext of a plaintext m. Likewise, $[\![\boldsymbol{v}]\!]$ represents a ciphertext vector, each of whose elements is an encryption of each element of a vector $\boldsymbol{v}$.

A public key of AHE has $\oplus$, $\otimes$, $\ominus$ operations on ciphertexts described as follows. Given two plaintexts m_1, m_2, we can compute $[\![m_1 + m_2]\!] = [\![m_1]\!] \oplus [\![m_2]\!]$ by using $\oplus$ operation. We can also compute multiplication by a constant k ($[\![k \cdot m]\!] = k \otimes [\![m]\!]$). Negation on a ciphertext is represented by $\ominus[\![m]\!]$.

In our setting, the user generates and holds a public/private key pair $(\mathsf{pk}, \mathsf{sk})$, and the server only receives a public key pk so that only the user can decrypt ciphertexts and the server can only conduct encryption and additively homomorphic computation.

2.3 Oblivious Transfer

Oblivious transfer (OT) is a secure two-party protocol between a sender and a chooser. A chooser in 1-out-of-N OT specifies an index $i \in \{0, \ldots, N-1\}$, and only obtains the i-th element of the sender's vector $\boldsymbol{v}$, without disclosing i to the sender (i.e., the sender learns nothing). We denote the execution of OT with an index i, and a vector $\boldsymbol{v}$ by $OT_1^N(i, \boldsymbol{v})$. While there are several efficient implementations that achieve OT_1^N functionality, we use simple protocols based on additively homomorphic operation which require $O(N)$ computational cost and communication size (as for the implementation detail, see Appendix A.)

2.4 GLOUDS

The graph level order unary degree sequence (GLOUDS) [14] is the succinct data structure of a DAG, which is a query-time efficient data structure that uses the space close to information-theoretic lower bound. GLOUDS regards a DAG as an integration of a spanning tree and "non-tree" edges that are not included in the tree, and introduces the idea of "shadow nodes", which are duplicates of non-tree nodes (nodes with incoming edges > 1) to virtually treat a graph as a tree, while it avoids unnecessary copies of nodes. When we transform a DAG into an equivalent tree, it is necessary to repeat copying of a subtree rooted from a non-tree node for all the incoming edges to the node, which causes exponential growth in total tree size. Since GLOUDS generates as many shadow nodes as the number of non-tree edges, it is considered to be efficient when there are not too many non-tree edges.

More precisely, GLOUDS consists of a trit (0,1,2) sequence B of length N and an auxiliary vector H, where N is the sum of the number of nodes and the number of edges + 1. The nodes in the DAG are numbered in level order (from top to bottom and left to right) and the root is numbered 0. The nodes are visited in level order during construction of GLOUDS. When each node is visited, 0 is stored in B, and all the children of the node are stored in left-to-right order. If a child is already observed, a trit 2 is stored in B, and 1 otherwise. The root node is considered as a child of an unshown supernode, and hence 1 is stored in B as the first element (i.e., $B[0] = 1$). H memorizes numbers of shadow nodes in the order in which they appear in B as 2. For the case of the DAG in Figure 1, after storing $B[0] = 1$, $B[1] = 0$ is stored when the node "0" is visited. Since the node "1" and "2" are the children of the node "0" and they are not observed, $B[2] = 1$

and $B[3] = 1$ are stored. Similarly, $B[4] = 0$, $B[5] = 1$ and $B[6] = 1$ are stored for the visit of the node "1", and $B[7] = 0$, $B[8] = 2$ and $B[9] = 1$ are stored for the visit of the node "2". Note that $B[8] = 2$ because the node "4", which is the left child of the node "2", is already observed. 4 is recorded in $H[0] = 4$. After visiting all the nodes, B and H are described as $B = 10110110210110210000$ and $H = [4, 7]$. Either 1 or 2 in vector B corresponds to any one of the nodes in the DAG, and 0 is regarded as a delimiter between groups of siblings. Note that 0 is also considered as a parent node of a right-neighbour group of siblings; therefore, the same node appears more than once in B.

Here, we define two operations on sequence B as:

Definition 1 *Operations on trit sequence B*

$\mathsf{Rank}_c(B, p)$*: returns the number of* $c \in \{0, 1, 2\}$ *in the prefix* $B[0, p)$ $(0 \leq p < N)$
$\mathsf{Select}_c(B, i)$*: returns the position of the* i*-th* $c \in \{0, 1, 2\}$ *in* B *(*i *starts from* 0*.)*

One can move from a position p in B that stores a trit 1 or 2 (a parent node), to another position p' (x-th child of the node) by carrying out the equation below.

$$p' = \begin{cases} \mathsf{Select}_0(B, \mathsf{Rank}_1(B, p)) + x & (if\ B[p] = 1) \\ \mathsf{Select}_0(B, H[\mathsf{Rank}_2(B, p)]) + x & (if\ B[p] = 2) \end{cases} \tag{1}$$

For simplicity, we let $\mathsf{SelRan}(B, p)$ be the first term of Eq. 1. The $\mathsf{SelRan}(B, p)$ computes a position in B of the left delimiter of $B[p]$'s children. Since the siblings are stored sequentially, one can specify the x-th child by adding an offset x to $\mathsf{SelRan}(B, p)$. Figure 1 has an example of SelRan. For example, let us consider the case of $p = 2$. $B[2]$ corresponds to the node "1". The children of the node "1" are "3" and "4", and they correspond to $B[5]$ and $B[6]$. $\mathsf{SelRan}(B, 2)$ returns 4 and $B[4] = 0$ is the left delimiter of them. We define a map of a position in B and a node id, such that $\mathsf{ID}(p)$ returns id of the node that corresponds to $B[p]$. For example, $\mathsf{ID}(2) = 1$ and $\mathsf{ID}(4) = 1$. Note that $\mathsf{ID}(p) = \mathsf{ID}(\mathsf{SelRan}(B, p))$. GLOUDS can be regarded as a generalization of the level order unary degree sequence (LOUDS) [18], which is a succinct data structure for ordered trees; hence, our protocol can be immediately applied to DT.

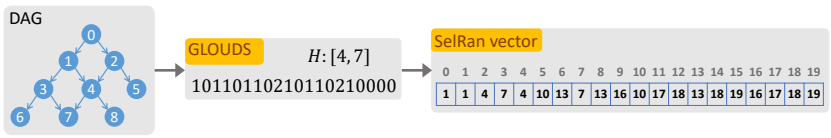


Fig. 1: Example of a BDG, corresponding GLOUDS and SelRan vector.

2.5 BDG and Efficient Design Principle of a Look-up Vector by GLOUDS

BDG consists of a rooted DAG and a set of split functions $\{\mathsf{Split}_i\}_{i=0}^{m-1}$, where m is the number of internal nodes. Given an attribute $\mathcal{X} \in \mathbb{Z}$, a split function that is assigned to an internal node performs a greater than operation: $\mathsf{Split}_i(\mathcal{X}) = \langle t_i < \mathcal{X} \rangle$ to choose either a right or left child.

BDG in our protocol is mainly represented by a look-up vector $\boldsymbol{v}$ of length N (also referred to as the SelRan vector), and a vector of ciphertexts $[\![\boldsymbol{o}]\!]$ that encrypts an offset vector $\boldsymbol{o}$ of length N. $\boldsymbol{v}$ represents the DAG, and $\boldsymbol{o}$ represents outputs of all the split functions taking a query (a set of attributes). Both $\boldsymbol{v}$ and $[\![\boldsymbol{o}]\!]$ are held by the party holding the BDG, and the other party traverses the BDG by obliviously referring to those vectors. $\boldsymbol{v}$ and $\boldsymbol{o}$ are described as:

$$\boldsymbol{v}[i] = \begin{cases} \mathsf{SelRan}(B, i) & (if\ B[i] \neq 0) \\ i & (else) \end{cases} \text{ and } \boldsymbol{o}[i] = \begin{cases} \langle \boldsymbol{\theta}[i] < \mathcal{X}_i \rangle + 1 & (if\ \boldsymbol{\tau}[i] = \mathsf{I}) \\ 0 & (else) \end{cases},$$

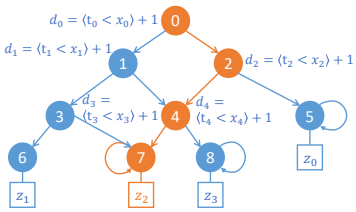
where $\mathcal{X}_i$ is a user's attribute for the split function that is associated with node $\mathsf{ID}(i)$, $\boldsymbol{\tau}$ is a **type vector** storing the types of each position and $\boldsymbol{\theta}$ is a **threshold vector**. $\boldsymbol{\tau}[i] = \mathsf{L}$(eaf) if node $\mathsf{ID}(i)$ is a leaf, $\boldsymbol{\tau}[i] = \mathsf{Z}$(ero) if node $\mathsf{ID}(i)$ is not a leaf and $B[i] = 0$. $\boldsymbol{\tau}[i] = \mathsf{I}$(nternal) otherwise. $\boldsymbol{\theta}[i]$ is a threshold of a split function that is associated with node $\mathsf{ID}(i)$ when $\boldsymbol{\tau}[i] = \mathsf{I}$. $\boldsymbol{\theta}[i]$ is set to empty otherwise. $\boldsymbol{v}[p]$ returns the position of the left delimiter of node $\mathsf{ID}(p)$'s children and $\boldsymbol{o}[p]$ returns the choice of a child. Therefore, one can compute the position of next node in B by:

$$p' = \boldsymbol{v}[p] + \boldsymbol{o}[p].$$

Note that the outputs of split functions include both parties' privacy; hence, the two parties need to jointly compute $[\![\boldsymbol{o}]\!]$ without revealing their private parameters. We will describe how this is accomplished in Subsection 3.4. $\boldsymbol{v}$ and $\boldsymbol{o}$ allow *self-loop* at positions $\{i \mid B[i] = 0, 0 \leq i < N\}$ by setting $\boldsymbol{v}[i] = i$ and $\boldsymbol{o}[i] = 0$. If one reaches such position i and node $\mathsf{ID}(i)$ is a leaf, one can stay on the same leaf to conceal the path length from the root toward each leaf. The self-loop at a non-leaf node can avoid incorrect traversal. The party holding BDG also prepares **label vector** $\boldsymbol{z}$. $\boldsymbol{z}[i]$ is set to a class label associated with node $\mathsf{ID}(i)$ if $\boldsymbol{\tau}[i] = \mathsf{L}$ and $B[i] = 0$. Otherwise, $\boldsymbol{z}[i]$ is a random value within the possible range of class labels.

Figure 2 has an example of these data structures that represent a BDG. The nodes and edges that are colored in orange show an example of a traversal from the root node to the node 7 when $\langle t_0 < x_0 \rangle = 1$, $\langle t_2 < x_2 \rangle = 0$ and $\langle t_4 < x_4 \rangle = 0$. The corresponding elements in the table in Figure 2 are also colored in orange. The traversal starts by referring to $\boldsymbol{v}[0] = 1$ to know the next position is $\boldsymbol{v}[0] + d_0 = 3$. Similarly, one can know the next position by $\boldsymbol{v}[3] + d_2 = 8$, and visits the node 7 by $\boldsymbol{v}[8] + d_4 = 14$. Finally, one reaches the

position $\boldsymbol{v}[14] + 0 = 18$ where a self-loop is allowed, and stays at the position while computing $\boldsymbol{v}[18] + 0 = 18$.



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ID	0	0	1	2	1	3	4	2	4	5	3	6	7	4	7	8	5	6	7	8
B	1	0	1	1	0	1	1	0	2	1	0	1	1	0	2	1	0	0	0	0
$\boldsymbol{v}$	1	1	4	7	4	10	13	7	13	16	10	17	18	13	18	19	16	17	18	19
$\boldsymbol{z}$	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	z_0	z_1	z_2	z_3
$\boldsymbol{\tau}$	I	Z	I	I	Z	I	I	Z	I	L	Z	L	L	Z	L	L	L	L	L	L
$\boldsymbol{\theta}$	t_0		t_1	t_2		t_3	t_4		t_4											
$\boldsymbol{o}$	d_0	0	d_1	d_2	0	d_3	d_4	0	d_4	0	0	0	0	0	0	0	0	0	0	0

Fig. 2: Example of representation of BDG by data structures described in Subsection 2.5. The vectors at the bottom of the figure represent the BDG on the left side of the figure. r_i is a random value.

It is possible to design more space-efficient SelRan vector and auxiliary vectors. Due to the space limitation, we will describe how we design such vectors in Subsection 3.5.

2.6 Security definition

Our security definition follows the ideal/real simulation paradigm [8], which is a mental experiment that proves security by comparing a real execution and ideal execution, which is secure by definition. In an ideal execution, an incorruptible trusted party participates in the protocol in addition to the two parties. The trusted party computes the desired function and distributes the results to the participants receiving inputs from the participants through a perfectly private channel.

We formulate the definition of security in the presence of a semi-honest adversary. We denote the view of the server during an execution of a protocol π as $\mathsf{view}^{\pi}_{S}(x, y)$, where x is the input from the user and y is the input from the server. Similarly, the view of the user is denoted as $\mathsf{view}^{\pi}_{U}(x, y)$. Let $\mathsf{output}^{\pi}_{S}(x, y), \mathsf{output}^{\pi}_{U}(x, y)$, be the outputs of the server and the user, respectively. The security of a protocol π, which computes functionality $f = (f_S, f_U)$ in the presence of a semi-honest adversary, is defined as follows [8]:

Definition 2 *Protocol π is secure if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that*

$$\{(S_1(x, f_S(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{(\mathsf{view}^{\pi}_{S}(x, y), \mathsf{output}^{\pi}(x, y))\}_{x,y\in\{0,1\}^*}, \tag{2}$$

$$\{(S_2(y, f_U(x, y)), f(x, y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{(\mathsf{view}^{\pi}_{U}(x, y), \mathsf{output}^{\pi}(x, y))\}_{x,y\in\{0,1\}^*}. \tag{3}$$

When the functionality f is deterministic, we can use the following simpler definition:

Definition 3 *Protocol π is secure if correctness is satisfied as follows*

$$\{\mathsf{output}^{\pi}(x,y)\}_{x,y\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \{f(x,y)\}_{x,y\in\{0,1\}^*}, \tag{4}$$

and if there exist S_1 and S_2 such that

$$\{S_1(x, f_S(x,y))\}_{x,y\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}^{\pi}_S(x,y)\}_{x,y\in\{0,1\}^*}, \tag{5}$$

$$\{S_2(y, f_U(x,y))\}_{x,y\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}^{\pi}_U(x,y)\}_{x,y\in\{0,1\}^*}\ . \tag{6}$$

3 Method

3.1 Problem Setting

We assumed a user had a private attribute vector $\boldsymbol{x}$, and a server had a private BDG T, in our protocol. Both $\boldsymbol{x}$ and T must be concealed from the other party. The user and the server participate in the two-party secure BDG evaluation protocol. The user only obtains an output of BDG $T(\boldsymbol{x})$, while he/she gains no information about the server's private information except for $T(\boldsymbol{x})$. The server obtains nothing. We assumed the user and the server shared three kinds of information: (1) length of the SelRan vector, (2) height of the BDG, and (3) length of the attribute vector. We considered a standard adversarial model in this work: a semi-honest model [8], in which even a corrupted party adheres to the specifications of a protocol.

3.2 Overview of Our Protocol

Our protocol is composed of two phases: a comparison and an evaluation phase.

Comparison phase: Construct offset vector The server eventually constructs and stores the encrypted offset vector without decrypting user inputs in this phase. The user and the server securely calculate split functions associated with nodes to achieve this purpose. The server stores all the decisions on which branch will be selected as ciphertexts. We used a secure comparison protocol to calculate split functions.

Evaluation phase: Compute class label on BDG Two parties descend from the root to a leaf in the evaluation phase by recursively referring to the SelRan vector and $[\![\boldsymbol{o}]\!]$ constructed in the comparison phase. After a leaf is reached, the user retrieves a label associated with the leaf from the label vector $\boldsymbol{z}$.

We will first describe several secure two party protocols that will be building blocks in Subsection 3.3, and then explain how to construct the comparison phase and evaluation phase in detail in Subsection 3.4. Our protocol can be seen as a sequential composition of the two protocols, Comparison Phase and Evaluation Phase. Therefore, security of our protocol is obvious if the underlying two protocols are secure.

3.3 Building Blocks

Comparison Protocol A two-party secure comparison protocol that securely computes $\langle x < y \rangle$, is required to calculate split functions in Comparison phase. We used a variant [34] of the DGK comparison protocol [12] in our implementation, which is based on additively homomorphic encryption.

While problem settings of comparison protocols vary, we assumed the following setting: a user and a server had a plaintext input x and y. Only the server acquired the encrypted comparison result $[\![\langle x < y \rangle]\!]$. Since we simply used the protocol and did not modify it, we will not go into details about the specification of the comparison protocol here. When x and y are ℓ bit integers, the time complexity and communication of both the user and the server are $O(\ell)$ in the DGK comparison protocol.

Recursive Oblivious Transfer The user recursively accesses the server's SelRan vector $\boldsymbol{v}$, in the evaluation phase, i.e., the user repeats querying an element of $\boldsymbol{v}$ and sets the next query depending on the query result. Not only queries but also intermediate results sent from the server need to be hidden to protect private information for both parties. We used a known secure two-party protocol called recursive oblivious transfer (ROT) [28, 31] for this problem.

Assuming a server has a plaintext vector $\boldsymbol{v}$ of length N and a user specifies a query p_0, ROT ensures that the user obtains $\boldsymbol{v}[\boldsymbol{v}[\ldots \boldsymbol{v}[p_0] \ldots]]$ and the server obtains nothing after an arbitrary number of iterations. ROT consists of σ steps, where σ is a common parameter between the user and the server. Except for the initial and the last steps, rest of the steps repeat the same protocol. The initial step computes the next position starting from the initial position p_0 specified by the user. At the end of the initial step, the user and the server gain shares of the next position $p_1 = \boldsymbol{v}[p_0]$. The k-th step ($k = 1, \ldots, \sigma - 2$) updates the position using the shares of the k-th position p_k. i.e., the user and the server gain shares of the next position $p_{k+1} = \boldsymbol{v}[p_k]$, p'_{k+1} and r_{k+1} where $r_{k+1} \in_R \mathbb{Z}$ is a random value. In the last step, the final value $\boldsymbol{v}[p_\sigma]$ is not divided into shares and only the user knows the value.

For convenience, we denote $(p'_{k+1}, r_{k+1}) \leftarrow \mathsf{ROT}(p'_k, r_k, \boldsymbol{v})$ for the k-th step of ROT, which takes shares of a query p_k (i.e., p'_k, r_k), and a vector $\boldsymbol{v}$ as inputs, and outputs p'_{k+1} to the user and r_{k+1} to the server. The initial step can also be represented by this notation by setting 0 to r_0. We describes the detailed procedure of ROT in Appendix B for self-containment. The time complexity and communication size of one round in ROT is $O(N)$ on both the user and the server sides due to the cost of OT.

eROT The main goal of eROT is recursive references to the offset vector $\boldsymbol{o}$ when it is encrypted. Specifying a query p_0, the user obtains $\boldsymbol{o}[\boldsymbol{o}[\ldots \boldsymbol{o}[p_0] \ldots]]$ after an arbitrary number of iterations, and the server obtains nothing. To achieve this goal, we assumed a server had an $N \times \lambda$ ciphertext matrix $\boldsymbol{\Omega}$, instead of $[\![\boldsymbol{o}]\!]$. Each row $\boldsymbol{\Omega}[i]$ is meant to represent $\boldsymbol{o}[i]$. More concretely, $\boldsymbol{\Omega}[i]$ is a vector, each

of whose elements is an encrypted bit of $\mathsf{UNARY}_\lambda(\boldsymbol{o}[i])$. For example, $\boldsymbol{o}[i] = 0$ is represented by $\boldsymbol{\Omega}[i][0] = [\![1]\!]$, $\boldsymbol{\Omega}[i][1] = [\![0]\!]$ and $\boldsymbol{\Omega}[i][2] = [\![0]\!]$ when $\lambda = 3$.

Initial (0-th) step:

1. The server generates a random value $r_1 \in_R \mathbb{Z}$, and prepares a vector $\boldsymbol{u}'$ whose i-th element is masked by r_1: $\boldsymbol{u}'[i] = \bigoplus_{j=0}^{\lambda-1}(\boldsymbol{\Omega}[i][j] \otimes (j + r_1)_{\mathsf{mod}\ N})$ (namely, $\boldsymbol{u}[i] = [\![\boldsymbol{o}[i]]\!], \boldsymbol{u}'[i] = [\![(\boldsymbol{o}[i] + r_1)_{\mathsf{mod}\ N}]\!]$.) The server stores r_1.
2. The user (chooser) and the server (sender) engage in $OT_1^N(p_0, \boldsymbol{u}')$. The user obtains $(p_1 + r_1)_{\mathsf{mod}\ N}$ decrypting $\boldsymbol{u}'[p_0] = [\![(p_1 + r_1)_{\mathsf{mod}\ N}]\!]$.

k-th ($k = 1, \ldots, \sigma - 2$) step:
The user holds $p'_k = (p_k + r_k)_{\mathsf{mod}\ N}$ and the server holds r_k.

1. The server generates a random value $r_{k+1} \in_R \mathbb{Z}$. Then, the server prepares a vector $\boldsymbol{u}'$ whose i-th element is masked by r_{k+1}: $\boldsymbol{u}'[i] = \bigoplus_{j=0}^{\lambda-1}(\boldsymbol{\Omega}[i][j] \otimes (j + r_{k+1})_{\mathsf{mod}\ N})$. The server stores r_{k+1}.
2. The server rotates $\boldsymbol{u}'$ by r_k elements to obtain $\hat{\boldsymbol{u}}'$.
3. The user (chooser) and the server (sender) engage in $OT_1^N(p'_k, \hat{\boldsymbol{u}}')$, and the user obtains $(p_{k+1} + r_{k+1})_{\mathsf{mod}\ N}$ decrypting $\boldsymbol{u}'[p_k] = [\![(p_{k+1} + r_{k+1})_{\mathsf{mod}\ N}]\!]$.

Last step:
The server does not mask $\boldsymbol{u}[i](= \bigoplus_{j=0}^{\lambda-1}(\boldsymbol{\Omega}[i][j] \otimes j_{\mathsf{mod}\ N}))$ in the last step to send a true value to the user.

1. The server rotates $\boldsymbol{u}$ by $r_{\sigma-1}$ elements to obtain $\hat{\boldsymbol{u}}$.
2. The user (chooser) and the server (sender) engage in $OT_1^N(p'_{\sigma-1}, \hat{\boldsymbol{u}})$, and the user obtains $\boldsymbol{u}[p_{\sigma-1}] = [\![p_\sigma]\!]$. The user obtains $\boldsymbol{o}[\boldsymbol{o}[\ldots\boldsymbol{o}[p_0]\ldots]] = p_\sigma$ by decrypting $[\![p_\sigma]\!]$.

For convenience, we denote $(p'_{k+1}, r_{k+1}) \leftarrow \mathsf{eROT}(p'_k, r_k, \boldsymbol{\Omega})$ for the k-th step of eROT, which takes shares of a query p_k (i.e., p'_k, r_k) and a matrix $\boldsymbol{\Omega}$ as inputs, and outputs p'_{k+1} to the user and r_{k+1} to the server. The initial step can also be represented by this notation setting from 0 to r_0.

Since the major part of the time complexity is the inner product and OT, the time complexity on the server side in one round is $O(N\lambda)$. The time complexity on the user side is $O(N)$ per iteration. The communication size per iteration is $O(N)$ due to the communication size for OT.

We state that the following security theorem is established for eROT.

Theorem 1. eROT *is secure in the semi-honest setting.*

Proof. **Correctness:** Each row of $\boldsymbol{\Omega}$ is an unary representation of a value, and hence conducting $\boldsymbol{u}'[i] = \bigoplus_{j=0}^{\lambda-1}(\boldsymbol{\Omega}[i][j] \otimes (j+r)_{\mathsf{mod}\ N})$ correctly yields an encryption of $(p+r)_{\mathsf{mod}\ N}$, where p is the value stored at $\boldsymbol{\Omega}[i]$. Therefore, by performing the initial step of eROT, the two parties can obtain shares $(p_1 + r_1)_{\mathsf{mod}\ N}$ and r_1 of the true position p_1. In the k-th step, the user's input $p'_k = (p_k + r_k)_{\mathsf{mod}\ N}$ to OT is a share of the true position p_k, and the two parties can obtain a correct element by rotating $\boldsymbol{u}'$ by r_k before conducting OT. After decrypting the

encrypted value obtained by OT, the user knows the share of the next position p_{k+1}. In the last step, the server does not mask $\boldsymbol{u}'$ and, therefore by induction it holds that the protocol correctly outputs $\boldsymbol{o}[\boldsymbol{o}[\ldots\boldsymbol{o}[p_0]\ldots]]$ to the user.
Security: All the messages are exchanged by OT. Considering that secure OT is used, it is guaranteed that no information of the user is leaked to the server. Security against a semi-honest user is established by secret sharing. Shares of intermediate results received by the user are indistinguishable from uniformly distributed random values due to the property of modular addition. Thus, a semi-honest user cannot acquire any information from intermediate results. □

3.4 Secure BDG Evaluation Using GLOUDS and AHE

Comparison Phase The server and the user construct an encrypted matrix $\boldsymbol{\Omega}$, which corresponds to offset vector $\boldsymbol{o}$. The comparison phase ensures that no information from the server or the user will be disclosed, other than the number of comparisons. The following describes how we constructed $\boldsymbol{\Omega}$. The detailed algorithm is provided in Algorithm 1.

Construction of $\boldsymbol{F}$: Each split function is associated with one of positions $\{j \mid \boldsymbol{\tau}[j] = \mathsf{I} \wedge 0 \leq j < N\}$. The server and the user conduct a secure comparison protocol in Step (1) of Algorithm 1 to securely compute all the comparison results between attributes and thresholds. The server finally constructs a flag matrix $\boldsymbol{F}$. We do not need to compute $\boldsymbol{F}[j]$ if $\boldsymbol{\tau}[j] \neq \mathsf{I}$.

Construction of $\boldsymbol{W}$: The server constructs a matrix $\boldsymbol{W}$, each of whose rows is an encrypted 2-bit unary vector that represents an output of a split function.

Construction of $\boldsymbol{\Omega}$: The server constructs $\boldsymbol{\Omega}$ in Step (3) based on $\boldsymbol{W}$ and $\boldsymbol{\tau}$. To make an encryption of $\mathsf{UNARY}_3(\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle + 1)$, we set $\boldsymbol{\Omega}[j][0] = [\![0]\!]$ if $\boldsymbol{\tau}[j] = \mathsf{I}$. If $\boldsymbol{\tau}[j] \neq \mathsf{I}$, $\boldsymbol{\Omega}[j]$ stores $\mathsf{UNARY}_3(0)$. Note that the lengths of rows of $\boldsymbol{\Omega}$ can be reduced to 1 when $\tau[j] \neq \mathsf{I}$ (because the offset is 0). This is because the server knows the offsets that do not rely on any user information and can minimize the bit length. To use the reduced form of the offset matrix, we modify the inner product in each step of eROT to $\bigoplus_{j=0}^{u_j}(\boldsymbol{\Omega}[i][j] \cdot (j+r)_{\mathsf{mod}\ N})$, where $u_j \in \{1, 3\}$ is the length of the row. As a result, we can also reduce the time complexity to $O(N)$, where N is the length of GLOUDS.

Also note that the calculation of an offset can be omitted when the node is a shadow node. A new type of position for shadow nodes should be defined to do that to distinguish them from other internal nodes.

We state that the following security theorem is established for Algorithm 1.

Theorem 2. *Algorithm 1 is secure in the semi-honest setting.*

Proof. **Correctness:** When an attribute $\mathcal{X}_j$ is less than an threshold $\boldsymbol{\theta}[j]$, $(\boldsymbol{W}[j][1], \boldsymbol{W}[j][2])$ becomes $([\![1]\!], [\![0]\!])$, otherwise $([\![0]\!], [\![1]\!])$ assuming the correctness of the underlying secure comparison protocol. Therefore, all the rows of $\boldsymbol{\Omega}$ satisfy the condition that they represent offsets in encrypted unary vectors.

Algorithm 1 Detailed description of comparison phase

- Public inputs: length of GLOUDS (N); height d; length of attribute vector (n)
- Private input of server: threshold vector $\boldsymbol{\theta}$; type vector $\boldsymbol{\tau}$
- Private input of user: attribute vector $\boldsymbol{x}$

Step (1): Server and User conduct comparison protocol coorperatively and Server obtains $[\![\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle]\!]$. $\mathcal{X}_j$ is an element of attribute vector corresponding to the position j. Server constructs a flag matrix $\boldsymbol{F}$.

for $j \in \{0, \ldots, N-1\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{F}[j][0] \leftarrow [\![1]\!]$, $\boldsymbol{F}[j][1] \leftarrow [\![\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle]\!]$, $\boldsymbol{F}[j][2] \leftarrow [\![0]\!]$

Step (2): Server constructs $\boldsymbol{W}$ from $\boldsymbol{F}$

for $j \in \{0, \ldots, N-1\}, k \in \{1, 2\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{W}[j][k] \leftarrow \boldsymbol{F}[j][k-1] \oplus (\ominus \boldsymbol{F}[j][k])$ ▷ $[\![\mathsf{UNARY}_2(\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle)]\!]$

Step (3): Server constructs an encrypted offset matrix $\boldsymbol{\Omega}$ based on $\boldsymbol{W}$.

for $j \in \{0, \ldots, N-1\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{Z}$ or $\boldsymbol{\tau}[j] = \mathsf{L}$ **then**
    $\boldsymbol{\Omega}[j][0] \leftarrow [\![1]\!]$, $\boldsymbol{\Omega}[j][1] \leftarrow [\![0]\!]$, $\boldsymbol{\Omega}[j][2] \leftarrow [\![0]\!]$ ▷ $[\![\mathsf{UNARY}_3(0)]\!]$
  else if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{\Omega}[j][0] \leftarrow [\![0]\!]$, $\boldsymbol{\Omega}[j][1] \leftarrow \boldsymbol{W}[j][1]$, $\boldsymbol{\Omega}[j][2] \leftarrow \boldsymbol{W}[j][2]$
    ▷ $[\![\mathsf{UNARY}_3(\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle + 1)]\!]$

Security: We have assumed that the underlying secure comparison protocol is secure in the semi-honest setting. Since the procedures after the secure comparison protocol only require server side operations on ciphertext, the security of the comparison phase is guaranteed by the security of the secure comparison protocol.

Evaluation Phase This section describes the evaluation phase in which the participants securely descend a BDG using ROT and eROT.

We need to recursively refer to $\boldsymbol{v}$ and $\boldsymbol{\Omega}$ by starting from an initial position $p_0 = 0$ to move from the root to the leaf. The next position p_{i+1}, given a starting position p_i on GLOUDS, which corresponds to the child, is calculated by adding the p_i-th elements of a SelRan vector $\boldsymbol{v}$, and an encrypted offset matrix $\boldsymbol{\Omega}$. The next iteration will be executed after the next position p_{i+1} is set.

The private information of both parties must simultaneously be protected. There are two main security requirements: (1) the server should not know the positions specified by the user or the results of the protocol, and (2) $\boldsymbol{v}$ and $\boldsymbol{\Omega}$ held by the server should be concealed from the user. We used ROT and eROT to recursively refer to $\boldsymbol{v}$ and $\boldsymbol{\Omega}$ concealing private information. Algorithm 2 describes the details of the evaluation phase satisfies the previously explained functionality and security requirements.

Algorithm 2 Detailed description of evaluation phase

- Public input: length of GLOUDS (N); height d
- Private input of server: SelRan vector $\boldsymbol{v}$, encrypted offset matrix $\boldsymbol{\Omega}$, label vector $\boldsymbol{z}$

Step (1): Initialization

$\mathcal{B}$ conducts: $r_1 \leftarrow 0$, $r_2 \leftarrow 0$, $r' \leftarrow r_1 + r_2$

$\mathcal{A}$ conducts: $p'_0 \leftarrow 0$

Step (2): Update the position in GLOUDS by iterating ROT, eROT.

for $k = 0$ **to** $d - 1$ **do**

Server and User engage in ROT and eROT.

$(\beta'_{k+1}, r_1) \leftarrow \mathsf{ROT}(p'_k, r', \boldsymbol{v})$, $(\omega'_{k+1}, r_2) \leftarrow \mathsf{eROT}(p'_k, r', \boldsymbol{\Omega})$

$\mathcal{B}$ conducts: $r' \leftarrow (r_1 + r_2)_{\mathsf{mod}\ N}$

$\mathcal{A}$ conducts: $p'_{k+1} \leftarrow (\beta'_{k+1} + \omega'_{k+1})_{\mathsf{mod}\ N}$

Step (3): Get the output of BDG $T(\boldsymbol{x})$ from $\boldsymbol{z}$ using $OT_1^N(p'_d, \boldsymbol{z})$.

First, the server and the user start initialization in Step (1). The server sets $r_1 = r_2 = r' = 0$. The r_1, r_2 store random values used in the previous iterations of ROT and eROT, and r' is sum of r_1 and r_2 modulo N. The user sets the initial position $p'_0 = 0$. The two parties engage in ROT and eROT in Step (2) to update the position on GLOUDS by recursively referring to $\boldsymbol{v}$ and $\boldsymbol{\Omega}$. The (β'_{k+1}, r_1) and (ω'_{k+1}, r_2) correspond to random shares of $\boldsymbol{v}[p_{k+1}]$ and an offset $\boldsymbol{o}[p_{k+1}]$, which can be recovered by using the server's random values r_1, r_2, and rotating look-up vectors (without knowing these values due to the security of OT). Since the position of the x-th child is determined by $\mathsf{SelRan}(B, p) + x$, the next query is $p'_{k+1} = (\beta'_{k+1} + \omega'_{k+1})_{\mathsf{mod}\ N}$. This iteration is conducted d times regardless of the depth of a leaf, which should be reached. It should be noted that a position is fixed once it is reached at a position in B with trit 0, which ensures that the last position is in the position that corresponds to the leaf due to the definition of $\boldsymbol{v}$ and $\boldsymbol{\Omega}$. Finally, the user obtains the output of the BDG $T(\boldsymbol{x})$ from $\boldsymbol{z}$ using $OT_1^N(p'_d, \boldsymbol{z})$ in Step (3).

We state that the following security theorem is established for Algorithm 2.

Theorem 3. *Algorithm 2 is secure in the semi-honest setting.*

Proof. **Correctness:** Due to the way the look-up vector $\boldsymbol{v}$ and $\boldsymbol{\Omega}$ are constructed, it is obvious that the evaluation phase can correctly compute $\boldsymbol{v}[p_k] + \boldsymbol{o}[p_k]$ in k-th step, if $\boldsymbol{v}$ and $\boldsymbol{\Omega}$ are not randomized. $\boldsymbol{v}$ is randomized by r_1 and $\boldsymbol{\Omega}$ (i.e., $\boldsymbol{o}$) is randomized by r_2. Since following equation is established by considering the property of modular addition,

$$\begin{aligned}\{p'_{k+1} - r'\}_{\mathsf{mod}\ N} &= \{(\beta'_{k+1} - r_1) + (\omega'_{k+1} - r_2)\}_{\mathsf{mod}\ N} \\ &= \boldsymbol{v}[p_k] + \boldsymbol{o}[p_k] = p_{k+1},\end{aligned}$$

$p'_{k+1} = (\beta'_{k+1} + \omega'_{k+1})_{\mathsf{mod}\ N}$ and $r' = (r_1 + r_2)_{\mathsf{mod}\ N}$ are the shares of p_{k+1}, hence one can obtain shares of $\boldsymbol{v}[p_{k+1}]$ and $\boldsymbol{o}[p_{k+1}]$ by conducting ROT and eROT with the same arguments p'_{k+1} and r'. The label corresponding to the leaf is obtained by OT in Step (3). Therefore, by induction it holds that the evaluation phase correctly outputs $T(\boldsymbol{x})$.

Security: Since the functionality of the evaluation phase is deterministic, we can state Definition 3 for the analysis. In the following, we prove Eqs. (5) and (6) are established. We assume to use OT protocol described in the Appendix A.

Security against semi-honest server:

Algorithm S_1 can be constructed as follows:

1. It outputs a dummy query vector consisting of N ciphertexts $[\![\mathbf{0}]\!]$ to simulate each step of ROT and eROT. Note that in Algorithm 2, the user sends only one ciphertext vector that represents r', and the server uses the same ciphertext for ROT and eROT. Therefore it is sufficient for S_1 to generate N ciphertexts.
2. It repeats Step 1 for d times.
3. It output a dummy query vector consisting of N ciphertexts $[\![\mathbf{0}]\!]$ to simulate OT in the final step.

Since all the messages received by the server are ciphertexts, the view of the server and the outputs of the simulator are computationally indistinguishable by a standard hybrid argument due to the semantic security of the underlying public-key encryption scheme. Thus, Eq. (5) is established.

Security against semi-honest user:

In the evaluation phase, two types of OT protocol is used. OT_A is used in ROT and Step (3) in Algorithm 2 and OT_B is used in eROT. Note that OT_A sends only a single ciphertext to the user while OT_B sends N ciphertexts to the user. See Appendix A for more details about OT_A and OT_B. Algorithm S_2 can be constructed as follows:

1. S_2 simulates 0-th round of Step (2) in Algorithm 2. It generates a dummy intermediate of ROT $[\![b]\!]$, where $b \in_R [0, N)$, and a dummy intermediate of eROT $[\![\boldsymbol{v}]\!] = [\![(w_0, \ldots, w_{N-1})]\!]$ where $w_0 \in_R [0, N)$ and $w_1, \ldots, w_{N-1} \in_R \mathbb{Z}_p$. It stores $p_1 = (b + w_0)_{\mathsf{mod}\ N}$
2. S_2 simulates k-th round of Step (2) in Algorithm 2. It generates a dummy intermediate of ROT $[\![b]\!]$, where $b \in_R [0, N)$, and a dummy intermediate of eROT $[\![\boldsymbol{v}]\!] = [\![(w_0, \ldots, w_{N-1})]\!]$ where $w_{p_k} \in_R [0, N)$ and $\{w_i \mid i \neq p_k \wedge i \in \{0, \ldots, N-1\}\} \in_R \mathbb{Z}_p$. It stores $p_{k+1} = (b + w_{p_k})_{\mathsf{mod}\ N}$.
3. It repeats Step 2 for $d - 1$ times.
4. S_2 generates $[\![T(\boldsymbol{x})]\!]$ to simulate Step (3) in Algorithm 2.

The additive secret share β' received by the user in each iteration of ROT is uniformly random over $[0, N)$ due to the property of modular addition. For the intermediate of eROT in k-th round of Step (2) in Algorithm 2, the p'_k-th element ω' is uniformly random over $[0, N)$ and the other elements are uniformly

random over $\mathbb{Z}_p$. $p'_{k+1} = (\beta' + \omega')_{\mathsf{mod}\ N}$ and p'_{k+1}-th element is uniformly random over $[0, N)$ in (k+1)-th round. Therefore, the distributions of the outputs of S_2 and the view of the user are identical. Thus, Eq. (6) is established. □

3.5 Reducing lengths of look-up vectors

The original GLOUDS supports a traversal both from a parent to its child and from a child to its parent; however, in BDG, only one-way traversal, i.e., from a parent to its child is necessary. Therefore, the SelRan vector includes redundant information. This section describes how we designed the SelRan vector with minimum required length for BDG evaluation. In fact, we could reduce the length of the SelRan vector to the number of edges from the sum of the number of edges and the number of nodes. This resulted in reduced computational and communication costs in the evaluation phase, and these costs could be reduced by half if a DAG of BDG is a tree.

The basic idea underlying our approach was to delete elements that did not need to be visited. As you might have already noted, $B[p] = 0$ is not visited during the traversal except for leaves, and when it corresponds to a leaf, the move to trit 0 from trit 1 or 2 is redundant. Since we have as many such trits in B as the number of all nodes in DAG, we can reduce such elements from the SelRan vector. We constructed the look-up vector $\boldsymbol{v}$ in the following way. First, we constructed two index data structure traversing nodes in level order as in the construction of GLOUDS. Note that shadow nodes were not revisited. Child nodes of visited nodes were stored in a vector $\boldsymbol{u}$, while traversing. We simultaneously constructed a dictionary $\boldsymbol{s}$ that mapped a visited node to the index of $\boldsymbol{u}$ in which its children started to be lined up. The look-up vector $\boldsymbol{v}$ was constructed by referring to $\boldsymbol{u}$ and $\boldsymbol{s}$. The i-th element of $\boldsymbol{v}[i]$ should be the position where the children of $\boldsymbol{u}[i]$ start to line up, viz., $\boldsymbol{v}[i] = \boldsymbol{s}[\boldsymbol{u}[i]]$. Exceptionally, when the node associated with the i-th element is a leaf, $\boldsymbol{v}[i] = i$. The data structures to represent a BDG also needed to be modified. Thresholds were stored in the positions associated with internal nodes, and labels were stored in the positions corresponding to leaf nodes. There is an example of the modified data structures representing a BDG in Figure 3.

Since the modified data structures were built with the same design principle as that of the original, what was modified in the protocol was only the construction of the encrypted offset matrix. Specifically, each offset was decreased by 1. Therefore, the algorithm for the comparison phase was modified as Algorithm 3.

3.6 Complexity

This subsection discusses the asymptotic time complexity and communication complexity of our protocol. The majority of computational and communication costs in the comparison phase are due to the comparison protocol and the construction of an encrypted offset matrix. The time complexity of the comparison protocol on the whole is $O(\ell m)$ on the server side. It is $O(\ell(n + m))$ on the user side by considering the encryption of an attribute vector and decryption

Algorithm 3 Detailed description of modified comparison phase

– Public inputs: length of GLOUDS (N); height d; length of attribute vector (n)
– Private input of server: threshold vector $\boldsymbol{\theta}$; type vector $\boldsymbol{\tau}$
– Private input of user: attribute vector $\boldsymbol{x}$

Step (1): Server and User conduct comparison protocol coorperatively and Server obtains $[\![\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle]\!]$. $\mathcal{X}_j$ is an element of attribute vector corresponding to the position j. Server constructs a flag matrix $\boldsymbol{F}$.

for $j \in \{0, \ldots, N-1\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{F}[j][0] \leftarrow [\![1]\!]$, $\boldsymbol{F}[j][1] \leftarrow [\![\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle]\!]$, $\boldsymbol{F}[j][2] \leftarrow [\![0]\!]$

Step (2): Server constructs $\boldsymbol{W}$ from $\boldsymbol{F}$

for $j \in \{0, \ldots, N-1\}, k \in \{1, 2\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{W}[j][k] \leftarrow \boldsymbol{F}[j][k-1] \oplus (\ominus \boldsymbol{F}[j][k])$ ▷ Encryption of $\mathsf{UNARY}_2(\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle)$

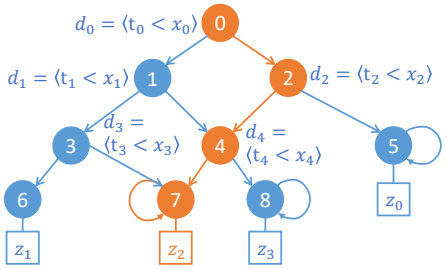
Step (3): Server construct an encrypted offset matrix $\boldsymbol{\Omega}$ based on $\boldsymbol{W}$.

for $j \in \{0, \ldots, N-1\}$ **do**
  if $\boldsymbol{\tau}[j] = \mathsf{L}$ **then**
    $\boldsymbol{\Omega}[j] \leftarrow ([\![0]\!], [\![1]\!])$ ▷ Encryption of $\mathsf{UNARY}_2(0)$
  else if $\boldsymbol{\tau}[j] = \mathsf{I}$ **then**
    $\boldsymbol{\Omega}[j] \leftarrow (\boldsymbol{W}[j])$ ▷ Encryption of $\mathsf{UNARY}_2(\langle \boldsymbol{\theta}[j] < \mathcal{X}_j \rangle)$

of intermediate results. The construction of an encrypted offset matrix requires $O(N)$ computational cost as the computational cost is linear to the sum of the lengths of its rows. Therefore, the total time complexity of the comparison phase is $O(\ell m + N)$ on the server side and $O(\ell(n + m))$ on the user side. Since an encrypted offset matrix is constructed offline, all of the communication cost is required by the secure comparison protocol. The communication cost for the server is $O(\ell m)$ and that for the user is $O(\ell n + m)$ in the comparison phase. The time complexities of both the server and the user are $O(dN)$ for the evaluation phase. This is because ROT and eROT require $O(dN)$ computational cost. The communication cost is also $O(dN)$. We have summarized the time and communication complexity in Table 1.

4 Experiments

We evaluated the efficiency of our protocol with experiments under various settings. We implemented our protocol, which is secure against the semi-honest model using the C++ library of elliptic curve Elgamal encryption [24]. We used secp256k1 for the security parameters of lifted-Elgamal, which is secure at the 128-bit security level [9]. We used a standard desktop PC with a Xeon 3.40-GHz processor for the server and a standard desktop PC with a Xeon 2.40-GHz processor for the user (1 thread each). Both the server and the client were in the



i	0	1	2	3	4	5	6	7	8	9	10
ID	0	1	2	3	4	4	5	6	7	7	8
v	1	3	5	7	9	9	6	7	8	9	10
z	r_0	r_1	r_2	r_4	r_4	r_5	z_0	z_1	z_2	z_2	z_3
τ	I	I	I	I	I	I	L	L	L	L	L
θ	t_0	t_1	t_2	t_3	t_4	t_4					
o	d_0	d_1	d_2	d_3	d_4	d_4	0	0	0	0	0

Fig. 3: Example of construction of modified data structures representing BDG.

Table 1: Time complexity and Communication of each phase of our method. (S: Server; U: User) d is the height of a DAG, ℓ is the bit length of user's and server's inputs, m is the number of split functions, n is the number of nodes in a DAG and N is the length of B.

Phase	Time	Communication	# rounds
Comparison(S)	$O(\ell m + N)$	$O(\ell m)$	2
Comparison(U)	$O(\ell(n+m))$	$O(\ell n)$	
Evaluation(S)	$O(dN)$	$O(dN)$	$d+1$
Evaluation(U)	$O(dN)$	$O(dN)$	

same local area network (LAN) in our experiments. Note that the look-up tables were modified in the way described in Subsection 3.5 in the experiments.

4.1 Experiment on Simulated Dataset

First, we will present the results obtained from experiments on the simulated dataset. The dataset was composed of pairs of BDGs and attribute vectors. We varied the height d of the BDGs from 16 to 20 one by one, while fixing the number of nodes to 1110 (the number of comparisons m was 557.) and the lengths of attribute vectors n to 395. These parameters (except for d) were taken from Brickell et al. [6] to enable performance to be later compared in Section 4.3. Figures 4 plots the CPU time and communication size of the server and the user in our protocol. We observed that even when height d was 20, our protocol finished within a practical timeframe and communication size (27 s and 14 MB). We also confirmed that the CPU time and communication size of both the server and the user were linear to d, which is consistent with theoretical complexity.

4.2 Experiments in Various Latency Network Environments

We also confirmed that runtime overhead caused by network latency was not too large. We measured the runtimes of our protocol on the BDG with a depth of 20 that was included in our simulated datasets by varying the latency of the network by using the tc command of Linux. Figure 5 summarizes the results.

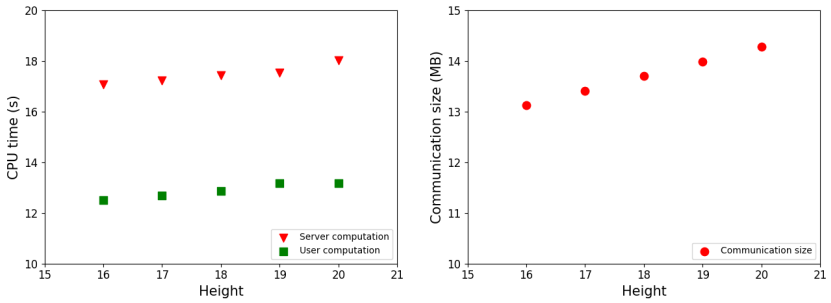


Fig. 4: CPU time (s) of server and user, and communication size (MB) on simulated datasets. We varied d from 16 to 20 while fixing other parameters ($m = 557, n = 95$, and $N = 1110$.)

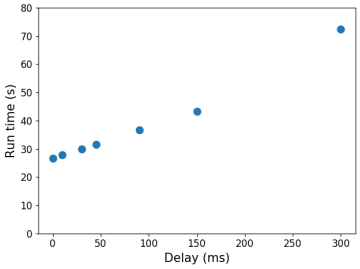


Fig. 5: Experimental results with BDG of $d = 20, N = 1110, m = 557$ and $n = 395$ in various latency network environments. We varied delay from 0 to 300 by using the tc command and measured runtime of the protocol.

Since our protocol requires $d + 3$ communication rounds, the increase in runtime is relatively large. However, even when network latency is 300ms, the runtime is about 1min. Additionally, the regional round trip times (RTTs) are less than 45 ms [33] (in these cases, the increase in runtime caused by network latency will be 6 s at most), and even for the trans-Pacific, the RTTs are less than 150 ms.

4.3 Comparison to Conventional Methods

Brickell et al. [6] proposed a $O(n + \ell N + d)$ time BP (BDG) evaluation method based on AHE and Garbled Circuit. [6] reported the performance of their protocol on a BDG (1107 nodes and 395 attributes) as 302 s in CPU time and 25 MB in communication size. Since the exact topology of the BDG they used was unshown, we conducted an experiment on BDGs that have the same number of nodes and attributes with various heights for fair comparison. The results revealed that our method maintained 11 times better performance in runtime (26 s) than that of [6] and required 1.8 times less communication size (14 MB) even when the DAG is as high as $d = 20$. Barni et al. proposed a privacy-preserving evalution method of LBP which is a generalization of BP. The time complexity of this method is $O(n+m\ell'+d)$ where $\ell'(> \ell)$ is bit length of threshold. Barni et al.'s [4] performance on the ECG dataset ($d = 4, m = 6$ and $n = 4$) was 6.8 s in computation (without network communication) and 0.1122 MB in communication size. The performance of our method with the same parameters was about 8.85 times better (0.768 s) than [4] in terms of computational cost, although our method incurred slightly more cost in communication (0.156 MB). Additionally, the security level in our experimental setting was higher than that of [4, 6]. (They conducted experiment at a 80 bit security.)

4.4 Detailed Comparison with Methods Specialized in DT

We also conducted experiments on DTs trained using several real datasets used by conventional methods [5, 30, 35]. Even compared to the methods specialized in DT, the experimental results showed that the performance of our method exceeds that of Bost et al.'s method [5] and is almost equivalent to those of Wu et al.'s and Tai et al.'s methods [30, 35].

The experimental results from Bost et al. [5] on the ECG dataset were 4.020 s in computation (without network communication) and 3.555 MB in communication size. Compared to Bost et al. [5], the runtime of our method was over 8 times better than the computational time (0.768 s) of Bost et al. [5], and required 21 times lower communication size (0.156 MB).

Table 2 summarizes the runtime and communication size of Tai et al. and Wu et al. [30, 35] and our protocol. Since the experimental environments of ours and Wu et al.'s [35] were different, the numbers for our method in Table 2 were adjusted by multiplying the ratio of CPU frequencies. The security level in our experiments was equivalent to that of Tai et al. and Wu et al. [30, 35]. Compared to Wu et al.'s method [35], our protocol had an advantage on deep decision trees.

Table 2: Experimenetal results on real datasets of [30, 35] and our method. d, m and n respectively stands for a depth of a decision tree (without super root in GLOUDS), the number of internal nodes and dimention of attributes. It should be noted that for resutls of [30, 35], we used the numbers shown in the paper [30, 35]. Their benchmarks are performed on 2.30 GHz CPU for a client and 2.60GHz CPU for a server.

Dataset	d	m	n	Method	Run time (s)	Comm. (MB)
Heart disease	3	5	13	Wu et al.	0.370	0.117
				Tai et al.	0.250	0.140
				Our method	0.474	0.166
Credit screening	4	5	15	Wu et al.	0.551	0.095
				Tai et al.	0.270	0.160
				Our method	0.491	0.186
Breast cancer	8	12	9	Wu et al.	0.545	0.206
				Tai et al.	0.340	0.170
				Our method	0.757	0.247
Housing	13	92	13	Wu et al.	4.08	1.91
				Tai et al.	1.98	0.850
				Our method	4.52	1.60
Spambase	17	58	57	Wu et al.	16.6	17.8
				Tai et al.	1.80	0.920
				Our method	3.74	1.54

When $d = 17, m = 58$, and $n = 57$, our method achieved about a 4 fold faster runtime. The performance of [35] slightly exceeded that of our method in several cases; however, the differences between the two methods in this experiment were less than 0.5 s in runtime (housing). This was because our protocol had better asymptotic complexities of computation and communication costs than those of [35]. Table 3 summarizes the complexities of [35] and our method. Wu et al.'s method [35] required exponential computational time and communication size on the server side. The complexity of our method, on the other hand, was worse than that of Tai et al. [30]. The computation and communication costs of Tai et al.'s method [30] were linear to the number of internal nodes and did not depend on the heights of trees, as listed in Table 3. However, the results summarized in Table 2 indicates that in practice, our method only incurred slightly more costs even for DTs, compared to the fastest secure method of DT evaluation, despite its extensibility to BDGs.

Evaluation of DT Equivalent to BDG The methods specialized in DTs can be used for BDGs by transforming a BDG into an equivalent DT, however, their computational costs increase along with the increase of redundant nodes and edges incurred by the transformation. Therefore, while the state-of-the-art method by Tai et al. [30] performed slightly better than our method for DT evaluation, its runtime became worse than our method's runtime when it is tested on complex BDGs.

Table 3: Time and communication complexities of conventional methods [30,35] and the proposed method. S: Server; U: User

	Time	Communication	# rounds
Wu et al. [35](S)	$O(\ell m + 2^d)$	$O(\ell m + 2^d)$	3
Wu et al. [35](U)	$O(\ell(n+m)+d)$	$O(\ell n + m)$	
Tai et al. [30](S)	$O(\ell m)$	$O(\ell m)$	2
Tai et al. [30](C)	$O(\ell(n+m))$	$O(\ell(n+m))$	
Proposed(S)	$O(\ell m + dN)$	$O(\ell m + dN)$	$d+3$
Proposed(U)	$O(\ell(n+m)+dN)$	$O(\ell n + m + dN)$	

Table 4: Computational cost of each public key operation.

	time (ms)
$\oplus$	1.40619
$\otimes$	22.3278
$\mathsf{Enc}_{\mathsf{pk}}$	284.127
$\mathsf{Dec}_{\mathsf{sk}}$	103.976

Fig. 6: Abstract structure of BDG used in the experiment on BDGs

Let m' be the number of nodes of the tree that is equivalent to such the BDG. Then, the numbers of operations required for Tai et al's method excluding those of secure comparison protocol, which is a common part with our method, are as follows; Addition: $4m'$, Multiplication: $3m'$, Encryption: m', Decryption (average): $m'/2$. Our method requires $(3d+5)N$ additions, $(4d+6)$ multiplications, $N(d+2)+d+5$ encryptions, $2(d+2)$ decryptions in amount. We measured computational costs of each operation in pre-experiment and summarized the results in Table 4.

For example, let us consider BDGs that have the diamond-shaped structures surrounded by arrows outlined in Figure 6. When the number of diamond is 6, d is 20, m is 250, N is 494 and hence m' is 15037, the number of the internal nodes of the equivalent DT is 15037, which is 60 times larger than original BDG. When the number of diamond is 6, we estimated the computational cost of Tai et al's method to be 6.15s (excluding 10.9s of DGK comparison protocol) and our method is estimated to be 4.09s

5 Conclusion

We proposed an efficient protocol for evaluating BDGs, which was designed by AHE and did not use heavy cryptographic primitives, such as fully homomorphic encryption. The protocol obliviously evaluated a look-up table that was constructed based on GLOUDS to achieve linear time and communication complexities. We also proposed a design principle for the look-up tables to further reduce the table size. The results obtained from the experiments indicated that the actual runtime and communication size were well concordant with theoretical complexities and that the runtime of our method was an order of magnitude

faster than that in the previous approaches [4, 6]. We also confirmed that our method was even faster for the DT evaluations compared to a previous approach that specialized in DT [35] when the tree was deep. Our method demonstrated a runtime in an experiment with BDGs that was faster than the state-of-the-art method of DT evaluation [30] that took advantage of the fact that a graph with information equivalent to that in a tree was much more compact than the tree. These results confirmed the efficiency of our method, and we also hope that it will contribute to secure utilization of valuable classifiers that aggregate knowledge extracted from abundant data resources. Another remarkable feature of our protocol is that it directly simulates step-by-step graph traversal, whereas other efficient methods [30, 35] reformulate the graph traversal as an evaluation of polynomial equations. By using an additional look-up table, our protocol enables traversals both to ascendant and to descendant. Such a feature could be useful for various applications that require more complex graph traversal (i.e. searching on DFA).

References

1. Amazon machine learning - predictive analytics with aws (2017), `https://aws.amazon.com/aml/`
2. Google cloud machine learning at scale — google cloud platform (2017), `https://cloud.google.com/products/machine-learning/`
3. Machine Learning - Microsoft Azure (2017), `https://azure.microsoft.com`
4. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Michael Backes, P.N. (ed.) ESORICS. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009)
5. Bost, R., Popa, R., Tu, S., Goldwasser, S.: Machine Learning Classification over Encrypted Data. In: NDSS. pp. 1–14 (2015)
6. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: CCS. pp. 498–507. ACM Press, New York (2007)
7. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE TC **100**(8), 677–691 (1986)
8. Carmit Hazay, Y.L.: Efficient Secure Two-Party Protocols. Information Security and Cryptography, Springer, Heidelberg, 1st edn. (2010)
9. Certicom Research: Standards for Efficient Cryptography 2 (SEC 2) : Recommended Elliptic Curve Domain Parameters (2010), `http://www.secg.org/sec2-v2.pdf`
10. Cock, M.D., Dowsley, R., Horst, C., Katti, R., Nascimento, A.C.A., Newman, S.C., Poon, W.S.: Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. Cryptology ePrint Archive, Report 2016/736 (2016), `https://eprint.iacr.org/2016/736`
11. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
12. Damgård, I., Geisler, M., Krøigaard, M.: Efficient and Secure Comparison for On-Line Auctions. In: Josef Pieprzyk, Hossein Ghodosi, E.D. (ed.) ACISP. LNCS, vol. 4586, pp. 416–430. Springer, Heidelberg (2007)

13. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: ICML. pp. 201–210. JMLR (2016)
14. Fischer, J., Peters, D.: GLOUDS: Representing tree-like graphs. Discrete Algorithms **36**, 39 – 49 (2016)
15. Fredrikson, M., Jha, S., Ristenpart, T.: Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In: CCS. pp. 1322–1333. ACM Press, New York (2015)
16. Fredrikson, M., Lantz, E., Jha, S., Lin, S.: Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In: USENIX. pp. 17–32 (2014)
17. Huang, C.L., Chen, M.C., Wang, C.J.: Credit scoring with a data mining approach based on support vector machines. Expert Systems with Applications **33**(4), 847–856 (2007)
18. Jacobson, G.: Space-efficient static trees and graphs. In: FOCS. pp. 549–554. IEEE Press, New York (1989)
19. Kohavi, R., Li, C.H.: Oblivious decision trees graphs and top down pruning. In: IJCAI. pp. 1071–1077. Morgan Kaufmann, San Francisco (1995)
20. Lavecchia, A.: Machine-learning approaches in drug discovery: methods and applications. Drug Discovery Today **20**(3), 318–331 (2015)
21. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious Neural Network Predictions via MiniONN Transformations. In: CCS. pp. 619–631. ACM Press, New York (2017)
22. Madabhushi, A., Lee, G.: Image analysis and machine learning in digital pathology: Challenges and opportunities. Medical Image Analysis **33**, 170 – 175 (2016)
23. Meinel, C.: Modified branching programs and their computational power, LNCS, vol. 370. Springer, Heidelberg (1989)
24. Mitsunari, S.: C++ library implementing elliptic curve elgamal crypto system, `https://github.com/herumi/mcl`
25. Mohassel, P., Niksefat, S.: Oblivious Decision Programs from Oblivious Transfer: Efficient Reductions. In: Keromytis, A.D. (ed.) FC. LNCS, vol. 7397, pp. 269–284. Springer, Heidelberg (2012)
26. Oliveira, A.L., Sangiovanni-Vincentelli, A.: Using the minimum description length principle to infer reduced ordered decision graphs. Machine Learning **25**(1), 23–50 (1996)
27. Oliver, J.J.: Decision graphs: an extension of decision trees. Technical report, Monash University, Department of Computer Science (1992)
28. Shimizu, K., Nuida, K., Ratsch, G.: Efficient privacy-preserving string search and an application in genomics. Bioinformatics **32**(11), 1652–1661 (2016)
29. Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., Criminisi, A.: Decision Jungles: Compact and Rich Models for Classification. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) NIPS. pp. 234–242. Curran Associates, Inc., New York (2013)
30. Tai, R.K., Ma, J.P., Zhao, Y., Chow, S.S.: Privacy-preserving decision trees evaluation via linear functions. In: ESORICS. LNCS, vol. 10493, pp. 494–512. Springer, Heidelberg (2017)
31. Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient dna searching through oblivious automata. In: CCS. pp. 519–528. ACM Press, New York (2007)
32. Vaidya, J., Yu, H., Jiang, X.: Privacy-preserving svm classification. Knowledge and Information Systems **14**(2), 161–178 (2008)
33. Verizon: IP Latency Statistics — Verizon Enterprise Solutions (2017), `http://www.verizonenterprise.com/about/network/latency/`

34. Veugen, T.: Improving the DGK comparison protocol. In: WIFS. pp. 49–54. IEEE Press, New York (2012)
35. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately Evaluating Decision Trees and Random Forests. PoPETS (4), 1–21 (2016)
36. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS. pp. 162–167. IEEE Press, New York (1986)

A OT implementation

While there are several efficient implementations that achieve OT_1^N functionality, we use a simple three step protocol based on additively homomorphic operation.

1. The user prepares a binary query vector $\boldsymbol{q}$, that consists of $N-1$ zeros and a one at the i-th element. The user sends $[\![\boldsymbol{q}]\!]$ to the server.
2. The server computes inner product $[\![\boldsymbol{q}\cdot\boldsymbol{v}]\!] = [\![\boldsymbol{v}[i]]\!]$, and computes $[\![\boldsymbol{v}[i]]\!] \oplus [\![0]\!]$ for a random ciphertext $[\![0]\!]$ of plaintext 0 and sends it to the user.
3. The user decrypts the result and obtains $\boldsymbol{v}[i]$.

This implementation requires $O(N)$ computational cost and communication size due to N encryptions (user), N multiplications (server), and a transmission of N ciphertexts.

In principle, various OT protocols can be used for the eROT, however, this implementation does not efficiently work for eROT, if it is implemented with the lifted-Elgamal encrytption, because of its restriction on the message size. (i.e., the protocol needs to decrypt an encryption of a ciphertext.) In order to use the encryption scheme like the lifted-Elgamal, here we also propose another OT protocol in which the server masks all elements by using random values in the range of plaintext of lifted-Elgamal encrytption except for the element the user chose. The server can only keep a chosen element unmasked by creating masks in the following manner: the server flips each bit of the query vector, multiplies them by different random values, and then adds each element to each encrypted message. The user only obtains the unmasked element. For further discussion in Section 3.4, we call the first protocol OT_A and the latter one OT_B.

B Details of ROT

The detailed description of ROT is as follows:
Initial (0-th) step:

1. The server generates a random value $r_1 \in_R \mathbb{Z}$ and prepares a vector $\boldsymbol{v}'$ whose i-th element is masked by r_1 ($\boldsymbol{v}'[i] = (\boldsymbol{v}[i] + r_1)_{\mathsf{mod}\ N}$). The server stores r_1.
2. The user (chooser) and the server (sender) engage in $OT_1^N(p_0, \boldsymbol{v}')$. The user obtains $\boldsymbol{v}'[p_0] = (p_1 + r_1)_{\mathsf{mod}\ N}$.

k-th ($k = 1, \ldots, \sigma - 2$) step:
The user holds $p'_k = (p_k + r_k)_{\mathsf{mod}\ N}$ and the server holds r_k.

1. The server generates a random value $r_{k+1} \in_R \mathbb{Z}$, and the server prepares a vector $\boldsymbol{v}'$ whose i-th element is masked by r_{k+1} ($\boldsymbol{v}'[i] = (\boldsymbol{v}[i] + r_{k+1})_{\mathsf{mod}\ N}$). The server stores r_{k+1}.
2. The server rotates $\boldsymbol{v}'$ by r_k elements to obtain $\hat{\boldsymbol{v}}'$.
3. The user (chooser) and the server (sender) engage in $OT_1^N(p'_k, \hat{\boldsymbol{v}}')$. The user obtains $\boldsymbol{v}'[p_k] = (p_{k+1} + r_{k+1})_{\mathsf{mod}\ N}$.

Last step:

The server does not mask $\boldsymbol{v}[i]$ so that a true value can be sent to the user.

1. The server rotates $\boldsymbol{v}$ by $r_{\sigma-1}$ elements to obtain $\hat{\boldsymbol{v}}$.
2. The user (chooser) and the server (sender) engage in $OT_1^N(p'_{\sigma-1}, \hat{\boldsymbol{v}})$. The user obtains $\boldsymbol{v}[p_{\sigma-1}] = p_\sigma$.