SoK: Modular and Efficient Private Decision Tree Evaluation[*]

Ágnes Kiss[1], Masoud Naderpour[2], Jian Liu[3], N. Asokan[4], Thomas Schneider[1]

[1]TU Darmstadt
{kiss, schneider}@encrypto.cs.tu-darmstadt.de
[2]University of Helsinki
masoud.naderpour@helsinki.fi
[3]University of California, Berkeley[**]
liu.jian@berkeley.edu
[4]Aalto University
asokan@acm.org

Abstract. Decision trees and random forests are widely used classifiers in machine learning. Service providers often host classification models in a cloud service and provide an interface for clients to use the model remotely. While the model is sensitive information of the server, the input query and prediction results are sensitive information of the client. This motivates the need for private decision tree evaluation, where the service provider does not learn the client's input and the client does not learn the model except for its size and the result.

In this work, we identify the three phases of private decision tree evaluation protocols: feature selection, comparison, and path evaluation. We systematize protocols for each of these phases to identify the best available instantiations using the two main paradigms for secure computation: garbling techniques and homomorphic encryption. There is a natural tradeoff between runtime and communication considering these two paradigms: garbling techniques use fast symmetric-key operations but require a large amount of communication, while homomorphic encryption is computationally heavy but requires little communication.

Our contributions are as follows: Firstly, we systematically review and analyse state-of-the-art protocols for the three phases of private decision tree evaluation. Our methodology allows us to identify novel combinations of these protocols that provide better tradeoffs than existing protocols. Thereafter, we empirically evaluate all combinations of these protocols by providing communication and runtime measures, and provide recommendations based on the identified concrete tradeoffs.

Keywords: Privacy-preserving protocols, secure computation, homomorphic encryption, decision tree evaluation, machine learning

1 Introduction

Machine learning is pervasively being applied in various real-world scenarios. All major IT companies, including Amazon, Apple, Google, Facebook and Microsoft, are working on and use machine learning technologies. First a *training phase* takes place where a model is trained on a large dataset so that later it can make predictions on an input, which happens in the *evaluation phase*. One of the most commonly used predictive models are decision trees, whose extension to random forests – sets of smaller decision trees trained on different random subsets of features – is considered to be among the most accurate classification models [DCBA14].

In many scenarios, predictive models such as decision trees and neural networks are trained or queried on sensitive data and should be handled in a private manner [WGC18]. Solutions for *private training* of decision trees exist that have been heavily optimized [LP00, LP02, VC05, VCKP08, BS09, FW12].

In this work, we focus on the *private evaluation* of decision trees with security against passive adversaries. In this scenario, the server holds a decision tree and offers its evaluation as a service to its client with sensitive

---

[*] Please cite the conference version of this work to appear in PoPETs'19(2).
[**] This work was done when the author was in Aalto University

Sub-protocol	Path_H	Path_G
$\text{Sel}_\text{H} + \text{Comp}_\text{G}$	HGH §4.1	HGG [BFK$^+$09]
$\text{Sel}_\text{G} + \text{Comp}_\text{G}$	GGH §4.1	GGG [BFK$^+$09]
$\text{Sel}_\text{H} + \text{Comp}_\text{H}$	HHH [TMZC17]	HHG §4.1
$\text{Sel}_\text{G} + \text{Comp}_\text{H}$	$\times$	$\times$

Table 1: Combinations of sub-protocols for private decision tree evaluation $\text{ABC} = \text{Sel}_\text{A} + \text{Comp}_\text{B} + \text{Path}_\text{C}$. $\text{Sel}_\text{H}/\text{Sel}_\text{G}$ denotes oblivious feature selection, $\text{Comp}_\text{H}/\text{Comp}_\text{G}$ oblivious comparison, and $\text{Path}_\text{H}/\text{Path}_\text{G}$ oblivious path evaluation with homomorphic encryption (H) and garbling techniques (G).

input to the decision tree. Moreover, the model may leak information about the sensitive training data used for its generation, which is a valuable asset of the server. The server's goal is to offer a service without compromising the client's input, or the client learning anything about the decision tree (beyond its size and the result).

Note that recent work [TZJ$^+$16, PMG$^+$17, SSSS17, JSD$^+$18] has shown that it is possible for an attacker who has only blackbox oracle access to prediction APIs of machine learning models to succeed in learning model parameters by repeated adaptive queries to the API. Defending against such attacks is an active area of research and several proposals have already appeared, e.g., [TZJ$^+$16, KMAM17, JSD$^+$18]. We deem dealing with blackbox attacks out of scope for this paper.

1.1 Applications

Cloud-hosted evaluation of machine learning models is a widely used service that companies such as Amazon, Google or IBM provide to their customers. While doing so [Bar15], Amazon Web Services (AWS) recognizes that their clients care about the privacy of their data [Ser18]. These services can be enhanced by using private evaluation of machine learning models to allow for providing a service without violating neither the privacy of the resulting model nor the privacy of the customers' input features. Most machine learning APIs such as MLJAR [MLJ17], BigML [Big18], Wise.io [Wis18], or Azure Machine Learning Studio [Mic18] support decision tree or random forest classifiers. Decision tree evaluation has a tremendous amount of applications, and is being used by companies such as Facebook [IK17] and Microsoft [RG16].

Private decision tree evaluation can provide a solution for private medical diagnosis [TASF09], such as electrocardiogram classification [BFK$^+$09, BFL$^+$11], where the medical history and genomic data of the patient are sensitive information. Decision trees and random forests can be applied for malware [AM18], and text classification [RMD18]. Moreover, decision trees and random forests have been successfully used for detecting insider threat [MAAG15] and multimedia protocol tunneling [BSR18], as well as spam campaigners [GKG$^+$18], and predicting the interruptability intensity of mobile users [YGL17], data hotspots in cellular networks [NIZ$^+$16], and even private interactions in underground forums [OTGT18]. Private evaluation of decision trees and random forests can enhance the privacy that these techniques provide, while protecting the sensitive business information of the service provider. Recently, secure evaluation of branching programs, an extension of decision trees, has also been utilized in a protocol for private set intersection [CO18].

1.2 Outline and Our Contributions

There exist two main paradigms for secure computation: *homomorphic encryption*, which often has little communication but high computation complexity, and *garbling techniques*, which require more communication, but utilize efficient symmetric-key primitives. In this work, we systematically analyze the state-of-the-art protocols for private decision tree evaluation that make use of these paradigms. We explore the tradeoffs and combinations of the identified sub-protocols, compare the performance of all combinations and identify the most efficient private decision tree evaluation protocols to date.

In more details, our contributions are as follows:

Systematization and analysis of existing techniques (§3). Private decision tree evaluation consists of three sub-protocols: The first is *private selection* of features that are to be compared with in each of the

2

$\mathcal{S}$	Server (holds decision tree)
$\mathcal{C}$	Client (holds input features)
T	Decision tree
n	Dimension of feature vector
t	Size of a feature
m	Number of decision nodes
m'	Number of depth-padded decision nodes
$\overline{m}$	Number of possibly depth-padded decision nodes, i.e., m or m'.
d	Depth of decision tree
$\mathbf{x} = \{x_1, \ldots, x_n\}$	Client's feature vector
$\mathbf{y} = \{y_1, \ldots, y_{\overline{m}}\}$	Server's thresholds for each decision node
κ	Symmetric security parameter
s	Statistical security parameter
τ_{sym}	Size of symmetric ciphertext
τ_{ElGamal}	Size of ciphertext in lifted ElGamal
τ_{Paillier}	Size of ciphertext in Paillier
τ_{DGK}	Size of ciphertext in DGK encryption

Table 2: Notations used throughout the paper.

decision nodes, the second is *private comparison*, and the third is *private evaluation of the path* corresponding to the given input in the decision tree. We analyze the state-of-the-art techniques for these sub-protocols that use additively homomorphic encryption (H) as well as those using garbling techniques (G). We denote the different approaches for oblivious selection, comparison, and path evaluation by $\mathrm{Sel_H}/\mathrm{Sel_G}$ (§3.1), $\mathrm{Comp_H}/\mathrm{Comp_G}$ (§3.2), and $\mathrm{Path_H}/\mathrm{Path_G}$ (§3.3), respectively, and describe their efficient instantiations.

Protocols for private decision tree evaluation (§4). We show that different sub-protocols can be combined efficiently, resulting in the six protocols shown in Tab. 1. We adapt all resulting protocols to an offline-online setting, where all operations independent of the input features are precomputed in an offline phase, which results in a more efficient online phase (§4.1). Thereafter, we present protocol extensions for some of our protocols (§4.2).

Empirical evaluation (§5). We evaluate the performance of the resulting private decision tree evaluation protocols and study the tradeoff between runtime and communication on example real-world datasets. We show the competitive performance of general-purpose secure two-party computation protocols for sub-protocols of private decision tree evaluation that has been claimed to be inefficient in [WFNL16, TMZC17].

Improvements and recommendations (§6). We show that our identified hybrid protocols that combine homomorphic encryption with garbling techniques achieve an order of magnitude runtime improvement over state-of-the-art protocols. We provide recommendations for different settings based on our findings.

Future work and open questions. The protocol based solely on garbling techniques can easily be extended to security against malicious clients (cf. §4.1). Tai et al. [TMZC17] propose the same (more costly) extension using zero-knowledge proofs for the protocol based solely on homomorphic encryption. As interesting direction for future work, one could study and extend the security of hybrid protocols against malicious clients.

2 Preliminaries

In this section, we detail the necessary preliminaries to our work. The notations we use are summarized in Tab. 2.

2.1 Decision Trees

A decision tree is a binary tree T with m internal nodes called *decision nodes*, as shown in the example in Fig. 1a. Every leaf node is called a *classification node* and is associated with a classification
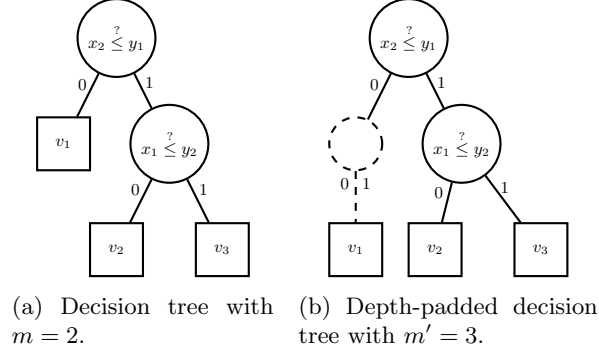
3

(a) Decision tree with $m = 2$.

(b) Depth-padded decision tree with $m' = 3$.

Fig. 1: Decision trees with depth $d = 2$.

value $v_1, \ldots, v_{m+1}$. A decision tree has a threshold vector $\mathbf{y} = \{y_1, \ldots, y_m\}$ and receives an n-dimensional feature vector $\mathbf{x} = \{x_1, \ldots, x_n\}$ as input. At the jth decision node ($j \in \{1, \ldots, m\}$), a comparison between an x_i and y_j takes place ($i \in \{1, \ldots, n\}$). Decision tree evaluation $T(\mathbf{x})$ means evaluating the comparisons at each decision node of a path, the result of which defines which branch is taken next. When a leaf node is reached on the path, the classification value stored in the leaf is outputted. The *depth d* of the decision tree is the length of the longest path between the root and any leaf.

Additionally, we define *depth-padding*, where dummy decision nodes are introduced at every leaf until depth d is reached in order for each path to have the same depth. A dummy node has one outgoing edge, i.e., independently from the comparison result it leads to the same node as shown in Fig. 1b. The number of these dummy nodes, which depends on the structure of the tree, is at most md, i.e., the number of decision nodes in the depth-padded tree is $m \leq m' \leq m(1 + d)$.

Real-world datasets. In this paper, we use the real-world datasets summarized in Tab. 3 for which we trained decision trees using `scikit-learn` [sld17]. These datasets are taken mostly from the UCI machine learning repository [Lic18]: the classical `iris` dataset for pattern recognition of the iris flower, the `wine` and `(breast) cancer` datasets for classification based on the chemical analysis of wines and the characteristics of a diagnostic image, respectively, as well as the `boston` dataset that includes house prices in Boston, the `digits` dataset that contains images of hand-written digits, which has been used by NIST [GBC+97], and the `diabetes` dataset that includes automatic and paper records of diabetes patient glucose measurements for classification. The `linnerud` dataset has been used in [Ten98] for regression and includes physiological measurements. The decision trees are of varying sizes and depths which allows us to study the behaviour of all our identified protocol combinations. Moreover, these datasets are an order of magnitude larger than any real-world dataset used by previous work on private decision tree evaluation [WFNL16, TMZC17]. In Tab. 3, we show the values of n (number of input features), d (depth), m (number of decision nodes), and m' (number of depth-padded decision nodes).

2.2 Cryptographic Techniques

Private decision tree evaluation protocols can be constructed based on both paradigms for secure computation: homomorphic encryption and garbling techniques. We detail the techniques used in this work.

Oblivious transfer. 1-out-of-2 oblivious transfer (OT) allows a sender to obliviously send one of two messages m_0, m_1 to a receiver according to the receiver's selection bit b, without the receiver learning anything about m_{1-b} or the sender learning b. Oblivious transfer generally requires expensive public-key operations. However, a small (symmetric security parameter κ) number of base OTs can be extended to any polynomial number of OTs efficiently using only symmetric-key operations [IKNP03]. Improvements have been presented on OT extension protocols with security against passive [ALSZ13] and active adversaries [ALSZ15, KOS15]. OTs can be efficiently precomputed [Bea95], which is suited for an offline-online scenario, where the parties

Dataset	n	d	m	m'	Source
`iris`	3	5	8	19	UCI
`wine`	7	5	11	26	UCI
`linnerud`	3	6	19	47	[Ten98]
`(breast) cancer`	12	7	21	66	UCI
`digits`	47	15	168	1161	UCI
`diabetes`	10	28	393	6432	UCI
`boston`	13	30	425	6768	UCI

Table 3: Decision trees of varying sizes (i.e., input features n, decision nodes m and padded decision nodes m'), and depths d for real-world datasets, trained using `scikit-learn` [sld17] which contains a library with these example datasets.

can compute parts of the protocol before providing their actual inputs to the computation. During precomputation in the offline phase, it is possible to run random OTs on random inputs, which are then used to mask the actual messages and choice bits. To retrieve the output of the actual OT in the online phase, the parties blind their messages using inexpensive XOR operations, and require $|m_0| + |m_1| + 1$ bits of communication, whereas all cryptographic operations are shifted to the offline phase, which requires τ_{sym} bits of communication per OT.

Yao's garbled circuit protocol. Secure two-party computation allows two parties to compute a functionality on their private inputs without learning anything about the other party's input (beyond the output). Yao's garbled circuit (GC) protocol [Yao82, Yao86] is one of the most widely used solutions for secure two-party computation. Yao's protocol is asymmetric: one of the parties acts as the garbler, who garbles the circuit, i.e., assigns two random keys corresponding to 0 and 1 to each of its wires and encrypts the gates using these keys. The garbler forwards this garbled circuit GC to the other party, the evaluator, who evaluates it after having received the keys corresponding to its input bits from the garbler obliviously through OTs. The protocol therefore relies on OTs linear in the input length of the evaluator and symmetric-key operations linear in the size of the Boolean circuit describing the function. Various optimizations exist including point-and-permute [BMR90], free-XOR [KS08a], fixed-key AES garbling [BHKR13], and half-gates [ZRE15], while at the implementation level Single Instructions Multiple Data (SIMD) operations can be used [BJL12]. All these state-of-the-art optimizations are implemented in the ABY framework [DSZ15], which we use in our implementation. With these optimizations, each AND gate of the circuit needs $2\tau_{\text{sym}}$ bits of communication.

Additionally, we use the point-and-permute technique in our protocols [BMR90]. In the classic garbling scheme, each wire i has two labels (k_i^0 for 0 and k_i^1 for 1) associated with it. In order to hide the truth values of these wires with minimal overhead, a *color bit* can be appended to each label (opposite color bits for 0 and 1). The relation of the color bits and the truth values are known only to the garbler, while the evaluator holds one label from each wire, the color bit of which it can use to proceed with evaluation. In the end of the protocol, the evaluator possesses the wire labels for the output value, which have the color bits appended to them.

We use the following notations for the garbling scheme: $(GC; \text{out}^0, \text{out}^1) \leftarrow \texttt{Garble}(C, \text{in}^0, \text{in}^1)$ denotes the garbler's garbling method that returns the garbled circuit GC, including the output wire keys (out^0, out^1) for both the 0 and 1 values, respectively. It gets as input circuit C and input wire key pairs (in^0, in^1) for the evaluator's input wires. $(GC; \texttt{color}) \leftarrow \texttt{Garble'}(C, \text{in}^0, \text{in}^1)$ denotes the same garbling method but returns the color bits along with GC. $\text{out} \leftarrow \texttt{Eval}(GC, \text{in})$ denotes the garbled circuit evaluation where the evaluator inputs the wire keys in corresponding to its inputs (received via OTs from the garbler), and receives the output wire keys out corresponding to its output bits. The actual output remains oblivious to the evaluator, since he does not learn which values his wire keys correspond to. $\text{out}' \leftarrow \texttt{Eval'}(GC, \text{in})$ denotes a similar method, where the evaluator receives blinded output bits out', i.e., the color bits of the outputs.

Homomorphic encryption. Homomorphic encryption allows anyone holding the public key pk to calculate $[x_1 + x_2] = [x_1] + [x_2]$, given two ciphertexts $[x_1] := \textsf{Enc}_{pk}(x_1)$ and $[x_2] := \textsf{Enc}_{pk}(x_2)$. Performing

---

Ideal functionality $\mathcal{F}_{\text{PDTE}}$.

1. **Private input of $\mathcal{S}$**: decision tree T.
2. **Private input of $\mathcal{C}$**: inputs $\mathbf{x} = \{x_1, \ldots, x_n\}$.
3. **Compute**: evaluate $T(\mathbf{x})$.
4. **Output of $\mathcal{C}$**: classification result $v = T(\mathbf{x})$.

Fig. 2: $\mathcal{F}_{\text{PDTE}}$ – Ideal functionality for private decision tree evaluation (PDTE). A decision tree T is defined by its topology, selection $\sigma : \overline{m} \to n$, thresholds $\mathbf{y} = \{y_1, \ldots, y_{\overline{m}}\}$ and classification values $v_1, \ldots, v_{\overline{m}+1}$.

$c \cdot [x]$ with constant c is supported since this is equal to adding $[x]$ with itself c times, which can be efficiently computed using the double-and-add method.

We use semantically secure additively homomorphic encryption schemes. More specifically, we use Paillier encryption with packing [Pai99, DJ01], DGK encryption [DGK08], and Lifted ElGamal encryption [ElG85]. With a plaintext space of $\mathbb{Z}_N$, the ciphertext space of Paillier and DGK encryption are $\mathbb{Z}_{N^2}^*$ and $\mathbb{Z}_N^*$, respectively, where N is an RSA modulus. Paillier encryption allows for packing of plaintexts and ciphertexts when the encrypted values are smaller than the plaintext space. We use ciphertext packing since the ciphertext $[x]$ in our case is much larger than the plaintext x. Thus, we can pack multiple ciphertexts $[x_1], ..., [x_n]$ into a single one: $[X] = [x_1 + 2^{|x_1|}x_2 + ... + 2^{|x_{n-1}|}x_n]$, and reduce the communication from $n \cdot \tau_{\text{Paillier}}$ to $n \cdot \tau_{\text{Paillier}}/|x_i|$.

Lifted ElGamal encryption is similar to the classical ElGamal scheme, with the exception that the plaintext message is in the exponent, i.e., m is replaced by g^m, where g is the group generator. This allows for additive homomorphism but makes decryption more difficult due to the discrete logarithm that needs to be solved. Lifted ElGamal can be instantiated using elliptic curve (EC) cryptography, where the ciphertext is represented by two EC points. This scheme is thus very efficient but can only be used when the plaintext values are of a known small subset of the plaintext space.

3 Protocol Building Blocks

Private decision tree evaluation is run between two parties: a server $\mathcal{S}$ holding a – potentially depth-padded – classification tree T (with threshold vector $\mathbf{y}$, input selection σ and classification values) and a client $\mathcal{C}$ providing its input feature vector $\mathbf{x}$. After the protocol evaluation, $\mathcal{C}$ obtains the classification result, without learning any information about the tree beyond its size and the output, and $\mathcal{S}$ learns nothing about $\mathcal{C}$'s input features. The corresponding ideal functionality $\mathcal{F}_{\text{PDTE}}$ is depicted in Fig. 2.

In this section, we recapitulate the state-of-the-art sub-protocols for private decision tree evaluation with security against semi-honest adversaries. We describe each protocol and refer to the original publications for specific details on each construction. In §3.1, we detail two protocols for privately selecting $\overline{m}$ features from n input features that are assigned to a decision node to be compared at (*selection phase*). In §3.2, we recapitulate two sub-protocols for private comparison (*comparison phase*): one based on garbled circuits and another based on additively homomorphic encryption. Thereafter in §3.3, we provide protocols for obliviously evaluating the path to the classification leaf based on the results of the previous phase (*path evaluation phase*).

In all sub-protocols presented in this section, the number of decision nodes in decision tree T is either the original number of decision nodes, i.e. m, or the number of depth-padded decision nodes m' (cf. §2.1). This is according to the approach chosen for path evaluation (§3.3). Therefore, we use $\overline{m}$ decision nodes in our description, where possibly $\overline{m} = m$ or $\overline{m} = m'$.

For completeness, we give in §A the communication of all sub-protocols in Tab. 5 and their asymptotic computation complexities in Tab. 6.

Ideal functionality $\mathcal{F}_{\mathrm{Sel}}$, given a secret sharing scheme S_1.

1. **Private input of $\mathcal{S}$**: selection $\sigma : \overline{m} \to n$.
2. **Private input of $\mathcal{C}$**: inputs $\mathbf{x} = \{x_1, \ldots, x_n\}$.
3. **Compute**: $\mathbf{z} = \{x_{\sigma(1)} \ldots, x_{\sigma(\overline{m})}\}$, i.e., select $\overline{m}$ values from the n inputs with selection σ.
4. **Output of $\mathcal{S}, \mathcal{C}$**: a secret share of $\mathbf{z}$ using S_1.

Fig. 3: $\mathcal{F}_{\mathrm{Sel}}$ – Ideal functionality for selecting $\overline{m}$ elements from $n \leq \overline{m}$ elements obliviously.

Ideal functionality $\mathcal{F}_{\mathrm{Comp}}$, given two secret sharing schemes S_1, S_2.

1. **Private inputs of $\mathcal{S}$, $\mathcal{C}$**: secret shares of the selection $x_{\sigma(1)} \ldots, x_{\sigma(\overline{m})}$ according to S_1.
2. **Private input of $\mathcal{S}$**: $\mathbf{y} = \{y_1, \ldots, y_{\overline{m}}\}$.
3. **Compute**: comparison of inputs, i.e., for $i \in \{1, \ldots, \overline{m}\}$: $c_i = 1$ if $x_{\sigma(i)} > y_i$, $c_i = 0$ otherwise.
4. **Output of $\mathcal{S}, \mathcal{C}$**: share of $c_1, \ldots, c_{\overline{m}}$ with S_2.

Fig. 4: $\mathcal{F}_{\mathrm{Comp}}$ – Ideal functionality for obliviously comparing $\overline{m}$ pairs of elements.

Ideal functionality $\mathcal{F}_{\mathrm{Path}}$, given secret sharing scheme S_2.

1. **Private inputs of $\mathcal{S}, \mathcal{C}$**: secret shares of the comparison results $\{c_1, \ldots, c_{\overline{m}}\}$ with S_2.
2. **Private input of $\mathcal{S}$**: decision tree T.
3. **Compute**: evaluation of the decision tree path using the comparison results.
4. **Output of $\mathcal{S}$**: classification result v.

Fig. 5: $\mathcal{F}_{\mathrm{Path}}$ – Ideal functionality for oblivious decision tree path evaluation.

3.1 Selection Phase

The selection phase obliviously selects from the n features of the client $\mathcal{C}$ a value for the $\overline{m}$ decision nodes (where they are to be compared to thresholds) according to the selection $\sigma : \overline{m} \to n$ of the server $\mathcal{S}$. The ideal functionality $\mathcal{F}_{\mathrm{Sel}}$ that describes this selection is shown in Fig. 3. We identify two ways to instantiate it, one based on additively homomorphic encryption ($\mathrm{Sel_H}$), and an other based on the secure evaluation of a Boolean selection network using garbled circuits ($\mathrm{Sel_G}$). In both protocols, the outputs of $\mathcal{S}, \mathcal{C}$ are secret shares of the result: none of the parties learns any information about the output, but they are able to recover it together.

Selection using additively homomorphic encryption ($\mathrm{Sel_H}$, Fig. 6). Additively homomorphic encryption has been used to perform oblivious selection in [BPSW07, BFK$^+$09]. The client $\mathcal{C}$ encrypts all its inputs, and sends the ciphertexts over to the server $\mathcal{S}$ along with its public key. $\mathcal{S}$ then selects the values according to selection σ and homomorphically blinds these with statistical blinding using fresh randomness (pads are longer than the plaintext value by s bits, where s is the statistical security parameter). $\mathcal{S}$ then sends these to $\mathcal{C}$, who decrypts them, retrieving a blinded selection of its inputs. In this sub-protocol, we use either Paillier encryption with ciphertext packing for the $\overline{m}$ ciphertexts to reduce the communication to $(n + \overline{m}/(t + s)) \cdot \tau_{\mathrm{Paillier}}$ bits (cf. §2.2), or DGK encryption with smaller ciphertexts and $(n + \overline{m}) \cdot \tau_{\mathrm{DGK}}$ bits of communication.

Selection using a garbled selection network ($\mathrm{Sel_G}$, Fig. 7). An alternative for selecting the features is evaluating a selection network [KS08b, BFK$^+$09] obliviously, which is based on Waksman's permutation
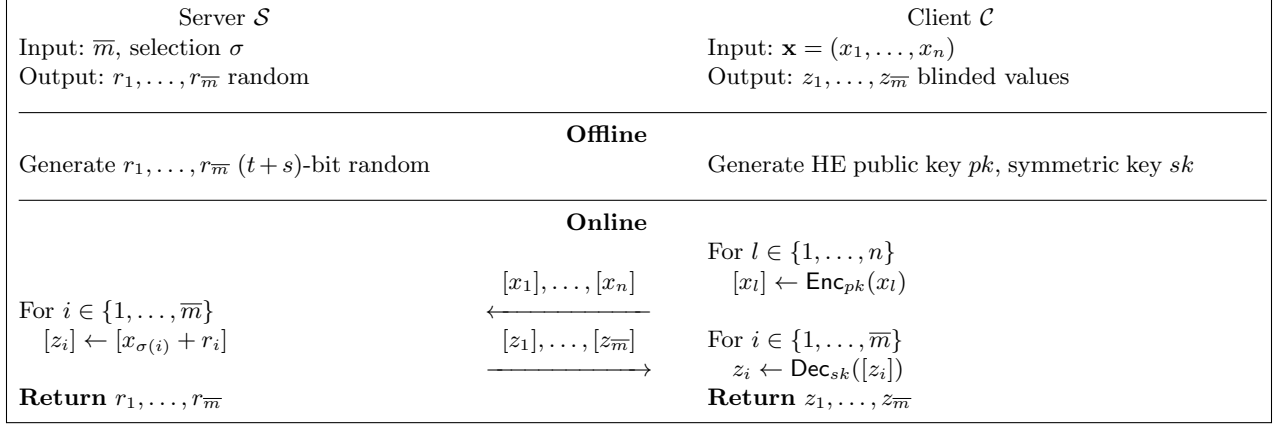
Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\overline{m}$, selection σ		Input: $\mathbf{x} = (x_1, \ldots, x_n)$
Output: $r_1, \ldots, r_{\overline{m}}$ random		Output: $z_1, \ldots, z_{\overline{m}}$ blinded values

Offline		
Generate $r_1, \ldots, r_{\overline{m}}$ $(t+s)$-bit random		Generate HE public key pk, symmetric key sk

Online		
		For $l \in \{1, \ldots, n\}$
	$[x_1], \ldots, [x_n]$	$\quad [x_l] \leftarrow \mathsf{Enc}_{pk}(x_l)$
	$\longleftarrow$	
For $i \in \{1, \ldots, \overline{m}\}$		
$\quad [z_i] \leftarrow [x_{\sigma(i)} + r_i]$	$[z_1], \ldots, [z_{\overline{m}}]$	For $i \in \{1, \ldots, \overline{m}\}$
	$\longrightarrow$	$\quad z_i \leftarrow \mathsf{Dec}_{sk}([z_i])$
Return $r_1, \ldots, r_{\overline{m}}$		**Return** $z_1, \ldots, z_{\overline{m}}$

Fig. 6: $\mathrm{Sel_H}$ – Oblivious selection using additively homomorphic encryption.

Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\overline{m}$, selection σ		Input: $\mathbf{x} = (x_1, \ldots, x_n)$
Output: $K^0 = (K^0_{1,1}, \ldots, K^0_{\overline{m},t}), K^1 = (K^1_{1,1}, \ldots, K^1_{\overline{m},t})$		Output: $K = (K_1, \ldots, K_{\overline{m},t})$ keys

Offline		
Generate C of selection network realizing σ	Precompute nt OTs	
$(GC; (K^0_{1,1}, \ldots, K^0_{\overline{m},t}), (K^1_{1,1}, \ldots, K^1_{\overline{m},t})) \leftarrow$	GC	
$\quad \mathsf{Garble}(C, (s^0_{1,1}, \ldots, s^0_{n,t}), (s^1_{1,1}, \ldots, s^1_{n,t}))$	$\longrightarrow$	

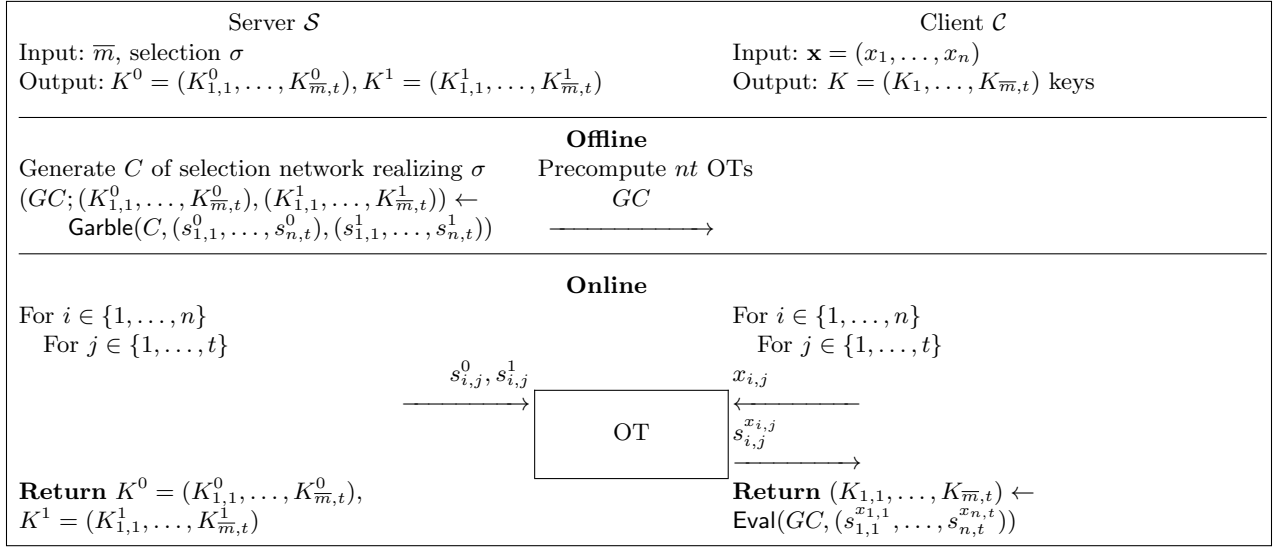| **Online**		
For $i \in \{1, \ldots, n\}$		For $i \in \{1, \ldots, n\}$
$\quad$ For $j \in \{1, \ldots, t\}$		$\quad$ For $j \in \{1, \ldots, t\}$
	$s^0_{i,j}, s^1_{i,j}$	$x_{i,j}$
	$\longrightarrow \boxed{\text{OT}} \xleftarrow{}$	
	$s^{x_{i,j}}_{i,j} \longrightarrow$	
Return $K^0 = (K^0_{1,1}, \ldots, K^0_{\overline{m},t})$,		**Return** $(K_{1,1}, \ldots, K_{\overline{m},t}) \leftarrow$
$K^1 = (K^1_{1,1}, \ldots, K^1_{\overline{m},t})$		$\mathsf{Eval}(GC, (s^{x_{1,1}}_{1,1}, \ldots, s^{x_{n,t}}_{n,t}))$

Fig. 7: $\mathrm{Sel_G}$ – Oblivious selection using a garbled selection network.

network [Wak68]. Firstly, the server sends the GC corresponding to the selection network to the client. They then perform nt OTs for each bit of the client's input. In most cases, $n \leq \overline{m}$, since features are used more than once. The size of a selection network that maps $n \leq \overline{m}$ t-bit elements to $\overline{m}$ elements is $t \cdot S^n_{\overline{m} \geq n} = t \cdot (0.5(n + \overline{m}) \log(n) + \overline{m} \log(\overline{m}) - n + 1)$ [KS16, Appendix C]. The communication of this variant is $(S^n_{\overline{m} \geq n} + n) \cdot 2t \cdot \tau_{\mathrm{sym}}$.

3.2 Comparison Phase

This section details two oblivious comparison protocols, the ideal functionality $\mathcal{F}_{\mathrm{Comp}}$ of which is given in Fig. 4. The outputs of $\mathcal{S}, \mathcal{C}$ are secret shares of the result. The first uses homomorphic encryption (instantiating both $\mathrm{Sel_H}$ and $\mathrm{Comp_H}$), the second uses Yao's GC ($\mathrm{Comp_G}$).

Comparison (including selection) using additively homomorphic encryption ($\mathrm{Sel_H} + \mathrm{Comp_H}$, Fig. 8). This comparison protocol, often referred to as the DGK comparison protocol, was proposed in [DGK07, DGK08, DGK09] and used for private decision tree evaluation in [WFNL16, TMZC17]. Since the comparison protocol is performed using homomorphically encrypted bits of the client's input and known

Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\mathbf{y} = (y_1, \ldots, y_{\overline{m}})$, selection σ		Input: $\mathbf{x} = (x_1, \ldots, x_n)$
Output: $a_1, \ldots, a_{\overline{m}}$ share		Output: $b_1, \ldots, b_{\overline{m}}$ share
Offline		
Generate $a_1, \ldots, a_{\overline{m}}$ random bits		Generate HE public key pk, symmetric key sk
Online		
		For $i \in \{1, \ldots, n\}$
		For $j \in \{1, \ldots, t\}$
	$[x_{1,1}], \ldots, [x_{n,t}]$	$[x_{i,j}] = \mathsf{Enc}_{pk}(x_{i,j})$
For $k \in \{1, \ldots, \overline{m}\} : s_k \leftarrow 1 - 2 \cdot a_k$	$\longleftarrow$	
For $i \in \{1, \ldots, t\}$		
$r_{k,i} \leftarrow \mathbb{Z}_p^*$, $[c_{k,i}] = [r_{k,i} \cdot f_{k,i}(x_\sigma, y)]$	$[c_{1,1}], \ldots, [c_{\overline{m},t}]$	For $k \in \{1, \ldots, \overline{m}\}$
Return $(a_1, \ldots, a_{\overline{m}})$	$\longrightarrow$	If $\exists i \in \{1, \ldots, t\} : \mathsf{Dec}_{sk}([c_{k,i}]) = 0$ then $b_k = 1$
		else $b_k = 0$
		Return $(b_1, \ldots, b_{\overline{m}})$

Fig. 8: $\mathrm{Sel_H}+\mathrm{Comp_H}$ – Oblivious selection and comparison using additively homomorphic encryption from [WFNL16, TMZC17].

Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\mathbf{y} = (y_1, \ldots, y_{\overline{m}})$, K^0, K^1		Input: $K = (K_{1,1}, \ldots, K_{\overline{m},t})$ keys
Output: (a) $\mathbf{a} = (a_1, \ldots, a_{\overline{m}})$ share		Output:(a) $\mathbf{b} = (b_1, \ldots, b_{\overline{m}})$ share
Output: (b) $\mathbf{k}^0 = (k_1^0, \ldots, k_{\overline{m}}^0), \mathbf{k}^1 = (k_1^1, \ldots, k_{\overline{m}}^1)$		Output: (b) $\mathbf{k} = (k_1, \ldots, k_{\overline{m}})$ keys
Generate C of $\overline{m}$ t-bit comparisons with $\mathbf{y}$		
(a) $(GC; (a_1, \ldots, a_{\overline{m}})) \leftarrow \mathsf{Garble'}(C, K^0, K^1)$	GC	
(b) $(GC; (k_1^0, \ldots, k_{\overline{m}}^0), (k_1^1, \ldots, k_{\overline{m}}^1)) \leftarrow \mathsf{Garble}(C, K^0, K^1)$	$\longrightarrow$	
Return (a) color bits $\mathbf{a} = (a_1, \ldots, a_{\overline{m}})$,		**Return** (a) $\mathbf{b} = (b_1, \ldots, b_{\overline{m}}) \leftarrow \mathsf{Eval'}(GC, K)$
Return (b) keys $\mathbf{k}^0 = (k_1^0, \ldots, k_{\overline{m}}^0)$, $\mathbf{k}^1 = (k_1^1, \ldots, k_{\overline{m}}^1)$		**Return** (b) $\mathbf{k} = (k_1, \ldots, k_{\overline{m}}) \leftarrow \mathsf{Eval}(GC, K)$

Fig. 9: $\mathrm{Comp_G}$ – Oblivious comparison using garbled circuits.

thresholds, the selection phase $\mathrm{Sel_H}$ can be realized within this protocol without additional overhead. Thus, it does not need to be combined with the protocols presented in §3.1 (cf. Tab. 1). It is observed that $x < y$ is true if and only if $\exists i \in \{1, \ldots, t\}$ such that $f_i(x, y) = x_i - y_i + 1 + 3 \sum_{j<i}(x_j \oplus y_j) = 0$ [DGK07]. $\mathcal{C}$ encrypts each bit of its input with an additively homomorphic encryption scheme (cf. §2.2), and sends the ciphertexts to $\mathcal{S}$, who chooses the appropriate feature to compare with at each decision node and generates a random bit a_k for each comparison ($k \in \{1, \ldots, \overline{m}\}$). It then performs the above comparison that outputs a blinded ciphertext $[r_i \cdot f_i(x, y)]$ for each bit of each feature. If any of these ciphertexts decrypts to zero, then $x_i < y_i$. The comparison bit is not revealed to the client in clear, but is secret shared between the parties ($\mathcal{S}$ holds random bit a_k, $\mathcal{C}$ holds the result blinded by a_k, i.e., $b_k = a_k \oplus [x_{\sigma,k} < y_k]$). The efficient lifted ElGamal encryption scheme can be used here (cf. §2.2), since the client needs only to check if the result is g^0 or not. The communication in this variant is $(n + \overline{m}) \cdot t \cdot \tau_{\mathrm{ElGamal}}$ bits.

Joye and Salehi presented an optimization of the DGK comparison protocol in [JS18] that improves the communication roughly by factor two when comparing two values with each other. However, it does not generalize the same way as the DGK comparison protocol presented in Fig. 8, since a selection step has to
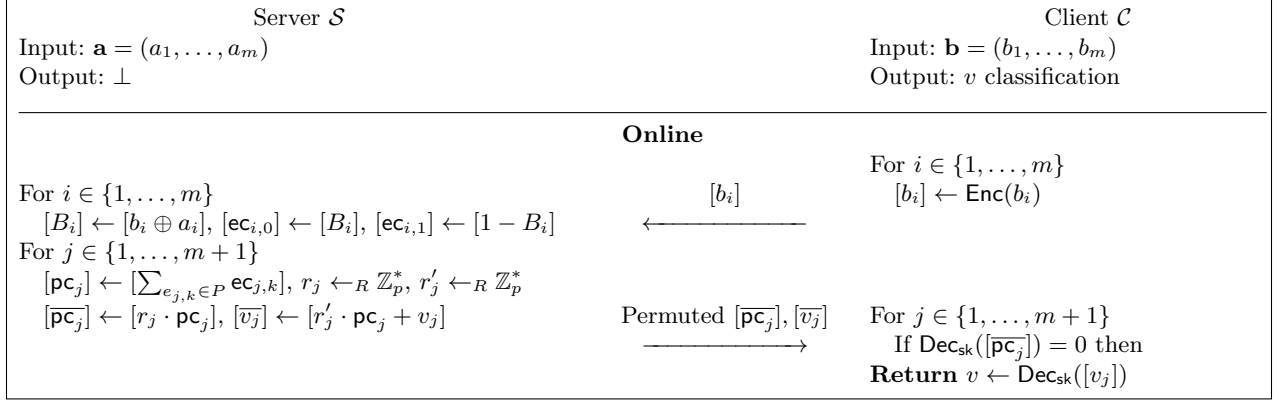
9

Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\mathbf{a} = (a_1, \ldots, a_m)$		Input: $\mathbf{b} = (b_1, \ldots, b_m)$
Output: $\perp$		Output: v classification

Online

For $i \in \{1, \ldots, m\}$ $[b_i]$ For $i \in \{1, \ldots, m\}$
   $[B_i] \leftarrow [b_i \oplus a_i]$, $[\mathsf{ec}_{i,0}] \leftarrow [B_i]$, $[\mathsf{ec}_{i,1}] \leftarrow [1 - B_i]$ $\longleftarrow$ $[b_i] \leftarrow \mathsf{Enc}(b_i)$
For $j \in \{1, \ldots, m+1\}$
   $[\mathsf{pc}_j] \leftarrow [\sum_{e_{j,k} \in P} \mathsf{ec}_{j,k}]$, $r_j \leftarrow_R \mathbb{Z}_p^*$, $r_j' \leftarrow_R \mathbb{Z}_p^*$
   $[\overline{\mathsf{pc}_j}] \leftarrow [r_j \cdot \mathsf{pc}_j]$, $[\overline{v_j}] \leftarrow [r_j' \cdot \mathsf{pc}_j + v_j]$ Permuted $[\overline{\mathsf{pc}}_j], [\overline{v_j}]$ For $j \in \{1, \ldots, m+1\}$
       $\longrightarrow$ If $\mathsf{Dec}_{\mathsf{sk}}([\overline{\mathsf{pc}}_j]) = 0$ then
           Return $v \leftarrow \mathsf{Dec}_{\mathsf{sk}}([v_j])$

Fig. 10: $\mathrm{Path_H}$ – Oblivious path evaluation using additively homomorphic encryption from [TMZC17].

Server $\mathcal{S}$		Client $\mathcal{C}$
Input: $\mathbf{k}^0 = (k_1^0, \ldots, k_{m'}^0), \mathbf{k}^1 = (k_1^1, \ldots, k_{m'}^1)$		Input: $\mathbf{k} = (k_1, \ldots, k_{m'})$
Output: $\perp$		Output: v classification

Offline

For $i \in \{1, \ldots, m'+1\}$
   $\overline{DT} \leftarrow DT.\mathsf{Depth\text{-}Pad}(l_i, d))$
$GDT \leftarrow \mathsf{Garble}(\overline{DT}, \mathbf{k}^0, \mathbf{k}^1)$ GDT
       $\longrightarrow$

Online

           Return $v \leftarrow \mathsf{Eval}(GDT, \mathbf{k})$
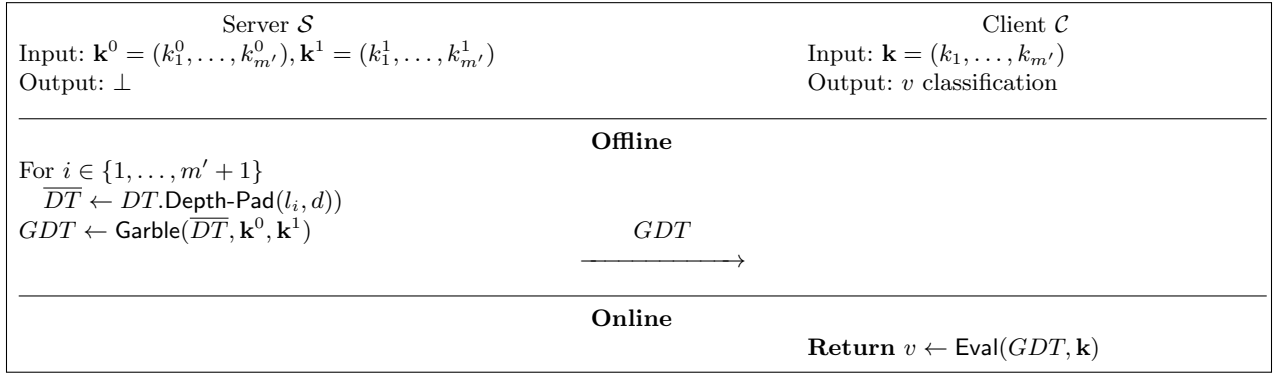
Fig. 11: $\mathrm{Path_G}$ – Oblivious path evaluation using a garbled decision tree.

be included separately before the comparison. This incurs an overhead when the client encrypts its elements, i.e., it now needs to encrypt $\overline{m}$ instead of n elements. However, Joye and Salehi also proposed a private decision tree evaluation protocol that uses their protocol such that only d comparisons between two values are performed as in plain decision tree evaluation. We compare with this private decision tree evaluation protocol in §4.2.

Comparison using garbled circuits ($\mathbf{Comp_G}$, Fig. 9). All oblivious comparisons can also be performed with generic secure two-party computation protocols. This method has been used for comparison in [BPSW07, BFK+09]. Yao's garbled circuit is particularly efficient for private comparison, and requires t AND gates per t-bit comparisons [KSS09]. Therefore, $\overline{m}$ comparisons require $2 \cdot \tau_{\mathrm{sym}} \cdot \overline{m}t$ bits communication. Variant (a) of this comparison protocol outputs the Boolean shares of the output of the comparisons (the color bits in Yao's GC protocol (cf. §2.2). Variant (b) outputs the keys that correspond to the output wires: the server holds key pairs for both 0 and 1, while the client holds the output wires corresponding to the output (without knowing the values they correspond to).

3.3 Path Evaluation Phase

This section describes two path evaluation protocols, the first using homomorphic encryption ($\mathrm{Path_H}$) and the second unsing a garbled decision tree ($\mathrm{Path_G}$). The ideal functionality $\mathcal{F}_{\mathrm{Path}}$ is given in Fig. 5.

Path evaluation using additively homomorphic encryption ($\mathbf{Path_H}$, Fig. 10). The path evaluation protocol of [TMZC17] is highly optimized and depends only on the number of decision nodes m. The client $\mathcal{C}$ encrypts his share of the comparison bit for each decision node, and sends it to the server. The

Protocol	$\overline{m}$	Selection (Sel$_*$)	Interface	Comparison (Comp$_*$)	Interface	Path evaluation (Eval$_*$)	Online rounds	Leakage
[BPSW07]	m	H		G		G	4	m, d_{path}
[WFNL16]	m	H (Fig. 8)		H (Fig. 8)		H	6	m
HGG [BFK$^+$09]	m'	H (Fig. 6)	G (Fig. 13)	G (Fig. 9)		G (Fig. 11)	4	m', d
GGG [BFK$^+$09]	m'	G (Fig. 7)		G (Fig. 9)		G (Fig. 11)	2	m', d
HHH [TMZC17]	m	H (Fig. 8)		H (Fig. 8)		H (Fig. 10)	4	m
HHG	m'	H (Fig. 8)		H (Fig. 8)	G (Fig. 14)	G (Fig. 11)	4	m', d
HGH	m	H (Fig. 6)	G (Fig. 13)	G (Fig. 9)		H (Fig. 10)	6	m
GGH	m	G (Fig. 7)		G (Fig. 9)		H (Fig. 10)	4	m

Table 4: Concrete protocols based on the presented sub-protocols for selection, comparison and path evaluation, where $\overline{m}$ denotes the number of decision nodes m or padded decision nodes m' in the protocols. HHH is the state-of-the-art protocol of [TMZC17]. Round complexity and leakage of different private decision tree evaluation protocols based on homomorphic encryption H and garbling techniques G.

server $\mathcal{S}$ homomorphically computes the encryption of the actual comparison bit, which is the *edge cost* of the edge from the decision node that corresponds to 0 (left). The other edge, corresponding to 1 (right), gets the opposite of this bit as its edge cost. For each leaf node its *path cost* is computed by summing up the edge costs along its path from the root. This means $\mathcal{O}(m)$ operations since intermediate values can be reused. These and the classification labels on each leaf node are then blinded by $\mathcal{S}$ using fresh randomness, and sent to the client $\mathcal{C}$. If the path cost decrypts to zero, $\mathcal{C}$ can decrypt the classification label. This path evaluation protocol has $(3m + 2) \cdot \tau_{\text{ElGamal}}$ bits of communication.

Path evaluation using a garbled decision tree (Path$_{\mathbf{G}}$, Fig. 11). The method for garbled decision tree (GDT) evaluation is analogous to Yao's garbled circuit technique [Yao86] described in §2.2, but $\mathtt{Eval}(GDT, \mathtt{in})$ returns the output classification in the clear. Our description is similar to that of [BPSW07] that has been improved in [BFK$^+$09, BFL$^+$11]. It relies on the idea of encrypting two keys at each decision node, one to the left and one to the right child nodes, along with their node indices, such that the evaluator $\mathcal{C}$ cannot deviate from the path corresponding to the comparison results with his input vector $\mathbf{x}$. We introduce *dummy decision nodes* for hiding the length of the path to the classification node. Revealing this length would reveal information about the topology of the tree, especially when multiple protocol runs with the same model are possible. One can use full decision trees [BPSW07], the overhead of which is exponential in the depth d. This is much higher for large trees than using depth-padding with m' decision nodes (cf. §2.1) as shown in Fig. 12. Moreover, we reveal only the depth-padded number of decision nodes m', and nothing about m since both of them would reveal information about the tree topology.

The server randomly permutes the nodes and garbles the decision tree so that each node has an index i and an encryption key k_i corresponding to it. At each node, the keys and indices of the children nodes are stored in an encrypted manner using the key of the node and keys k_i^0 or k_i^1 depending on the comparison result. This means that at node i, $\mathsf{Enc}_{k_i, k_i^0}(k_l || l)$ and $\mathsf{Enc}_{k_i, k_i^1}(k_r || r)$ are stored, where l is the index of the node on the left and r is the index of the node on the right. The order of the two ciphertexts is permuted according to the color bits of k_i^0 and k_i^1. The server sends GDT to the client, who can evaluate it by decrypting one path leading to the leaf node storing the classification label. The communication in this protocol is sending the GDT, each node of which stores two τ_{sym}-bit values and two indices, i.e., $2m' \cdot (\tau_{\text{sym}} + \log_2(m' + 1))$.

An extension of this protocol reduces the client's online computation to $\mathcal{O}(dt)$ and is presented in §4.2.

4 Protocols for Private Decision Tree Evaluation

In this section, we discuss all possible combinations of the sub-protocols from §3 that result in private decision tree evaluation protocols. We assess these in Tab. 4, which is an extended version of Tab. 1. In addition to the sub-protocols, we recapitulate the number of online rounds and the leakage about the model. In private decision tree evaluation, the client only learns the result and some information about the size of the
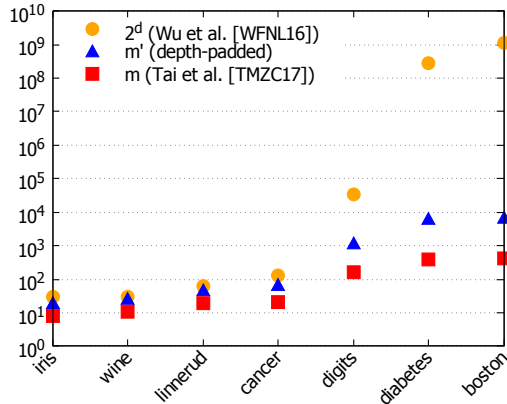
Fig. 12: Number of decision nodes in the different protocols for the decision trees trained on real-world datasets from Tab. 3. Note that the y-axis is in logarithmic scale.

server's decision tree: either the number of decision nodes m or the number of padded decision nodes m' and potentially the depth d. We adapt all resulting protocols with security against passive adversaries to an offline-online setting.

Security guarantees. We combine sub-protocols secure against passive adversaries with secret shared output between the two parties. The outputs of our primitives for selection (Sel_H and Sel_G) and comparison (Comp_H and Comp_G) as well as the inputs for comparison and path evaluation (Path_H and Path_G) are secret shared as indicated in Figs. 3-5, i.e., the parties together could reconstruct the result but do not learn any information by themselves. The secret sharing technique is either XOR-sharing, additive blinding or garbled inputs/outputs with the corresponding key(s). Therefore, our sequentially composed protocols remain secure against passive adversaries [BLW08].

Selection and Comparison Strategy. For oblivious selection and comparison, the state-of-the-art protocols use additively homomorphic encryption Sel_H (Fig. 6) [BFK$^+$09] and Comp_H (Fig. 8) [TMZC17]. This strategy is beneficial when it comes to communication and storage, but implies an increased runtime compared to garbling techniques. Garbling techniques have been used for oblivious selection Sel_G (Fig. 7) and comparison Comp_G (Fig. 9) in [BFK$^+$09]. Thereafter, these techniques have been claimed to be inefficient in [WFNL16, TMZC17]. However, our experiments in §5 show that they have a very efficient online phase, and perform better with respect to total runtime.

Path Evaluation Strategy. As discussed in §3.3, the path evaluation sub-protocol of [TMZC17] using homomorphic encryption via Path_H (Fig. 10) depends only on the number of decision nodes m and is therefore especially efficient with sparse and large decision trees. When this sub-protocol is in use, $\overline{m} = m$ in all preceding sub-protocols. This has been a tremendous improvement over the path evaluation method of Wu et al. [WFNL16], which depends exponentially on the depth d of the tree, due to expanding the tree to a full decision tree.

As opposed to this, in Path_G (Fig. 11) the paths need to have the same length in each path evaluation in order to hide the tree topology. Therefore, the tree is depth-padded to $\overline{m} = m'$ decision nodes. m' depends on the topology of the decision tree, i.e., the level of all leaves, but is upper bounded by $m(d+1)$ (cf. §2.1).

We depict in Fig. 12 the different numbers of decision nodes that are used in state-of-the-art protocols and our depth-padded approach, i.e., the original number of decision nodes m (Path_H, Fig. 10, Tai et al. [TMZC17]), the depth-padded number of decision nodes m' (Path_G, Fig. 11), and the number of decision nodes in a full tree 2^d (Wu et al. [WFNL16]). Using a full tree is impractical with large decision trees due to the exponential dependency on the depth d.
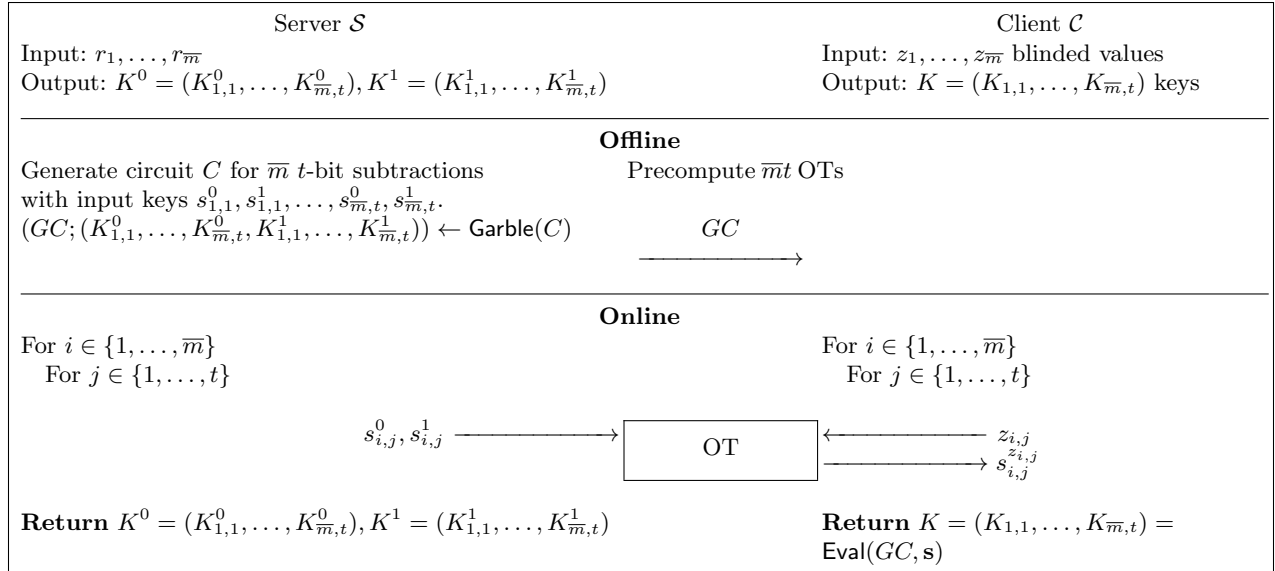
12

Server $\mathcal{S}$	Client $\mathcal{C}$
Input: $r_1, \ldots, r_{\overline{m}}$	Input: $z_1, \ldots, z_{\overline{m}}$ blinded values
Output: $K^0 = (K_{1,1}^0, \ldots, K_{\overline{m},t}^0), K^1 = (K_{1,1}^1, \ldots, K_{\overline{m},t}^1)$	Output: $K = (K_{1,1}, \ldots, K_{\overline{m},t})$ keys

Offline

Generate circuit C for $\overline{m}$ t-bit subtractions   Precompute $\overline{m}t$ OTs
with input keys $s_{1,1}^0, s_{1,1}^1, \ldots, s_{\overline{m},t}^0, s_{\overline{m},t}^1$.
$(GC; (K_{1,1}^0, \ldots, K_{\overline{m},t}^0, K_{1,1}^1, \ldots, K_{\overline{m},t}^1)) \leftarrow \mathsf{Garble}(C)$ $\xrightarrow{\quad GC \quad}$

Online

For $i \in \{1, \ldots, \overline{m}\}$ For $i \in \{1, \ldots, \overline{m}\}$
  For $j \in \{1, \ldots, t\}$ For $j \in \{1, \ldots, t\}$

$s_{i,j}^0, s_{i,j}^1 \xrightarrow{\hspace{2cm}}$ $\boxed{\text{OT}}$ $\xleftarrow{\hspace{2cm}} z_{i,j}$
$\xrightarrow{\hspace{2cm}} s_{i,j}^{z_{i,j}}$

Return $K^0 = (K_{1,1}^0, \ldots, K_{\overline{m},t}^0), K^1 = (K_{1,1}^1, \ldots, K_{\overline{m},t}^1)$ **Return** $K = (K_{1,1}, \ldots, K_{\overline{m},t}) = \mathsf{Eval}(GC, \mathbf{s})$

Fig. 13: $\mathrm{Sel_H} \to \mathrm{Comp_G}$ – Interface between the instantiation for $\mathrm{Sel_H}$ (Fig. 6) and $\mathrm{Comp_G}$ (Fig. 9).

4.1 Protocol Combinations

In this section, we describe all protocols resulting from valid combinations of sub-protocols presented in §3, and adapt them to the offline-online setting. Here, all operations dependent only of the input of the server $\mathcal{S}$'s decision tree are performed in the offline phase, whereas all operations that depend on the client $\mathcal{C}$'s input are performed in the online phase. Protocols that are efficient in the online phase are especially important in applications where the client uses a computationally weak device. Most of the computation can be done offline, when the device is idle, and the query itself triggers the online phase, which is performed more efficiently. We show that protocols based on garbling techniques are especially efficient in this setting, whereas the sub-protocols based on homomorphic encryption require the input directly at the start and therefore almost none of the computationally intense tasks can be performed offline. ABC denotes the protocol that uses $\mathrm{Sel_A}$ for selection, $\mathrm{Comp_B}$ for comparison and $\mathrm{Path_C}$ for path evaluation.

HHH: $\mathrm{Sel_H} + \mathrm{Comp_H} + \mathrm{Path_H}$ [TMZC17]. The state-of-the-art protocol using only homomorphic encryption for all the three sub-protocols was presented in [TMZC17], the comparison protocol of which was first used for private decision tree evaluation in [WFNL16]. We recapitulate it here as a combination of the homomorphic sub-protocol $\mathrm{Sel_H} + \mathrm{Comp_H}$ (Fig. 8) with $\mathrm{Path_H}$ (Fig. 10). The advantage of this protocol is that very few communication is necessary between the two parties and the selection happens along with the comparison. However, all computationally expensive cryptographic operations must be computed online, since all computations depend on the client's input vector.

HGH: $\mathrm{Sel_H} + \mathrm{Comp_G} + \mathrm{Path_H}$. An oblivious comparison protocol that can be used to output shares of the comparison bits is Yao's GC protocol is shown in $\mathrm{Comp_G}$ (Fig. 9). Here, m garbled comparison circuits can be precomputed in an offline phase. Thereafter, the client evaluates these in the online phase (using SIMD in parallel as described in §2.2) to retrieve the comparison color bits that are shares of the output bits. However, this comparison protocol requires a selection sub-protocol to choose m inputs to the m comparisons out of the client's $n \leq m$ inputs. In this protocol, we use $\mathrm{Sel_H}$ with Paillier or DGK encryption (cf. Fig. 6) in order to avoid nonlinear scaling. When combining these approaches, the result of $\mathrm{Sel_H}$ is not the same as the input of $\mathrm{Comp_G}$, so the interface described below and depicted in Fig. 13 is inserted to obliviously unblind the output. After the comparison, the path evaluation $\mathrm{Path_H}$ of [TMZC17] can be used as before (cf. Fig. 10).
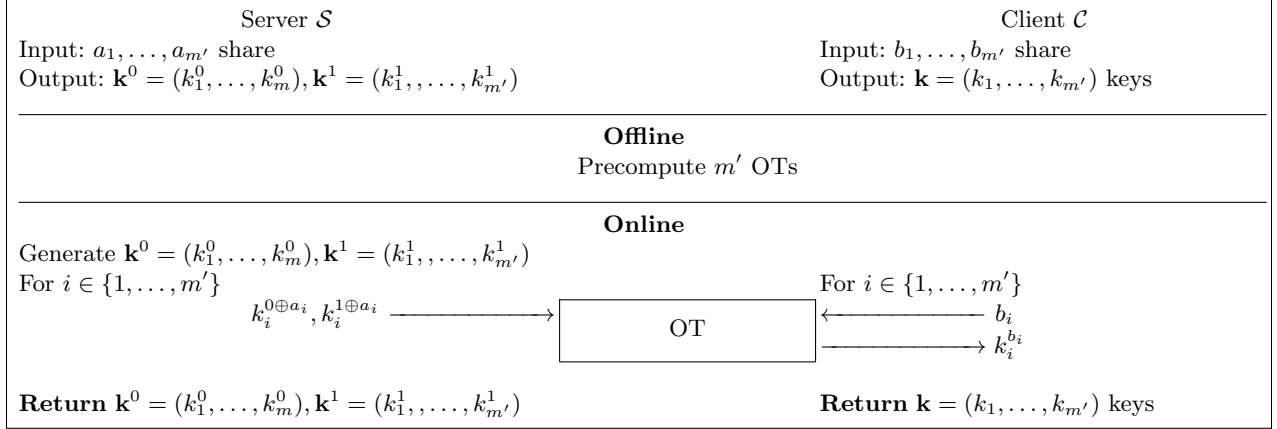
Server $\mathcal{S}$	Client $\mathcal{C}$
Input: $a_1, \ldots, a_{m'}$ share	Input: $b_1, \ldots, b_{m'}$ share
Output: $\mathbf{k}^0 = (k_1^0, \ldots, k_m^0), \mathbf{k}^1 = (k_1^1,, \ldots, k_{m'}^1)$	Output: $\mathbf{k} = (k_1, \ldots, k_{m'})$ keys

Offline
Precompute m' OTs

Online

Generate $\mathbf{k}^0 = (k_1^0, \ldots, k_m^0), \mathbf{k}^1 = (k_1^1,, \ldots, k_{m'}^1)$

For $i \in \{1, \ldots, m'\}$ For $i \in \{1, \ldots, m'\}$

$$k_i^{0 \oplus a_i}, k_i^{1 \oplus a_i} \longrightarrow \boxed{\text{OT}} \longleftarrow b_i$$
$$\longrightarrow k_i^{b_i}$$

Return $\mathbf{k}^0 = (k_1^0, \ldots, k_m^0), \mathbf{k}^1 = (k_1^1,, \ldots, k_{m'}^1)$ **Return** $\mathbf{k} = (k_1, \ldots, k_{m'})$ keys

Fig. 14: $\text{Comp}_H \to \text{Path}_G$ – Interface between the instantiation for Comp_H (Fig. 8) and Path_G (Fig. 11).

Combining homomorphic selection and garbled comparison (Sel$_H \to$ Comp$_G$, Fig. 13). The result of Sel_H in Fig. 6 is a set of blinded $(t + s)$-bit values for the client $\mathcal{C}$, and the blinding s-bit values for the server $\mathcal{S}$. In order to perform the comparison on t-bit values, $\mathcal{S}$ and $\mathcal{C}$ must unblind the result using a GC for subtracting each random value obliviously [BFK+09]. The subtractions are performed on t-bit values, since the most significant s bits of the blinded values can be dropped. The keys corresponding to the result are used in the comparison Comp_G (cf. Fig. 9). Overall, the $\overline{m}$ garbled subtraction circuits require $2\overline{m}t \cdot \tau_{\text{sym}}$ bits offline communication, $\overline{m}t \cdot \tau_{\text{sym}}$ bits for precomputing random OTs, and $2\overline{m}t \cdot \tau_{\text{sym}}$ bits online communication for the OTs.

GGH: Sel$_G$ + Comp$_G$ + Path$_H$. Instead of instantiating the selection with homomorphic encryption, one can use Sel_G, a selection network (with $\mathcal{O}(mt \log m)$ complexity) using garbled circuits (cf. Fig. 7). Here, most of the selection phase can be computed offline due to precomputed OTs and the independence of the GC from the client's inputs. Then, Comp_G (Fig. 9) can be directly performed, and the path evaluation sub-protocol Path_H of [TMZC17] (Fig. 10) is used as before. The advantage of this protocol is that though the offline communication is more than in the previous protocols due to the nonlinear selection network, it performs well in the online phase and due to the path evaluation phase, it depends only on the number of decision nodes m, since depth-padding is not required.

HGG: Sel$_H$ + Comp$_G$ + Path$_G$ [BFK+09]. Homomorphic selection Sel_H (Fig. 6) and garbled comparison Comp_G (Fig. 9) (with the interface from Fig. 13 between) can also be combined with the path evaluation technique Path_G (Fig. 11). This induces overhead in the number of decision nodes, since the depth-padded decision tree has a larger number of internal nodes $m' \geq m$ (cf. Fig. 12). However, this path evaluation protocol can more efficiently be transformed to an offline-online setting, since the GCs for comparison and the garbled decision tree can be sent offline.

GGG: Sel$_G$ + Comp$_G$ + Path$_G$ [BFK+09]. This protocol consists of sub-protocols that use garbling techniques: selection Sel_G (Fig. 7) and comparison Comp_G (Fig. 9) with GCs, and garbled path evaluation Path_G (Fig. 11). It has the highest communication, though most data can be sent in the offline phase. However, it is the most efficient online, even when large decision trees with many dummy nodes are considered.

Another advantage of this protocol is that it can be easily extended to provide security against a malicious client. Since the only messages sent by the client are in the OTs, one can use OT extension with security against malicious clients, e.g., [ALSZ15, KOS15, ALSZ17], which are only slightly less efficient than OT extension with passive security. This adds little overhead, whereas securing the other solutions against malicious clients needs more expensive conditional OTs or zero-knowledge proofs, e.g., as for HHH described in [TMZC17].

HHG: Sel$_H$ + Comp$_H$ + Path$_G$. Due to the worse computational efficiency of the homomorphic selection and comparison Sel_H + Comp_H (Fig. 8) and the high number of decision nodes m' for the garbled path

evaluation method Path_G (Fig. 11), this protocol combination is the least efficient. Moreover, it requires an interface of OTs described below and depicted in Fig. 14.

Combining homomorphic comparison and garbled path evaluation ($Comp_H{\to}Path_G$, Fig. 14). The result of $\text{Sel}_\text{H} + \text{Comp}_\text{H}$ (Fig. 8) is for both parties a share of the comparison bit $c_i = a_i \oplus b_i$ for all $i \in \{1, \ldots, m'\}$, while the input to Path_G (Fig. 11) is a key for all comparison bits for the client $\mathcal{C}$ and two keys for both possible bits for the server $\mathcal{S}$. After $\mathcal{S}$ generates these keys, $\mathcal{C}$ and $\mathcal{S}$ engage in a 1-out-of-2 OT protocol for each comparison bit (cf. §2.2). This conversion requires $2m'\tau_\text{sym}$ bit online and $m'\tau_\text{sym}$ bit offline communication.

4.2 Protocol extensions

In this section, we describe protocol extensions, including a natural modification of the HGG and GGG protocols, which enables the client to perform less computation and evaluate the decision tree in a similar manner as in plain evaluation, with only $\mathcal{O}(dt)$ cryptographic operations. We compare this method with the protocol of [JS18] that achieves the same asymptotic efficiency.

Path evaluation with computationally restricted client. A natural extension of HGG and GGG is to perform only the required d comparisons during the path evaluation phase as in the case of plain evaluation, instead of doing all m' comparisons in advance. This improves computation, though we cannot use SIMD operations for the comparisons anymore, since the GCs are evaluated sequentially along the evaluation path. Similarly, the client can decrypt only d (instead of m') homomorphically encrypted blinded values in HGG in Sel_H (cf. Fig. 6) before these comparisons. However, the communication remains unchanged, since the actual evaluation path taken needs to remain oblivious to the server.

Joye and Salehi present a similar protocol in [JS18]. In their protocol, full decision trees are utilized to hide the topology of the tree. In this protocol, a 1-out-of-2^i OT is performed at the ith level ($i \in \{1, \ldots, d\}$). However, the full tree can be replaced with the depth-padded tree described in §2.1, in which case 1-out-of-m' OTs are sufficient for levels where $m' < 2^i$. Though the communication of this protocol is better than that of HGG ($\mathcal{O}(n + m't)$) for small decision trees, its dependency on the depth ($\mathcal{O}(n + d(t + m'))$) makes it worse for larger examples such as our `boston` dataset. Moreover, it has $\mathcal{O}(d)$ online rounds instead of the constant number of 4 rounds provided by HGG. Therefore, we conclude that our HGG protocol is more efficient in most scenarios.

Other extensions. In §B.1, we describe modifications for widely used categorical variables, and in §B.2 for securely evaluating random forests to increase accuracy. Classification confidence values, i.e., the probability that a classification is correct, can also be appended to the classification labels in a straightforward manner.

5 Performance Evaluation

In this section, we compare the performance of all resulting protocols from §4. . We show the runtime and communication of all resulting combinations of the sub-protocols from §3. In our implementation, we instantiate all primitives corresponding to security level $\kappa = 128$, and statistical security parameter $s = 40$, and as in prior work, we set the bitlength to $t = 64$. τ_* denotes the ciphertext sizes: $\tau_\text{sym} = 128$ for symmetric-key encryption, $\tau_\text{Paillier} = 4096$ and $\tau_\text{DGK} = 2048$ for Paillier and DGK encryptions, resp. and $\tau_\text{ElGamal} = 514$ for ElGamal encryption using elliptic curves and point compression.

The underlying frameworks for our implementation are the `mcl` library[1] that includes a highly optimized lifted ElGamal implementation, and the ABY secure computation framework [DSZ15][2]. ABY implements the state-of-the-art optimizations of Yao's garbled circuit protocol described in §2.2. In addition, we implement an analogues technique to that of point-and-permute in order to avoid trial decryption of garbled nodes in oblivious path evaluation Path_G. We give more details on our implementation in §C.We present the complexities of the resulting protocols for the datasets in Tab. 3.

---

[1] `https://github.com/herumi/mcl`
[2] `https://github.com/encryptogroup/ABY`

(a) Offline communication in KBytes. (b) Online communication in KBytes. (c) Total communication in KBytes.
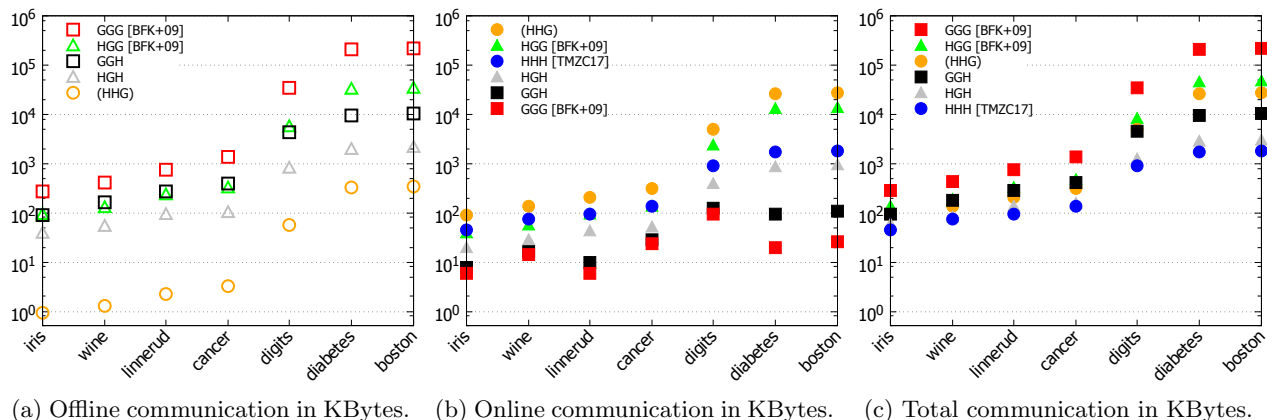
Fig. 15: Communication of protocols using the example datasets from Tab. 3. Note that the y-axis is in logarithmic scale.
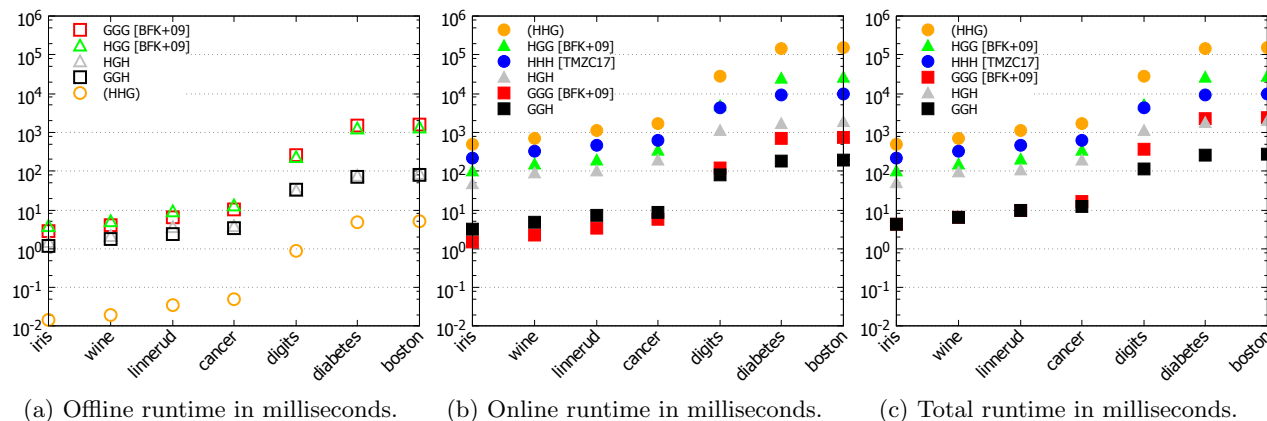

(a) Offline runtime in milliseconds. (b) Online runtime in milliseconds. (c) Total runtime in milliseconds.

Fig. 16: Runtime of protocols using the example datasets from Tab. 3. Note that the y-axis is in logarithmic scale.

Communication. The offline, online, and total communication of the protocols is given in Fig. 15a, Fig. 15b, and Fig. 15c, resp. We observe that garbling techniques allow for precomputation and offline communication, but in total require more communication. Homomorphic encryption-based methods have less communication, however, almost all expensive computation and all communication happens in the online phase.

Runtime. For our benchmarks we use two machines equipped with an Intel Core i7-4790 CPU @ 3.6 GHz and 32 GB of RAM that support Intel's AES-NI for fast AES operations. Our benchmarks are run in a LAN setting with 1 Gbit/s and 0.5 ms latency. Runtimes are reported from an average of 10 executions. In our experiments relying on oblivious transfers, we neglect the costs for the base OTs, since they are a one-time expense that can be re-used over multiple protocol executions. The offline runtimes are given in Fig. 16a, the online runtimes are given in Fig. 16b, while the total runtimes are depicted in Fig. 16c. We can see that homomorphic encryption-based methods (Sel_H and $Comp_H$) perform an order of magnitude worse than their garbling technique-based alternatives (Sel_G and $Comp_G$). However, we observe the advantage of the homomorphic encryption-based path evaluation technique ($Path_H$), where the number of decision nodes m is unchanged, whereas in the path evaluation method using a garbled decision tree ($Path_G$) depth-padding increases the number of decision nodes to $m \leq m' \leq m(1 + d)$, which can be significantly larger than m
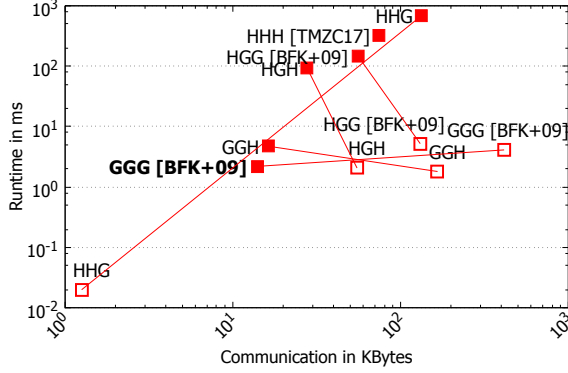
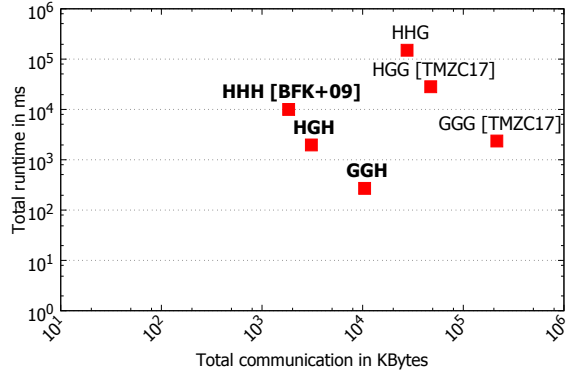(a) Tradeoff for small dataset `wine`.
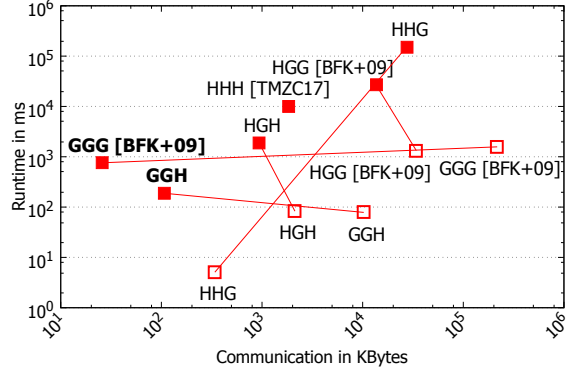
(b) Tradeoff for large dataset `boston`.

Fig. 17: Tradeoff between communication (x-axis) and runtime (y-axis). Note that both axes are in logarithmic scale. The figures show the offline (unfilled squares) and online (filled squares) complexities (connected with a line). Pareto points, i.e., protocols where one property (computation or communication) cannot be improved without the other property becoming worse, are marked in bold.
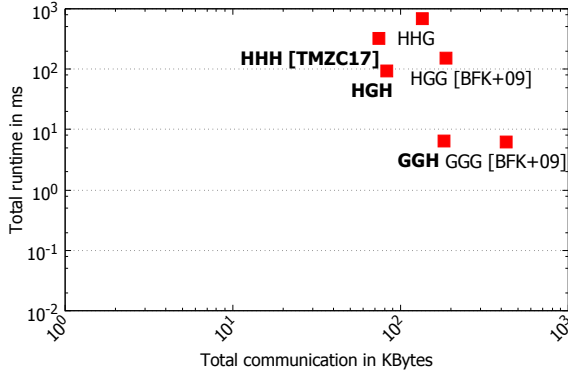
(a) Tradeoff for small dataset `wine`.

(b) Tradeoff for large dataset `boston`.

Fig. 18: Tradeoff between total communication (x-axis) and runtime (y-axis). Note that both axes are in logarithmic scale. Pareto points are marked in bold.

(cf. Fig. 12 and Tab. 3). Our protocol GGH has the fastest total and online runtime of below 1 second for our largest example.

Tradeoff. In order to show the communication (x-axis) vs. computation (y-axis) tradeoff of all resulting protocols, we give figures for two example datasets: we choose a small example dataset `wine`, and our largest example dataset `boston`.

The number of input features n is usually small for all datasets, whereas a growing number of decision nodes affects the efficiency of our sub-protocols significantly. We depict in Fig. 17a and Fig. 17b the tradeoff between offline and online runtime and communication for all protocols for the `wine` and `boston` datasets, respectively, while Fig. 18a and Fig. 18b depict the corresponding total complexities.

The larger the dataset, the more dummy nodes are introduced during depth-padding (cf. §2.1) in $\mathrm{Path_G}$. Therefore, the size of the dataset largely affects the protocols that use $\mathrm{Path_G}$ for path evaluation. Moreover, the selection network in $\mathrm{Sel_G}$ has non-linear size $\mathcal{O}(\overline{m}\log\overline{m})$ in $\overline{m}$, which implies a significant overhead in offline communication. This gap can be observed on the figures: the protocols using garbling techniques for

17

path evaluation lose their performance advantage with growing number of dummy nodes, and the protocols using GC-based selection techniques become less practical due to the largely increased amount of offline communication for transmitting the selection network (which implies storage requirements as well). For instance, the largest decision tree trained on the `boston` dataset requires around 1.8 MBytes of communication with HHH, 3 MBytes with HGH, 9 MBytes with GGH, and more than 200 MBytes with GGG. The designer of an application using private decision tree evaluation can consider these tradeoff and choose the best suited protocol.

6   Concrete Improvements and Recommendations

We investigate the concrete tradeoffs between our identified hybrid protocols GGH and HGH from §4.1 and the state-of-the-art protocols HHH of [TMZC17] optimized for communication and GGG of [BFK$^+$09] optimized for online computation. We show that the state-of-the-art protocols that exclusively use one of the two paradigms HHH with additively homomorphic encryption or GGG with garbling techniques can be replaced by our hybrid protocols HGH and GGH that provide a more reasonable tradeoff for larger decision trees.

In Fig. 17 and Fig. 18, we emphasize pareto points for online and total tradeoffs, respectively, i.e., we bold the protocols where there is no way to improve one property without the other property becoming worse.

Tradeoff with HHH [TMZC17]. The improvement in total and online runtime of GGH over HHH is more than an order of magnitude for our examples with increased total communication by 2-5x. HGH improves over HHH in total and online runtime by 3-5x with slightly increased total communication. The online communication is better for both GGH and HGH than for HHH.

Tradeoff with GGG [BFK$^+$09]. GGH improves over GGG for larger datasets due to the fact that it does not need any padding for the decision nodes. For small datasets, GGG is better than GGH in online runtime, but gets worse for both online and total runtimes with large datasets by almost an order of magnitude. GGH also provides an improvement in total communication by 3-20x. HGH improves even more for our datasets over the communication of GGG by 5-73x. The total (online) runtime of HGH is larger by 3-11x (2-32x), but this is decreasing with larger datasets due to the increasing number of decision nodes. The total runtime gets better for our biggest examples for HGH than for GGG.

Recommendations. We identify the following aspects to take into consideration when deciding which protocol to use for an application: the *dataset size*, and the boundaries on *communication and computational power* (e.g., network throughput and client storage capacity). When the client has high storage capacity, protocols GGG [BFK$^+$09] or GGH (cf. §4.1) provide the best online performance, depending on the size of the dataset. The former has a slightly more efficient online phase, but higher communication due to its selection protocol (cf. §3.1). When the client is computationally bounded and has little storage capacity, our protocols HGH and GGH from §4.1 provide the best solutions.

The state-of-the-art protocol of [TMZC17], i.e., HHH, has the lowest total communication, but uses computationally heavy operations that depend almost entirely on the client's input and hence do not allow for much offline precomputation. Our protocols HGH or GGH from §4.1 that combine its path evaluation step with garbling techniques yield more than an order of magnitude faster runtimes while only slightly increasing communication.

7   Related Work

Private decision tree evaluation was firstly considered in [BPSW07], with application to private evaluation of remote diagnostic programs. A protocol based on homomorphic encryption where the server evaluates a branching program on the client's encrypted input was proposed in [IP07]. This has been improved in [BFK$^+$09,BFL$^+$11] where a protocol based on mainly symmetric-key operations was proposed. Bost et al. use additively homomorphic encryption to evaluate the decision tree expressed as a polynomial [BPTG15]. Recently, Wu et al. [WFNL16], Tai et al. [TMZC17] and Joye and Saleji [JS18] improved the state-of-the-art

of private decision tree evaluation protocols. These works rely on additively homomorphic encryption using the Diffie-Hellman assumption and present protocols that achieve security against semi-honest adversaries or malicious clients. Tai et al. [TMZC17] eliminate the exponential dependency on the depth of the tree that was present in [WFNL16] by representing decision trees as linear functions. This implies enormous improvement when large decision trees are considered which, in practice, are usually not very dense, and therefore, we use this protocol as HHH. Wu et al.'s protocol [WFNL16] would perform better than HHH for circuits with $m \sim 2^d$ [TMZC17]. However, already for a very dense tree with $m = 2^{d-2}$, HHH has about half the communication compared to [WFNL16], whereas the necessary computation is around the same. Therefore, for most real-world decision trees with reasonable sizes, HHH performs better.

The protocols of [WFNL16] and [TMZC17] use the DGK comparison protocol [DGK07]. An alternative solution that performs well for small bitlengths t of the features and inputs have been proposed using fully homomorphic encryption in [LZS18].

A different solution for evaluating full private decision trees using the so-called commodity-based model was proposed in [CDH+17], where correlated randomness is distributed by a trusted authority to the computing parties or pre-computed during the offline phase, which are used in the online phase.

In concurrent and independent related work Tueno et al. [TKK19] improve the state-of-the-art by representing the decision tree as an array, and implementing oblivious array indexing. For this, they use either garbled circuits, oblivious transfer or oblivious RAM (ORAM), the latter of which results in a protocol with sub-linear complexity in the size of the decision tree. Their protocols require dt comparisons as that of [JS18] and our protocol extension for GGG and HGG §4.2, but require at least $\mathcal{O}(d)$ rounds of communication [TKK19, Tab. 2].

Alternatively, private decision tree evaluation can also be solved using generic private function evaluation (PFE) protocols such as [KM11, MS13, MSS14]. However, these solutions utilize Boolean circuits as the underlying representation of the functionality. Transforming the decision tree into a Boolean circuit would imply additional unnecessary overhead on the size of the function (i.e., $\mathcal{O}(tm \log m)$ gates for the selection of inputs as in $\mathsf{Sel_G}$ in §3.1). Secure evaluation of universal circuits (UCs) is an equivalent solution for PFE [Val76, KS08b, KS16, GKS17]. Then, UC- and OT-based PFE protocols [KS08b, MS13, KS16, GKS17] have computation and communication complexity $\mathcal{O}((tm \log m) \log(tm \log m))$, and HE-based PFE protocols [KM11, MS13, MSS14] need $\mathcal{O}(tm \log m)$ computation and communication. These are larger than the complexities achieved by protocols designed specifically for DTs (cf. Tabs. 5-6).

There are alternative approaches for classification based on machine learning, such as deep neural networks, that can be evaluated in a private manner [SS08, BFL+11, LJLA17, MZ17, RWT+18, JVC18, BFR+18, BDK+18]. Private evaluation of machine learning models incur a natural overhead, but they can perform classifications of real-world datasets in the order of seconds.

Acknowledgements

References

ALSZ13. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Computer and Communications Security (CCS'13)*, pages 535–548. ACM, 2013.

ALSZ15. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015.

ALSZ17. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *J. Cryptology*, 30(3):805–858, 2017.

AM18. Bushra A. AlAhmadi and Ivan Martinovic. Malclassifier: Malware family classification using network flow sequence behaviour. In *2018 APWG Symposium on Electronic Crime Research (eCrime'18)*, pages 1–13. IEEE, 2018.

Bar15. Jeff Barr. Amazon machine learning – make data-driven decisions at scale. `aws.amazon.com/blogs/aws/amazon-machine-learning-make-data-driven-decisions-at-scale`, 2015. Accessed: 2018-08-19.

BDK+18. Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. Hycc: Compilation of hybrid protocols for practical secure computation. In *ACM Computer and Communications Security (CCS'18*, pages 847–861. ACM, 2018.

Bea95. Donald Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

BFK+09. Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *LNCS*, pages 424–439. Springer, 2009.

BFL+11. Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.

BFR+18. Ferdinand Brasser, Tommaso Frassetto, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, and Christian Weinert. Voiceguard: Secure and private speech processing. In *Interspeech 2018*, pages 1303–1307. ISCA, 2018.

BHKR13. Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.

Big18. Inc. BigML. Machine learning made easy, beautiful and understandable. `https://bigml.com/`, 2018. Accessed: 2018-08-24.

BJL12. Dan Bogdanov, Roman Jagomägis, and Sven Laur. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Pacific Asia Workshop on Intelligence and Security Informatics (PAISI'12)*, volume 7299 of *LNCS*, pages 112–126. Springer, 2012.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *ACM Symposium on Theory of Computing (STOC'90)*, pages 503–513. ACM, 1990.

BPSW07. Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *ACM Computer and Communications Security (CCS'07)*, pages 498–507. ACM, 2007.

BPTG15. Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.

BS09. Justin Brickell and Vitaly Shmatikov. Privacy-preserving classifier learning. In *Financial Cryptography and Data Security (FC'09)*, volume 5628 of *LNCS*, pages 128–147. Springer, 2009.

BSR18. Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *USENIX Security Symposium'18*, pages 169–185. USENIX Association, 2018.

CDH+17. Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey C. Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, , 2017.

CO18.   Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party compu-
             tation, 2018.
DCBA14.   Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. Do we need hun-
             dreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*,
             15(1):3133–3181, 2014.
DGK07.   Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions.
             In *Australasian Conference on Information Security and Privacy (ACISP'07)*, volume 4586 of *LNCS*,
             pages 416–430. Springer, 2007.
DGK08.   Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Homomorphic encryption and secure comparison.
             International Journal of Applied Cryptography, 1(1):22–31, 2008.
DGK09.   Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'efficient and secure comparison
             for on-line auctions'. *IJACT*, 1(4):323–324, 2009.
DJ01.   Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's
             probabilistic public-key system. In *Public Key Cryptography (PKC'01)*, volume 1992 of *LNCS*, pages
             119–136. Springer, 2001.
DSZ15.   Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol
             secure two-party computation. In *Network and Distributed System Security Symposium (NDSS'15)*. The
             Internet Society, 2015.
ElG85.   Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In
             Advances in Cryptology – CRYPTO'85, volume 196 of *LNCS*, pages 10–18. Springer, 1985.
FW12.   Pui Kuen Fong and Jens H. Weber-Jahnke. Privacy preserving decision tree learning using unrealized
             data sets. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):353–364, 2012.
GBC$^+$97.   Michael D. Garris, James L. Blue, Gerald T. Candela, Patrick J. Grother, Stanley Janet, and Charles L.
             Wilson. NIST form-based handprint recognition system (release 2.0). *Interagency/Internal Report (NI-
             STIR) - 5959*, 1997.
GKG$^+$18.   Srishti Gupta, Abhinav Khattar, Arpit Gogia, Ponnurangam Kumaraguru, and Tanmoy Chakraborty.
             Collective classification of spam campaigners on twitter: A hierarchical meta-path based approach. In
             World Wide Web Conference on World Wide Web (WWW'18), pages 529–538. ACM, 2018.
GKS17.   Daniel Günther, Ágnes Kiss, and Thomas Schneider. More efficient universal circuit constructions. In
             Advances in Cryptology – ASIACRYPT'17, volume 10625 of *LNCS*, pages 443–470. Springer, 2017.
HEK12.   Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than
             custom protocols? In *Network and Distributed System Security Symposium (NDSS'12)*. The Internet
             Society, 2012.
IK17.   Aleksandar Ilic and Oleksandr Kuvshynov. Evaluating boosted decision trees for billions of
             users. https://code.facebook.com/posts/975025089299409/evaluating-boosted-decision-trees-
             for-billions-of-users, 2017. Accessed: 2018-08-19.
IKNP03.   Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In
             Advances in Cryptology – CRYPTO'03, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.
IP07.   Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptog-
             raphy Conference (TCC'07)*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.
JS18.   Marc Joye and Fariborz Salehi. Private yet efficient decision tree evaluation. In *Data and Applications
             Security and Privacy (DBSec'18)*, volume 10980 of *LNCS*, pages 243–259. Springer, 2018.
JSD$^+$18.   Mika Juuti, Sebastian Szyller, Alexey Dmitrenko, Samuel Marchal, and N. Asokan . PRADA: protecting
             against DNN model stealing attacks. *CoRR*, abs/1805.02628, 2018.
JVC18.   Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency frame-
             work for secure neural network inference. In *USENIX Security Symposium'18*, pages 1651–1669. USENIX
             Association, 2018.
KM11.   Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In
             Advances in Cryptology – ASIACRYPT'11, volume 7073 of *LNCS*, pages 556–571. Springer, 2011.
KMAM17.   Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in
             MLaaS paradigm. *CoRR*, abs/1711.07221, 2017.
KOS15.   Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead.
             In *Advances in Cryptology – CRYPTO'15*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.
KS08a.   Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications.
             In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of
             LNCS, pages 486–498. Springer, 2008.

KS08b.   Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC'08)*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008.

KS16.   Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In *Advances in Cryptology – EUROCRYPT'16*, volume 9665 of *LNCS*, pages 699–728. Springer, 2016.

KSS09.   Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security (CANS'09)*, volume 5888 of *LNCS*, pages 1–20. Springer, 2009.

Lic18.   Moshe Lichman. UCI machine learning repository. `https://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science, 2018. Accessed: 2018-08-24.

LJLA17.   Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM Computer and Communications Security (CCS'17)*, pages 619–631. ACM, 2017.

LP00.   Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO'00*, volume 1880 of *LNCS*, pages 36–54. Springer, 2000.

LP02.   Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

LZS18.   Wenjie Lu, Jun-Jie Zhou, and Jun Sakuma. Non-interactive and output expressive private comparison from homomorphic encryption. In *Asia Conference on Computer and Communications Security (AsiaCCS'18)*, pages 67–74. ACM, 2018.

MAAG15.   Michael J. Mayhew, Michael Atighetchi, Aaron Adler, and Rachel Greenstadt. Use of machine learning in big data analytics for insider threat detection. In *IEEE Military Communications Conference (MILCOM'15)*, pages 915–922. IEEE, 2015.

Mic18.   Microsoft. Azure machine learning studio. `https://azure.microsoft.com/`, 2018. Accessed: 2018-08-24.

MLJ17.   Inc. MLJAR. MLJAR: Machine learning for all. `https://mljar.com/`, 2016-2017. Accessed: 2018-08-24.

MS13.   Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology – EUROCRYPT'13*, volume 7881 of *LNCS*, pages 557–574. Springer, 2013.

MSS14.   Payman Mohassel, Seyed Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. In *Advances in Cryptology – ASIACRYPT'14*, volume 8874 of *LNCS*, pages 486–505. Springer, 2014.

MZ17.   Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (S&P'17)*, pages 19–38. IEEE, 2017.

NIZ+16.   Ana Nika, Asad Ismail, Ben Y. Zhao, Sabrina Gaito, Gian Paolo Rossi, and Haitao Zheng. Understanding and predicting data hotspots in cellular networks. *Mobile Networks and Applications (MONET)*, 21(3):402–413, 2016.

OTGT18.   Rebekah Overdorf, Carmela Troncoso, Rachel Greenstadt, and Damon McCoy . Under the underground: Predicting private interactions in underground forums. *CoRR*, abs/1805.04494, 2018.

Pai99.   Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

PMG+17.   Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS'17), 2017*, pages 506–519. ACM, 2017.

RG16.   Carl Rabeler and Craig Guyer. Microsoft decision trees algorithm. `https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/microsoft-decision-trees-algorithm`, 2016. Accessed: 2018-08-19.

RMD18.   Alejandro Rago, Claudia Marcos, and J. Andres Diaz-Pace. Using semantic roles to improve text classification in the requirements domain. *Language Resources and Evaluation*, 52(3):801–837, 2018.

RWT+18.   M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications, 2018.

Ser18.   Amazon Web Services. Data privacy. `https://aws.amazon.com/compliance/data-privacy-faq`, 2018. Accessed: 2018-08-19.

sld17.   scikit-learn developers. scikit-learn – machine learning in python. `http://scikit-learn.org/stable/modules/tree.html`, 2017. Accessed: 2018-08-22.

SS08.   Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *Information Security and Cryptology (ICISC'08)*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

SSSS17. Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (S&P'17)*, pages 3–18. IEEE Computer Society, 2017.

TASF09. Ajay Kumar Tanwani, M. Jamal Afridi, M. Zubair Shafiq, and Muddassar Farooq. Guidelines to select machine learning scheme for classification of biomedical datasets. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO'09)*, volume 5483 of *LNCS*, pages 128–139. Springer, 2009.

Ten98. Michel Tenenhaus. *La régression PLS: théorie et pratique*. Editions technip, 1998.

TKK19. Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2019(1):, 2019.

TMZC17. Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *European Symposium on Research in Computer Security (ESORICS'17)*, volume 10493 of *LNCS*, pages 494–512. Springer, 2017.

TZJ$^+$16. Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium'16*, pages 601–618. USENIX, 2016.

Val76. Leslie G. Valiant. Universal circuits (preliminary report). In *ACM Symposium on Theory of Computing (STOC'76)*, pages 196–203. ACM, 1976.

VC05. Jaideep Vaidya and Chris Clifton. Privacy-preserving decision trees over vertically partitioned data. In *Data and Applications Security (DBSec'05)*, volume 3654 of *LNCS*, pages 139–152. Springer, 2005.

VCKP08. Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A. Scott Patterson. Privacy-preserving decision trees over vertically partitioned data. *TKDD*, 2(3):14:1–14:27, 2008.

Wak68. Abraham Waksman. A permutation network. *Journal of the Association for Computing Machinery*, 15(1):159–163, 1968.

WFNL16. David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016(4):335–355, 2016.

WGC18. Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and private neural network training. *Cryptology ePrint Archive*, 2018/442:442, 2018.

Wis18. Wise.io. Machine learning for the industrial internet of things. `wise.io`, 2018. Accessed: 2018-08-24.

Yao82. Andrew C.-C. Yao. Protocols for secure computations (extended abstract). In *Foundations of Computer Science (FOCS'82)*, pages 160–164. IEEE, 1982.

Yao86. Andrew C.-C. Yao. How to generate and exchange secrets (extended abstract). In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

YGL17. Fengpeng Yuan, Xianyi Gao, and Janne Lindqvist. How busy are you?: Predicting the interruptibility intensity of mobile users. In Gloria Mark, Susan R. Fussell, Cliff Lampe, m. c. schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor, editors, *Conference on Human Factors in Computing Systems (CHI'17).*, pages 5346–5360. ACM, 2017.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

A Complexities of Sub-protocols

In this section, we describe the complexities of all sub-protocols described in §3.

Communication. The communication of each sub-protocol is presented in Tab. 5. For all phases we observe a direct tradeoff: The homomorphic encryption-based methods have lower communication, while in case of garbling techniques, most communication can be shifted offline, so these achieve the lowest online communication.

Computation. The asymptotic computational complexity of each sub-protocol is shown in Tab. 6. We observe that most computationally intense tasks can be shifted offline for the protocols based on garbling techniques, while almost all operations are performed online when using homomorphic encryption. Unfortunately, even when using garbling techniques, the client needs to perform most cryptographic operations online, since he plays the role of the evaluator (cf. 2.2). In these techniques, however, the client requires less computational power than using homomorphic encryption.

B Extensions

In this section, we present extensions to Comp_G and Path_G presented in §3, which directly apply to the resulting protocols that use those as sub-protocols in §4.

B.1 Categorical Variables

Since not every member of a feature vector is a numerical variable, we discuss the useful extension of the comparison phase in order to add support for categorical variables. In fact, a considerable portion of real datasets are a mixture of numerical and categorical variables. We measured the ratio of datasets containing categorical variables in the UCI machine learning repository [Lic18], and found that 33% of datasets contain categorical variables. For instance, a feature vector for the `breast cancer` dataset has the feature `breast quadrant` that specifies the position of the tumor in the patient's breast, i.e. it takes one of the following five values: `left-up`, `right-up`, `left-low`, `right-low` or `central`. At the end of the protocol, the patient's profile is classified as no-recurrence-event or recurrence-event that determines the possibility of cancer recurrence.

Given categorical variable $x_j \in U = \{u_1, \ldots, u_{|U|}\}$, the decision criteria is a set membership test of the form $x_j \in U'$, for some $U' \subset U$ ($j \in \{1, \ldots, n\}$). Wu et al. supply the $\text{Sel}_\text{H} + \text{Comp}_\text{H}$ subprotocol with the extension to handle categorical variables in [WFNL16]. We briefly describe a private approach to evaluate the decision nodes that operate over categorical variables when using Comp_G. We notice that for testing

Sub-protocol	Fig.	Offline	Online
Sel_H Paillier	Fig. 6	-	$n + \frac{\overline{m}}{t+s} \cdot \tau_{\text{Paillier}}$
Sel_H DGK	Fig. 6	-	$(n + \overline{m}) \cdot \tau_{\text{DGK}}$
Sel_G	Fig. 7)	$2t \cdot \tau_{\text{sym}} \cdot (n + S^n_{\overline{m} \geq n})$	$2 \cdot \tau_{\text{sym}} \cdot nt$
$\text{Sel}_\text{H} + \text{Comp}_\text{H}$	Fig. 8	-	$(n + \overline{m}) \cdot t \cdot \tau_{\text{ElGamal}}$
Comp_G	Fig. 9	$2\overline{m}t \cdot \tau_{\text{sym}}$	-
Path_H	Fig. 10	-	$(3m + 2) \cdot \tau_{\text{ElGamal}}$
Path_G	Fig. 11	$2m' \cdot (\tau_{\text{sym}} + \log_2(m' + 1))$	-
$\text{Sel}_\text{H} \rightarrow \text{Comp}_\text{G}$	Fig. 13	$3\overline{m}t \cdot \tau_{\text{sym}}$	$2\overline{m}t \cdot \tau_{\text{sym}}$
$\text{Comp}_\text{H} \rightarrow \text{Path}_\text{G}$	Fig. 14	$m' \cdot \tau_{\text{sym}}$	$2m' \cdot \tau_{\text{sym}}$

Table 5: Offline and online communication of all sub-protocols, where n, t, m, m' and $\overline{m}$ denote the number input features, the bit length of the features and thresholds, the number of decision nodes, padded decision nodes and either of the former two, respectively. $S^n_{\overline{m} \geq n}$ denotes the size of the selection network that select m bits from n bits. τ_{sym}, τ_{DGK}, τ_{Paillier} and τ_{ElGamal} denote the size of the ciphertexts in the respective encryption schemes. Note that $m \leq m' \leq m + md$.

Sub-protocol	Fig.	Offline Server	Offline Client	Online Server	Online Client
Sel_H Paillier	Fig. 6	-	-	$\mathcal{O}(\overline{m}t)\nu_\text{Paillier}$	$\mathcal{O}(nt+\overline{m})\nu_\text{Paillier}$
Sel_H DGK	Fig. 6	-	-	$\mathcal{O}(\overline{m}t)\nu_\text{DGK}$	$\mathcal{O}((n+\overline{m})t)\nu_\text{DGK}$
Sel_G	Fig. 7	$\mathcal{O}(t\overline{m}\log\overline{m})\nu_\text{sym}$	$\mathcal{O}(nt)\nu_\text{sym}$	-	$\mathcal{O}(t\overline{m}\log\overline{m})\nu_\text{sym}$
$\text{Sel}_\text{H}+\text{Comp}_\text{H}$	Fig. 8	-	-	$\mathcal{O}(t\overline{m})\nu_\text{ElGamal}$	$\mathcal{O}((n+\overline{m})t)\nu_\text{ElGamal}$
Comp_G	Fig. 9	$\mathcal{O}(\overline{m}t)\nu_\text{sym}$	-	-	$\mathcal{O}(\overline{m}t)\nu_\text{sym}$
Path_H	Fig. 10	-	-	$\mathcal{O}(md)\nu_\text{ElGamal}$	$\mathcal{O}(m)\nu_\text{ElGamal}$
Path_G	Fig. 11	$\mathcal{O}(m')\nu_\text{sym}$	-	-	$\mathcal{O}(m')\nu_\text{sym}$
$\text{Sel}_\text{H}\rightarrow\text{Comp}_\text{G}$	Fig. 13	$\mathcal{O}(\overline{m}t)\nu_\text{sym}$	$\mathcal{O}(\overline{m}t)\nu_\text{sym}+\alpha$	-	$\mathcal{O}(\overline{m}t)\nu_\text{sym}$
$\text{Comp}_\text{H}\rightarrow\text{Path}_\text{G}$	Fig. 14	$\mathcal{O}(m')\nu_\text{sym}$	$\mathcal{O}(m')\nu_\text{sym}$	-	-

Table 6: The number of cryptographic operations in the offline and online phases of all sub-protocols, with notations as in Tab. 5. ν_sym, ν_DGK, ν_Paillier and ν_ElGamal denote the cost of an operation (i.e., encryption, decryption or addition) in symmetric, DGK, Paillier and lifted ElGamal encryption, respectively.

the set inclusion, the server can create the $|U|$-bit masking value Mask based on U' such that $\mathsf{Mask}_i = 1$ if $u_i \in U'$, and 0 otherwise. This representation was used for private set intersection (PSI) in [HEK12]. Accordingly, for $x_j = u_i$, the client inputs the transformed $|U|$-bit variable x'_j which has only one *set* bit at position i. Then the original decision criteria $x_j \in U'$ can be evaluated by the Boolean circuit which implements $x'_j \wedge \mathsf{Mask} = x'_j$.

B.2 Random Forests

Random forests improve the accuracy of decision trees by aggregating the result of $n \geq 2$ decision trees trained on different random subsets of features. Before evaluating the n decision trees, in order to hide all information about the subset of features, all decision tree evaluations need to look the same from the client's point of view. Therefore, dummy comparisons are necessary to pad the decision trees to the maximal m or m' value.

Additionally, the aggregation step depending on the aggregation function has to be implemented. Wu et al. provide a method that allows for any affine functions of the individual results using additive secret sharing [WFNL16], which can be generalized to the path evaluation method of [TMZC17], i.e., Path_H (cf. Fig. 10). For Path_G (cf. Fig. 11) [BFK+09], instead of returning the classification results to the client, he can obtain keys corresponding to these. These can then be used to securely evaluate the garbled circuit corresponding to the aggregation function if the server uses these keys when generating it. The complexity of this step depends on the aggregation function.

C Implementation Choices

We describe our implementation choices used in §5.

Sel_H (Fig. 6) The ABY framework includes an implementation for both Paillier and DGK encryptions (cf. §2.2). The ciphertext lengths for 128-bit security are $\tau_\text{Paillier} = 4096$ and $\tau_\text{DGK} = 2048$. However, Paillier encryption allows for ciphertext packing which can reduce the number of ciphertexts and decryptions. We utilize ABY to build two versions of Sel_H, and conclude the advantage of the Paillier encryption both in runtime and communication.

Sel_G (Fig. 7) We implement the garbled selection network introduced in [KS08a] based on the implementation of Waksman's permutation network [Wak68] in ABY. We use fixed-key AES for symmetric-key encryption, thus $\tau_\text{sym} = 128$.

$\text{Sel}_\text{H}+\text{Comp}_\text{H}$ (Fig. 8) We use the `mcl` library to implement protocols using lifted ElGamal encryption over elliptic curve (EC) `secp256k1` with 256-bit key and 128-bit security, as Tai et al. [TMZC17]. This library has a highly optimized lifted EC-ElGamal implementation and supports point compression, i.e.,

an elliptic curve point can be expressed in $256 + 1$ bits. Since an ElGamal ciphertext consists of two EC points, $\tau_{\mathrm{ElGamal}} = 514$.

Comp$_\mathbf{G}$ (Fig. 9) We use comparison circuits in ABY using SIMD operations to enhance the concrete performance of this sub-protocol.

Path$_\mathbf{H}$ (Fig. 10) We implement the path evaluation phase using lifted EC-ElGamal encryption over the same EC and library as for $\mathrm{Sel_H} + \mathrm{Comp_H}$.

Path$_\mathbf{G}$ (Fig. 11) We implement garbled decision tree path evaluation in the ABY framework using two AES encryptions with AES-NI.