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 constant-round 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 input to the decision tree. Moreover, the model may leak information about the training data used for its generation, which is a valuable asset of the server. The server’s goal is to offer the 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 works [TZJ+16, PMG+17, SSSS17, JSD+18] have 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, KMAM18, 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 while violating the privacy of neither the resulting model nor the customers’ input features. Most machine learning APIs such as BigML [Big18], MLJAR [MLJ17], 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 campaigns [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 [OTGM18]. 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 constant-round 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 constant-round 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 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 Sel\textsubscript{H}/Sel\textsubscript{G} (§3.1), Comp\textsubscript{H}/Comp\textsubscript{G} (§3.2), and Path\textsubscript{H}/Path\textsubscript{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.

Related work (§7) and open questions. Finally, we discuss our results in relation to related work. 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 future work, one could study and extend the security of hybrid protocols against malicious clients.

2 Preliminaries

In this section, we detail 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 value.

Table 2: Notations used throughout the paper.
A decision tree has a threshold vector $y = \{y_1, \ldots, y_m\}$ and receives an $n$-dimensional feature vector $x = \{x_1, \ldots, x_n\}$ as input. At the $j$th decision node ($j \in \{1, \ldots, m\}$), a comparison between an $x_i$ and $y_j$ takes place ($i \in \{1, \ldots, n\}$), where $\sigma : m \to n$ denotes the mapping for input selection. Decision tree evaluation $T(x)$ means evaluating the comparisons at each decision node of a path, the result of which defines the branch taken next. When a leaf node is reached on the path, the corresponding classification 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 $\sum_{i=1}^{m-1} i$, i.e., the number of decision nodes in the depth-padded tree is $m \leq m' \leq \frac{3}{2} m(m + 1)$. From here on, we use $m$ whenever both $m$ or $m'$ can be used in a protocol.

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, resp., as well as the boston dataset that includes house prices, 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 linerud 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 our identified protocol combinations. 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 $b$. OT generally requires expensive public-key operations [IR89]. However, a small (symmetric security parameter $\kappa$) number of base OTs can be extended to any polynomial number of OTs efficiently using only symmetric-key operations [IKNP03]. Improvements to such OT extension protocols have been presented 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 can precompute parts of the protocol before providing their 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
Yao’s garbled circuit (GC) protocol \cite{Yao82, Yao86} is one of the most widely used which can be efficiently computed using the double-and-add method.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$d$</th>
<th>$m$</th>
<th>$m'$</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>19</td>
<td>UCI \cite{Lic18}</td>
</tr>
<tr>
<td>wine</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>26</td>
<td>UCI \cite{Lic18}</td>
</tr>
<tr>
<td>linnerud</td>
<td>3</td>
<td>6</td>
<td>19</td>
<td>47</td>
<td>\cite{Ten98}</td>
</tr>
<tr>
<td>(breast) cancer</td>
<td>12</td>
<td>7</td>
<td>21</td>
<td>66</td>
<td>UCI \cite{Lic18}</td>
</tr>
<tr>
<td>digits</td>
<td>47</td>
<td>15</td>
<td>168</td>
<td>1,161</td>
<td>UCI \cite{Lic18}</td>
</tr>
<tr>
<td>diabetes</td>
<td>10</td>
<td>28</td>
<td>393</td>
<td>6,432</td>
<td>UCI \cite{Lic18}</td>
</tr>
<tr>
<td>boston</td>
<td>13</td>
<td>30</td>
<td>425</td>
<td>6,768</td>
<td>UCI \cite{Lic18}</td>
</tr>
</tbody>
</table>

Table 3: Example decision trees of varying sizes with $n$ input features, $m$ decision nodes, $m'$ padded decision nodes, and $d$ depth for real-world datasets, trained using scikit-learn \cite{slid17} which contains a library with these datasets.

actual messages and choice bits. To retrieve the output of the 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 $\tau_{\text{sym}}$ bits of communication per random OT \cite{ALSZ13}.

**Yao’s garbled circuit protocol.** Secure two-party computation allows two parties to compute an arbitrary functionality on their private inputs without learning anything about the other party’s input (beyond the output). Yao’s garbled circuit (GC) protocol \cite{Yao82, Yao86} is one of the most widely used solutions for secure two-party computation. Yao’s protocol works as follows: 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 sends this garbled circuit GC to the other party, the evaluator, who evaluates it after having obliviously received the keys corresponding to its input bits from the garbler 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 \cite{BMR90}, free-XOR \cite{KS08a}, fixed-key AES garbling \cite{BHKR13}, and half-gates \cite{ZRE15}, while at the implementation level Single Instructions Multiple Data (SIMD) operations exist including point-and-permute \cite{BMR90}, free-XOR \cite{KS08a}, fixed-key AES garbling \cite{BHKR13}, and symmetric-key operations linear in the size of the Boolean circuit describing the function. Various optimizations are implemented in the ABY framework \cite{DSZ15}, which we use in our implementation. With all 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 \cite{BMR90}. In the classic garbling scheme, each wire $i$ has two labels ($k_0^i$ for 0 and $k_1^i$ 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: $(GC, out^0, out^1) \leftarrow \text{Garble}(C, in^0, in^1)$ denotes the 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. $(GC, \text{color}) \leftarrow \text{Garble'}(C, in^0, in^1)$ denotes the same garbling method but returns the color bits along with GC. out $\leftarrow \text{Eval}(GC, in)$ denotes GC evaluation where the evaluator inputs his input wire keys in (received via OTs from the garbler), and receives the output wire keys out corresponding to the output bits. The actual output remains oblivious to the evaluator, since he does not learn the values the wire keys correspond to. out$^+$ $\leftarrow \text{Eval'}(GC, in)$ denotes a similar method, where the evaluator receives blinded output bits out$^+$, i.e., the color bits of the outputs.

**Homomorphic encryption.** Additively homomorphic encryption allows anyone holding the public key $pk$ to calculate $[x_1 + x_2] = [x_1] \oplus [x_2]$, given two ciphertexts $[x_1] := \text{Enc}_{pk}(x_1)$ and $[x_2] := \text{Enc}_{pk}(x_2)$. Multiplying $[c \cdot x] = c \cdot [x]$ 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.
Ideal functionality $\mathcal{F}_{\text{PDTE}}$.

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

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

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 $n$ ciphertexts $[x_1], \ldots, [x_n]$ into a single one: $[X] = [x_n||x_{n-1}||\ldots||x_1]$, 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 encrypted plaintext values are in a known small subset of the plaintext space.

3 Protocol Building Blocks

Private decision tree evaluation is run between two parties: a server $S$ holding a classification tree $T$ (with threshold vector $y$, input selection $\sigma$ and classification values $v$) and a client $C$ providing its input feature vector $x$. After the protocol evaluation, $C$ obtains the classification result, without learning any information about the tree beyond its size and the output, and $S$ learns nothing about $C$’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 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 one 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 $m$ or the number of decision nodes $m'$ after depth-padding (cf. §2.1). This is according to the approach chosen for path evaluation (§3.3). We use $m$ decision nodes in our description, where possibly $m = m$ or $m = m'$.

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 \( F_{\text{Sel}} \), given a secret sharing scheme \( S_1 \).

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

Fig. 3: \( F_{\text{Sel}} \) – Ideal functionality for obliviously selecting \( m \) elements from \( n \leq m \) elements.

Ideal functionality \( F_{\text{Comp}} \), given two secret sharing schemes \( S_1, S_2 \).

1. **Private inputs of \( S, C \):** secret \( S_1 \) shares of the selection \( x_{\sigma(1)}, \ldots, x_{\sigma(m)} \).
2. **Private input of \( S \):** \( y = \{y_1, \ldots, y_m\} \).
3. **Compute:** for \( i \in \{1, \ldots, m\} \): \( c_i = 1 \) if \( x_{\sigma(i)} < y_i \), else \( c_i = 0 \).
4. **Output of \( S, C \):** secret \( S_2 \) shares of \( c_1, \ldots, c_m \).

Fig. 4: \( F_{\text{Comp}} \) – Ideal functionality for obliviously comparing \( m \) pairs of elements.

Ideal functionality \( F_{\text{Path}} \), given secret sharing scheme \( S_2 \).

1. **Private inputs of \( S, C \):** secret \( S_2 \) shares of the comparison results \( \{c_1, \ldots, c_m\} \).
2. **Private input of \( S \):** decision tree \( T \).
3. **Compute:** evaluation of the decision tree path using the comparison results resulting in leaf \( v \).
4. **Output of \( S \):** classification result \( v \).

Fig. 5: \( F_{\text{Path}} \) – Ideal functionality for oblivious path evaluation.

### 3.1 Selection Phase

The selection phase obliviously selects from the \( n \) features of the client \( C \) a value for each of the \( m \) decision nodes (where they are to be compared to thresholds) according to the selection \( \sigma : \mathbb{m} \to n \) of the server \( S \). The ideal functionality \( F_{\text{Sel}} \) that describes this selection is shown in Fig. 3. We identify two ways to instantiate it, one based on additively homomorphic encryption (Sel\(_H\)), and another one based on the secure evaluation of a Boolean selection network using garbled circuits (Sel\(_C\)). In both protocols, the outputs of the parties are secret shares of the result: none of the parties learns any information about the output, but they are able to continue secure computation with it.

**Selection using additively homomorphic encryption (Sel\(_H\), Fig. 6).** Additively homomorphic encryption has been used to perform oblivious selection in [BPSW07, BFK+09]. In the protocol, the client \( C \) encrypts its inputs, and sends the ciphertexts to the server \( S \). \( S \) then selects the values according to selection \( \sigma \) 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). \( S \) then sends these to \( C \), who decrypts them, retrieving a blinded selection of its inputs. In this sub-protocol, we use either Paillier encryption with ciphertext packing for the \( m \) ciphertexts to reduce the communication to \((n + m)/(t + s)) \cdot \tau_{\text{Paillier}} \) bits (cf. §2.2), or DGK encryption with smaller ciphertexts and \((n + m) \cdot \tau_{\text{DGK}} \) bits of communication.

**Selection using a garbled selection network (Sel\(_C\), Fig. 7).** An alternative for obliviously selecting the features is evaluating a selection network [KS08b, BFK+09], which is based on Waksman’s permutation network [Wak08]. Firstly, \( S \) sends the GC corresponding to the selection network to \( C \). They then perform \( nt \)
### 3.2 Comparison Phase

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

**Comparison (including selection) using additively homomorphic encryption (Sel\(_H\) + 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 thresholds, the selection phase Sel\(_H\) can be realized within this protocol without additional overhead. Thus, it

---

**Diagram 1:** Selection using additively homomorphic encryption.

<table>
<thead>
<tr>
<th>Server ( S )</th>
<th>Client ( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( m ), selection ( \sigma ), public key ( pk )</td>
<td><strong>Input:</strong> ( x = (x_1, \ldots, x_n) ), key pair ( pk, sk )</td>
</tr>
<tr>
<td><strong>Output:</strong> ( r_1, \ldots, r_m ) random</td>
<td><strong>Output:</strong> ( z_1, \ldots, z_m ) blinded values</td>
</tr>
</tbody>
</table>

**Offline**

Generate \( r_1, \ldots, r_m \) random \((t + s)\)-bit values

**Online**

For \( i \in \{1, \ldots, m\} \)
- \( [x_1], \ldots, [x_n] \leftarrow [x_{\sigma(i)}] + r_i \)
- \( [z_1], \ldots, [z_m] \rightarrow [z_{\sigma(i)}] \)

**Return** \( r_1, \ldots, r_m \)

**Diagram 2:** Selection using a garbled selection network.

<table>
<thead>
<tr>
<th>Server ( S )</th>
<th>Client ( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( m ), selection ( \sigma )</td>
<td><strong>Input:</strong> ( x = (x_1, \ldots, x_n) )</td>
</tr>
<tr>
<td><strong>Output:</strong> ( \mathbf{K}^0 = (K_{1,1}^0, \ldots, K_{m,t}^0) ), ( \mathbf{K}^1 = (K_{1,1}^1, \ldots, K_{m,t}^1) )</td>
<td><strong>Output:</strong> ( \mathbf{K} = (K_1, \ldots, K_{m,t}) ) keys</td>
</tr>
</tbody>
</table>

**Offline**

Generate \( \mathbf{C} \) of selection network realizing \( \sigma : m \rightarrow n \)
- \( (GC, (K_{1,1}^0, \ldots, K_{m,t}^0), (K_{1,1}^1, \ldots, K_{m,t}^1)) \leftarrow \text{Garble}(\mathbf{C}, (s_{1,1}^0, \ldots, s_{n,t}^0), (s_{1,1}^1, \ldots, s_{n,t}^1)) \)

**Online**

For \( i \in \{1, \ldots, n\} \)
- For \( j \in \{1, \ldots, t\} \)
  - \( s_{i,j}^0, s_{i,j}^1 \leftarrow \text{OT} \)
  - \( x_{i,j} \leftarrow [s_{i,j}^0 + s_{i,j}^1] \)
  - \( s_{i,j}^1 \leftarrow [x_{i,j}] \)

**Return** \( K^0 = (K_{1,1}^0, \ldots, K_{m,t}^0) \), \( K^1 = (K_{1,1}^1, \ldots, K_{m,t}^1) \)

**Return** \( (K_{1,1}, \ldots, K_{m,t}) \leftarrow \text{Eval}(GC, (s_{1,1}^1, \ldots, s_{n,t}^1)) \)

OTs for each bit of the client’s input. In most cases, \( n \leq m \), since features are used more than once. The size of a selection network that maps \( n \leq m \) \( t \)-bit elements to \( m \) elements is \( t \cdot \frac{m!}{(m-t)!} = t \cdot (0.5(n+m) \log(n) + m \log(m) - n + 1) \) [KS16, Appendix C]. The communication of this protocol is \((n + 2S_{m \geq n})t \cdot \tau_{\text{sym}} \) bits offline and \( nt(2\tau_{\text{sym}} + 1) \) bits online.
The communication in this variant is formed with generic secure two-party computation protocols. This method has been used for comparison decision tree evaluation. We compare with this protocol in §d protocol that uses their protocol such that only more values are compared. This incurs an overhead when the client encrypts its elements, i.e., it now needs the DGK comparison protocol presented in Fig. 8, it does not allow for including the selection step when comparing two values with each other. However, as opposed to encryption scheme can be used here (cf. random bit x<y is true if and only if Hックス₀₁,...,χₙ=₀ then b₁=₁ else b₂=₀

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Fig8.png}
\caption{Sel₉+Comp₉ – Oblivious selection and comparison using additively homomorphic encryption from [WFNL16, TMZC17].}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Fig9.png}
\caption{Comp₉ – Oblivious comparison using garbled circuits.}
\end{figure}

already includes the oblivious selection protocol of §3.1 for free (cf. Tab. 1). It is observed in [DGK07] that x<y is true if and only if \(\exists i \in \{1, \ldots, t\}\) such that \(f_i(x, y) = x_i - y_i + 1 + 3 \sum_{j<i} (x_j \oplus y_j) = 0\). \(C\) encrypts each bit of its input with an additively homomorphic encryption scheme (cf. §2.2), and sends the ciphertexts to \(S\), who chooses the appropriate feature to compare with at each decision node and generates a random bit \(a_k\) for each comparison \(k \in \{1, \ldots, \overline{m}\}\). Then, the above comparison is performed 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 \(S\) holds random bit \(a_k\), \(C\) 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 \((n + \overline{m})t \cdot \tau_{\text{ElGamal}}\) bits.

Joye and Salehi [JS18] present an optimization of the DGK comparison protocol that improves the communication roughly by factor two when comparing two values with each other. However, as opposed to the DGK comparison protocol presented in Fig. 8, it does not allow for including the selection step when more values are compared. 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 propose 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 protocol in §4.2.

**Comparison using garbled circuits (Comp₉, Fig. 9).** All oblivious comparisons can also be performed with generic secure two-party computation protocols. This method has been used for comparison
Input: \(a = (a_1, \ldots, a_m)\), public key \(pk\)  
Output: \(\perp\)

For \(j \in \{1, \ldots, m+1\}\): \(r_j \leftarrow R \mathbb{Z}_p^*, r'_j \leftarrow R \mathbb{Z}_p^*\)

Input: \(b = (b_1, \ldots, b_m)\), key pair \(pk, sk\)
Output: classification \(v\)

For \(i \in \{1, \ldots, m\}\)  
\[ [B_i] \leftarrow [b_i \oplus a_i], [ec_{ci,0}] \leftarrow [B_i], [ec_{ci,1}] \leftarrow [1 - B_i] \]

For \(j \in \{1, \ldots, m+1\}\)  
\[ [pc_j] \leftarrow [\sum_{e_{j,k} \in P} ec_{j,k}], [pc'_j] \leftarrow [r_j \cdot pc_j], [\tau_j] \leftarrow [r'_j \cdot pc_j + v_j] \]

Permuted \([pc'_j], [\tau_j]\)  
For \(j \in \{1, \ldots, m+1\}\)  
If \(\text{Dec}_{sk}([\tau_j]) = 0\) then  
Return \(v \leftarrow \text{Dec}_{sk}([v_j])\)

Fig. 10: Path\(_H\) – Oblivious path evaluation using additively homomorphic encryption from [TMZC17].

Input: \(k^0 = (k^0_1, \ldots, k^0_m'), k^1 = (k^1_1, \ldots, k^1_m)\)  
Output: \(\perp\)

\(DT \leftarrow \text{DepthPad}(DT)\)
\(GDT \leftarrow \text{Garble}(DT, k^0, k^1)\)

\(GDT\)

Online  
Return \(v \leftarrow \text{Eval}(GDT, k)\)

Fig. 11: Path\(_G\) – Oblivious path evaluation using a garbled decision tree.

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 \(2\overline{m}t \cdot \tau_{\text{sym}}\) bits communication. Variant (a) of this comparison protocol outputs the Boolean shares of the output of the comparisons, i.e., 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 keys for the output wires (without knowing the values they correspond to).

### 3.3 Path Evaluation Phase

This section describes two path evaluation protocols, the first using homomorphic encryption (Path\(_H\)) and the second using a garbled decision tree (Path\(_G\)). The ideal functionality \(F_{\text{Path}}\) is given in Fig. 5.

**Path evaluation using additively homomorphic encryption (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 \(C\) encrypts his share of the comparison bit for each decision node, and sends it to the server. The server \(S\) homomorphically computes the encryption of the actual comparison bit (by removing the blinding), 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 requires \(O(m)\) operations since intermediate values can be reused. These and the classification labels on each leaf node are then blinded.
This means that at node
i
where
the respective figures. In sub-protocols based on homomorphic encryption, we can precompute encryp-
tions offline-online setting. We note that it is possible to precompute parts of the protocols even if not shown in
d and potentially the depth
m
server’s decision tree: either the number of decision nodes
m
or the number of padded decision nodes
m
′
in the protocols.

We adapt all resulting protocols with security against passive adversaries to an
sym
-interface comparison. The online rounds and leakage about the model. In
3
that result in private decision
trees.

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
server’s decision tree: either the number of decision nodes
m
or the number of padded decision nodes
m
′
and potentially the depth
d.

4 Protocols for Private Decision Tree Evaluation

by
S
using fresh randomness, and sent to the client
C.
If the path cost decrypts to zero,
C
can decrypt the
classification label. This path evaluation protocol has
(3m + 2) · \tau_{ElGamal}
bits of communication.

Path evaluation using a garbled decision tree (PathG, Fig. 11). The method for garbled deci-
sion tree (GDT) evaluation is analogous to Yao’s garbled circuit technique [Yao86] described in §2.2, but
Evaluating (GDT, in) returns the output classification in the clear. Our description is similar to that of [BPSW07]
that has been improved in [Sch08, 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
C
cannot deviate from the path corresponding to the comparison results with his input vector
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. Alternatively, one can use full decision trees [BPSW07], but then the overhead
is exponential in the depth
m.
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 child nodes are stored
in an encrypted manner using the key of the node and keys
k
l
i
or
k
r
i
depending on the comparison result.
This means that at node
i,
Enc
k
i
(k
l
i
|l) and
Enc
k
i
(k
r
i
|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
l
i
and
k
r
i
. The server sends
GDT
to the client, who can evaluate it by decrypting one path
leading to the leaf node storing the classification. The communication in this protocol is sending the
GDT,
each node of which stores two \tau_{sym}-bit values and two indices, i.e.,
2m(m + \log_2(m + 1)). An extension of
this protocol presented in §4.2 reduces the client’s computation to \Theta(dt) symmetric key operations.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>(m)</th>
<th>Selection (Sel.)</th>
<th>Interface</th>
<th>Comparison (Comp.)</th>
<th>Interface</th>
<th>Path evaluation (Eval.)</th>
<th>Online rounds</th>
<th>Leakage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BPSW07]</td>
<td>m</td>
<td>H (Fig. 6)</td>
<td>G</td>
<td>G (Fig. 9)</td>
<td>G (Fig. 11)</td>
<td>4</td>
<td>(m, d_{path})</td>
<td></td>
</tr>
<tr>
<td>[WFNL16]</td>
<td>m</td>
<td>H (Fig. 8)</td>
<td>G</td>
<td>G (Fig. 9)</td>
<td>G (Fig. 11)</td>
<td>6</td>
<td>(m)</td>
<td></td>
</tr>
<tr>
<td>HHG</td>
<td>m</td>
<td>H (Fig. 8)</td>
<td>G (Fig. 7)</td>
<td>G (Fig. 9)</td>
<td>G (Fig. 10)</td>
<td>4</td>
<td>(m)</td>
<td></td>
</tr>
<tr>
<td>GGG</td>
<td>m</td>
<td>H (Fig. 7)</td>
<td>G (Fig. 13)</td>
<td>G (Fig. 11)</td>
<td>G (Fig. 10)</td>
<td>4</td>
<td>(m)</td>
<td></td>
</tr>
<tr>
<td>HGH</td>
<td>m</td>
<td>G (Fig. 7)</td>
<td>G (Fig. 7)</td>
<td>G (Fig. 9)</td>
<td>H (Fig. 10)</td>
<td>6</td>
<td>(m)</td>
<td></td>
</tr>
<tr>
<td>GGH</td>
<td>m</td>
<td>G (Fig. 13)</td>
<td>G (Fig. 9)</td>
<td>G (Fig. 10)</td>
<td>H (Fig. 10)</td>
<td>4</td>
<td>(m)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Concrete protocols based on the presented sub-protocols for selection, comparison and path evaluation,
where \(m\) denotes either the number of decision nodes
m
or padded decision nodes
m
′
in the protocols.

Round complexity and leakage of different private decision tree evaluation protocols based on homomorphic
encryption
H
and garbling techniques
G.
independent from the client’s input, while in sub-protocols based on garbling techniques, we can perform the base OTs and precompute OTs (cf. §2.2).

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 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.

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) [BFK+09] and comparison Comp_G (Fig. 9) [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 than the method based only on homomorphic encryption [TMZC17] with respect to total runtime in most cases as well.

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 used, m = m in all preceding sub-protocols. This has been a tremendous improvement for real-world decision trees 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 order to hide the tree topology. Therefore, the tree is depth-padded to m = m’ decision nodes. m’ depends on the topology of the decision tree, i.e., the level of all leaves (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]).
We show that protocols based on garbling techniques are especially efficient in this setting, whereas the order to perform the comparison on (Fig. 6 is a set of blinded unblind the output. After the comparison, the path evaluation Path \( H \) as the input of \( \text{Comp} G \) in order to achieve linear complexity. When combining these approaches, the result of \( \text{Sel} \) from the client's \( m \) can be precomputed in an offline phase. Thereafter, the client evaluates these in the online phase (using encryption for all three sub-protocols was presented in [TMZC17], the comparison protocol of which was since the computation depends on the client's input. However, this comparison protocol requires a selection sub-protocol to choose \( m \), so the interface described below and depicted in Fig. 13 is inserted to obliviously compare and Path \( H \) (Fig. 10). The advantage of this protocol is that very low communication is necessary between the two parties and the selection happens along with the comparison. However, most computationally expensive cryptographic operations must be computed online, since the computation depends on the client’s input.

**HHH: \( \text{Sel}_H + \text{Comp}_H + \text{Path}_H \)** [TMZC17]. The state-of-the-art protocol using only homomorphic encryption for all 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 \( \text{Sel}_H + \text{Comp}_H \) (Fig. 8) with \( \text{Path}_H \) (Fig. 10). The advantage of this protocol is that very low communication is necessary between the two parties and the selection happens along with the comparison. However, most computationally expensive cryptographic operations must be computed online, since the computation depends on the client’s input.

**HGH: \( \text{Sel}_H + \text{Comp}_G + \text{Path}_H \).** An oblivious comparison protocol that can be used to output shares of the comparison bits is Yao’s GC protocol is shown in \( \text{Comp}_G \) (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 from the client’s \( n \leq m \) inputs. In this protocol, we use \( \text{Sel}_H \) with Paillier or DGK encryption (cf. Fig. 6) in order to achieve linear complexity. When combining these approaches, the result of \( \text{Sel}_H \) is not the same as the input of \( \text{Comp}_G \), so the interface described below and depicted in Fig. 13 is inserted to obliviously unblind the output. After the comparison, the path evaluation \( \text{Path}_H \) of [TMZC17] can be used as before (cf. Fig. 10).

**Combining homomorphic selection and garbled comparison** (\( \text{Sel}_H \rightarrow \text{Comp}_G \), Fig. 13). The result of \( \text{Sel}_H \) in Fig. 6 is a set of blinded \((t + s)\)-bit values for the client \( C \), and the blinding \( s \)-bit values for the server \( S \). In order to perform the comparison on \( t \)-bit values, \( S \) and \( C \) must unblind the result using a GC for subtracting

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server ( S )</strong></td>
<td>Input: ( r_1, \ldots, r_m ) &lt;br&gt;Output: ( K^0 = (K_{1,1}^0, \ldots, K_{m,t}^0) ), ( K^1 = (K_{1,1}^1, \ldots, K_{m,t}^1) )</td>
</tr>
<tr>
<td><strong>Client ( C )</strong></td>
<td>Input: ( z_{1,1}, \ldots, z_{m,t} ) blinded values (truncated) &lt;br&gt;Output: ( K = (K_{1,1}, \ldots, K_{m,t}) ) keys</td>
</tr>
<tr>
<td><strong>Offline</strong></td>
<td>Generate circuit ( C ) for ( t )-bit subtractions of ( r_1, \ldots, r_m ) &lt;br&gt;( (GC, (K_{1,1}^0, \ldots, K_{m,t}^0), (K_{1,1}^1, \ldots, K_{m,t}^1)) ) &lt;br&gt;( \text{Garble}(C, (s^0_{1,1}, \ldots, s^0_{m,t}), (s^1_{1,1}, \ldots, s^1_{m,t})) )</td>
</tr>
<tr>
<td><strong>Online</strong></td>
<td>For ( i \in {1, \ldots, m} ) &lt;br&gt;For ( j \in {1, \ldots, t} ) &lt;br&gt;( s^0_{i,j}, s^1_{i,j} ) &lt;br&gt;( \text{OT} ) &lt;br&gt;( z_{i,j} ) &lt;br&gt;( s_{i,j} )</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>( K^0 = (K_{1,1}^0, \ldots, K_{m,t}^0) ) &lt;br&gt;( K^1 = (K_{1,1}^1, \ldots, K_{m,t}^1) ) &lt;br&gt;( K = (K_{1,1}, \ldots, K_{m,t}) = \text{Eval}(GC, s) )</td>
</tr>
</tbody>
</table>
each random value obliviously \cite{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 Comp\(_G\)
(cf. Fig. 9). Overall, the \( m \) garbled subtraction circuits require \( 2m \cdot \tau_{\text{sym}} \) bits offline communication, \( m \cdot \tau_{\text{sym}} \) bits for precomputing random OTs, and \( m t (2 \tau_{\text{sym}} + 1) \) 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 \( O(m t \log m) \) 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 higher than in the previous protocols due to the selection network with superlinear
complexity, 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\(_G\) \cite{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\(_G\) \cite{BFK+09}.** This protocol consists exclusively of sub-protocols that
use garbling techniques: selection Sel\(_G\) (Fig. 7) and comparison Comp\(_G\) (Fig. 9) with garbled circuits, and
garbled path evaluation Path\(_G\) (Fig. 11). It has the highest communication, though most data can be sent in
the offline phase.

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., \cite{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 \cite{TMZC17}.

**HHG: Sel\(_H\) + Comp\(_H\) + Path\(_G\).** Due to the worse 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\(_H\)→Path\(_G\); Fig. 14).* The result
of Sel\(_H\) + Comp\(_H\) (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 \( C \) and two keys for both
possible bits for the server \( S \). After \( S \) generates these keys, \( C \) and \( S \) engage in a 1-out-of-2 OT protocol
for each comparison bit (cf. §2.2). This conversion requires \( m' \tau_{\text{sym}} \) bit offline and \( m' (\tau_{\text{sym}} + 1) \) bit online
communication, since we can use the correlated OT (C-OT) extension optimization of \cite{ALSZ13}.
4.2 Protocol Extensions

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

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, and a 1-out-of-2 \(i\) OT is performed at the \(i\)th 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 (\(O(n + d(t + m'))\)) is better than that of HGG (\(O(n + m't)\)) for small decision trees, its dependency on the depth makes it worse for larger examples such as the \textit{boston} dataset (cf. §2.1). Moreover, it has \(O(d)\) 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 categorical variables, and in §B.2 for securely evaluating random forests. Classification confidence values (correctness probabilities), can be appended to the classification labels in a straightforward manner.

5 Performance Evaluation

In this section, we compare the runtime and communication of all resulting protocols from §4. In our implementation, we instantiate primitives corresponding to security level \(\kappa = 128\) (and \(\kappa = 112\) for public-key primitives) as recommended by NIST for use until 2030 [BBB+16], and statistical security parameter \(s = 40\), and as in prior work, we set the bitlength to \(t = 64\). \(\tau\) denotes the ciphertext sizes: \(\tau_{\text{sym}} = 128\) for symmetric-key encryption, \(\tau_{\text{Paillier}} = 4096\) for Paillier encryption, \(\tau_{\text{DGK}} = 2048\) for DGK encryption, and \(\tau_{\text{ElGamal}} = 514\) for ElGamal encryption with elliptic curves and point compression.

The underlying frameworks for our open-source implementation that can be found at https://encryptode/code/PDTE are the ABY secure two-party computation framework [DSZ15]\(^1\) and the mcl library\(^2\) that includes a highly optimized lifted ElGamal implementation. 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\(_G\) [Sch08]. We precompute OTs and homomorphic encryption when possible (i.e., Figs. 6, 7, 8, 10, 14). We give further details on our implementation in §C. We show the performance of the protocols for the datasets from Tab. 3 as well as for a full tree \texttt{full(13)} with \(d = n = 13\) for a comparison for dense trees.

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 require more communication in total. Homomorphic encryption-based methods have less communication, however, almost all expensive computation and all communication happen in the online phase. Methods using Path\(_H\) (i.e., GGH, HGH and HHH [TMZC17]) have a clear advantage for large sparse trees, which is lost in case extremely dense trees are considered (such as our example full tree \texttt{full(13)}). For these kind of trees, Wu et al.’s protocol [WFNL16] has lower communication as discussed in §7.

---

\(^1\) https://github.com/encryptogroup/ABY
\(^2\) https://github.com/herumi/mcl
Fig. 15: Communication of protocols using the example datasets from Tab. 3. Note that the y-axis is in logarithmic scale.

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

**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, we neglect the costs for the base OTs and for generating the keypair for additively homomorphic encryption, since they are a one-time expense that can be re-used over multiple protocol executions. We also neglect the cost for trial decryption of the classification value in Path_H, and note that for a 16-bit value it is around 40 ms on average.

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 (SelH and CompH) perform an order of magnitude worse than their garbling technique-based alternatives (SelG and CompG). 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 \( m \leq m' \leq \frac{1}{2}m(m + 1) \), which can be significantly larger than \( m \) (cf. Fig. 12 and Tab. 3). Our protocol GGH has the fastest total and online runtime of below 1 second for our largest real-world example boston. For extremely dense decision trees such as our example full tree full(13), this advantage is lost (since no padding is necessary) and the runtime of
protocols based on garbling techniques is an order of magnitude lower than that of the protocols based on homomorphic encryption in the LAN setting.

**Tradeoff.** In order to show the communication ($x$-axis) vs. computation ($y$-axis) tradeoff of all resulting protocols, we give figures for decision trees trained on two example real-world datasets: a small dataset *wine*, and our largest dataset *boston*.

While the number of input features $n$ is relatively small for all datasets, the number of (potentially padded) decision nodes significantly affects the efficiency of our sub-protocols. 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) when Path$_G$ is used for path evaluation. Moreover, the selection network in Sel$_G$ has superlinear size $O(m \log m)$, which implies a significant overhead in offline communication. This gap can be observed on the figures: the protocols using garbling techniques for path evaluation lose their performance advantage with growing number of dummy nodes, and the protocols using garbling techniques for selection 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, our largest decision tree trained on the *boston* dataset requires around 1.8 MBytes of total 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 tradeoffs 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.
Fig. 18: Tradeoff between total communication (x-axis) and runtime (y-axis). Note that both axes are in logarithmic scale, with an additional 0 on the x axis for depicting the offline communication of the HHH protocol. Pareto points are marked in bold.

In Figs. 17 and 18, we mark in bold pareto points for online and total tradeoffs, respectively, i.e., we bold the protocols where there is no possibility 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 2-5x with only 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 large datasets by an order of magnitude in all complexities 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 and communication by up to a factor of 2, but is worse in both total complexities. HGH improves even more (by 5-73x) for our datasets over the communication of GGG due to the difference in the number of decision nodes. The online runtimes of HGH are larger by 2-18x for real-world datasets, and the total runtimes are also larger by a factor of 5 for small datasets. However, it gets better with larger (sparse) datasets due to the increasing number of decision nodes in the depth-padded tree.

**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 from §4.1 provide the best online performance, depending on the dataset size. The former may have 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 HHH of [TMZC17] has the lowest total communication, but uses computationally heavy public-key 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 with garbling techniques yield an order of magnitude faster runtimes while 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 Salehi [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. These, in practice, are usually not very dense, and therefore, we use this protocol to instantiate 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.

The protocols of [WFNL16] and [TMZC17] use the DGK comparison protocol [DGK07]. Joye and Salehi present an optimization on the DGK protocol and a private decision tree evaluation protocol with \( O(d) \) rounds in [JS18] (cf. §3.2). For private decision tree evaluation with constant rounds, their optimization on the DGK protocol is not applicable. The alternative solution to the DGK protocol of [LZS18] uses fully homomorphic encryption and performs well for small bitlengths \( t \) of the features and inputs.

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] represent the decision tree as an array, and implement 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 \( O(d) \) rounds of communication [TKK19, Tab. 2].

Alternatively, private decision tree evaluation can 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., \( O(t m \log m) \) gates for the selection of inputs as in Sel\(_Q\) in §3.1). Secure evaluation of universal circuits (UCs) is an equivalent solution for PFE [Val76, KS08b, KS16, GKS17]. UC- and OT-based PFE protocols [KS08b, MS13, KS16, GKS17] need \( O((tm \log m) \log(tm \log m)) \), and HE-based protocols [KM11, MS13, MSS14] need \( O(t m \log m) \) computation and communication. These complexities are larger than that of protocols designed specifically for DTs.

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

We thank the anonymous reviewers for their valuable feedback on the paper. This work was supported by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, by the DFG as part of project E4 within the CRC 1119 CROSSING, by the Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS), and by Business Finland (CloSet project, 3881/31/2016).
References


Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A. Scott Patterson. Privacy-preserving decision trees over vertically partitioned data. ACM Transactions on Knowledge Discovery from Data (TKDD), 2(3):14:1–14:27, 2008.


A Complexities of Sub-protocols

In this section, we describe the complexities of all sub-protocols described in §3. Note that oblivious transfers and homomorphic encryptions are precomputed in the offline phase when possible.

**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 CompG and PathG 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 \( x_j \in U' = \{U' \subset U \ (j \in \{1, \ldots, n\}) \}. \) Wu et al. supply the SelH + CompH subprotocol with the extension to handle categorical variables in [WFNL16]. We briefly describe a private approach to evaluate the

<table>
<thead>
<tr>
<th>Sub-protocol</th>
<th>Fig.</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelH Paillier</td>
<td>Fig. 6</td>
<td>-</td>
<td>((n + \frac{m}{t_{\text{sym}}}) \cdot t_{\text{Paillier}})</td>
</tr>
<tr>
<td>SelH DGK</td>
<td>Fig. 6</td>
<td>-</td>
<td>((n + \frac{m}{t_{\text{DGK}}}) \cdot t_{\text{DGK}})</td>
</tr>
<tr>
<td>SelG</td>
<td>Fig. 7</td>
<td>((n + 2S_{\text{sym}}^m) t_{\text{sym}})</td>
<td>(n(2t_{\text{sym}} + 1))</td>
</tr>
<tr>
<td>SelH + CompH</td>
<td>Fig. 8</td>
<td>-</td>
<td>((n + \frac{m}{t_{\text{sym}}}) \cdot t_{\text{ElGamal}})</td>
</tr>
<tr>
<td>CompG</td>
<td>Fig. 9</td>
<td>-</td>
<td>((3m + 2) \cdot t_{\text{ElGamal}})</td>
</tr>
<tr>
<td>PathH</td>
<td>Fig. 10</td>
<td>-</td>
<td>((3m + 2) \cdot t_{\text{ElGamal}})</td>
</tr>
<tr>
<td>PathG</td>
<td>Fig. 11</td>
<td>(2m' \cdot (t_{\text{sym}} + \log_2(m' + 1)))</td>
<td>(m'(2t_{\text{sym}} + 1))</td>
</tr>
<tr>
<td>SelH→CompG</td>
<td>Fig. 13</td>
<td>(3m' \cdot t_{\text{sym}})</td>
<td>(m'(t_{\text{sym}} + 1))</td>
</tr>
<tr>
<td>CompH→PathG</td>
<td>Fig. 14</td>
<td>(m' \cdot t_{\text{sym}})</td>
<td>(m'(t_{\text{sym}} + 1))</td>
</tr>
</tbody>
</table>

Table 5: Offline and online communication of all sub-protocols, where \( n, t, m, m' \) and \( m' \) denote the number input features, the bitlength of the features and thresholds, the number of decision nodes, padded decision nodes and either of the former two, respectively. \( S_{m' \geq n} \) denotes the size of the selection network that selects \( m' \) bits from \( n \) bits. \( t_{\text{sym}}, t_{\text{DGK}}, t_{\text{Paillier}} \) and \( t_{\text{ElGamal}} \) denote the size of the ciphertexts in the respective encryption schemes. Note that \( m \leq m' \leq \frac{m}{2}(m + 1)\).
We describe our implementation choices used in §5.

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

\textbf{SelG} (Fig. 7). We implement the garbled selection network introduced in [KS08a] in ABY.

\textbf{SelH + CompH} (Fig. 8). We use the \texttt{mcl} library to implement protocols using lifted ElGamal encryption over elliptic curve (EC) \texttt{secp256k1} with 256-bit key 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, $\tau_{\text{ElGamal}} = 514$. Our baseline implementation is that of [LZS18].

---

<table>
<thead>
<tr>
<th>Sub-protocol</th>
<th>Fig.</th>
<th>Offline Server</th>
<th>Offline Client</th>
<th>Online Server</th>
<th>Online Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelH Paillier</td>
<td>Fig. 6</td>
<td>$O(m)\nu_{\text{Paillier}}$</td>
<td>-</td>
<td>$O((m + n)\nu_{\text{Paillier}}$</td>
<td>-</td>
</tr>
<tr>
<td>SelH DGK</td>
<td>Fig. 6</td>
<td>$O(\tau_{\text{DGK}})$</td>
<td>-</td>
<td>$O((m + n)\nu_{\text{DGK}})$</td>
<td>-</td>
</tr>
<tr>
<td>SelG</td>
<td>Fig. 7</td>
<td>$O((m \log m + n)\nu_{\text{sym}})$</td>
<td>$O(m)\nu_{\text{sym}}$</td>
<td>-</td>
<td>$O((m + n)\nu_{\text{sym}})$</td>
</tr>
<tr>
<td>SelG + CompH</td>
<td>Fig. 8</td>
<td>-</td>
<td>$O((m\nu_{\text{ElGamal(off)}})$</td>
<td>$O((m\nu_{\text{ElGamal(on)}})$</td>
<td>-</td>
</tr>
<tr>
<td>CompH</td>
<td>Fig. 9</td>
<td>$O(m)\nu_{\text{sym}}$</td>
<td>-</td>
<td>$O((m\nu_{\text{ElGamal(on)}})$</td>
<td>-</td>
</tr>
<tr>
<td>PathH</td>
<td>Fig. 10</td>
<td>-</td>
<td>$O((m\nu_{\text{ElGamal(on)}})$</td>
<td>$O((m\nu_{\text{ElGamal(on)}})$</td>
<td>-</td>
</tr>
<tr>
<td>PathG</td>
<td>Fig. 11</td>
<td>$O(m^t)\nu_{\text{sym}}$</td>
<td>$O((m)\nu_{\text{sym}})$</td>
<td>-</td>
<td>$O((m)\nu_{\text{sym}})$</td>
</tr>
<tr>
<td>SelH→CompG</td>
<td>Fig. 13</td>
<td>$O((m\nu_{\text{sym}})$</td>
<td>$O((m)\nu_{\text{sym}})$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CompH→PathG</td>
<td>Fig. 14</td>
<td>$O(m)\nu_{\text{sym}}$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6: The number of cryptographic operations in the offline and online phases of all sub-protocols, with notations as in Tab. 5. $\nu_{\text{sym}}$, $\nu_{\text{DGK}}$, $\nu_{\text{Paillier}}$ and $\nu_{\text{ElGamal}}$ denote the cost of an operation (i.e., encryption, decryption or addition) in symmetric, DGK, Paillier and lifted ElGamal encryption, respectively. $\nu_{\text{ElGamal(off/on)}}$ denote the offline/online costs of precomputed ElGamal encryption.

---

decision nodes that operate over categorical variables when using CompG. We notice that for testing the set inclusion, the server can create the $|U|$-bit masking value Mask based on $U'$ such that $\text{Mask}_i = 1$ 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 $x'_j \land \text{Mask} = x'_j$.

### 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. [WFNL16] provide a method that allows for any affine function of the results using additive secret sharing, which can be generalized to PathH (cf. Fig. 10) of [TMZC17]. For PathG (cf. Fig. 11) [BFK+09], instead of returning the results to the client, he can obtain keys corresponding to these. These can then be used to securely evaluate the GC corresponding to the aggregation function if the server uses these keys to generate it. The complexity depends on the aggregation function.

### C Implementation Choices

We describe our implementation choices used in §5.

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

\textbf{SelG} (Fig. 7). We implement the garbled selection network introduced in [KS08a] in ABY.

\textbf{SelH + CompH} (Fig. 8). We use the \texttt{mcl} library to implement protocols using lifted ElGamal encryption over elliptic curve (EC) \texttt{secp256k1} with 256-bit key 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, $\tau_{\text{ElGamal}} = 514$. Our baseline implementation is that of [LZS18].
**Comp**\textsubscript{G} (Fig. 9). We use comparison circuits in ABY using SIMD operations to enhance the performance of this sub-protocol.

**Path**\textsubscript{H} (Fig. 10). We implement the path evaluation phase using lifted EC-ElGamal encryption over the same curve and library as for Sel\textsubscript{H} + Comp\textsubscript{H}.

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