# Practical Secure Decision Tree Learning in a Teletreatment Application

Sebastiaan de Hoogh[1], Berry Schoenmakers[2],
Ping Chen[3], and Harm op den Akker[4]

[1] TU Delft `s.j.a.dehoogh@tudelft.nl`
[2] TU Eindhoven `berry@win.tue.nl`
[3] KU Leuven `ping.chen@cs.kuleuven.be`
[4] Roessingh R&D and U Twente `h.opdenakker@rrd.nl`

**Abstract.** In this paper we develop a range of practical cryptographic protocols for secure decision tree learning, a primary problem in privacy preserving data mining. We focus on particular variants of the well-known ID3 algorithm allowing a high level of security and performance at the same time. Our approach is basically to design special-purpose secure multiparty computations, hence privacy will be guaranteed as long as the honest parties form a sufficiently large quorum.
Our main ID3 protocol will ensure that the entire database of transactions remains secret except for the information leaked from the decision tree output by the protocol. We instantiate the underlying ID3 algorithm such that the performance of the protocol is enhanced considerably, while at the same time limiting the information leakage from the decision tree. Concretely, we apply a threshold for the number of transactions below which the decision tree will consist of a single leaf—limiting information leakage. We base the choice of the "best" predicting attribute for the root of a decision tree on the Gini index rather than the well-known information gain based on Shannon entropy, and we develop a particularly efficient protocol for securely finding the attribute of highest Gini index. Moreover, we present advanced secure ID3 protocols, which generate the decision tree as a secret output, and which allow secure lookup of predictions (even hiding the transaction for which the prediction is made). In all cases, the resulting decision trees are of the same quality as commonly obtained for the ID3 algorithm.
We have implemented our protocols in Python using VIFF, where the underlying protocols are based on Shamir secret sharing. Due to a judicious use of secret indexing and masking techniques, we are able to code the protocols in a recursive manner without any loss of efficiency. To demonstrate practical feasibility we apply the secure ID3 protocols to an automated health care system of a real-life rehabilitation organization.

## 1 Introduction

Data mining is an evolving field that attempts to extract sensible information from large databases without the need of *a priori* hypotheses. The goal of the design of these data mining algorithms is to be simple and efficient, while providing sensible outputs (such as reliable predictions). Applications include improving services to the (predicted) needs of customers, and automatization of services as we will show below. In health care, for example, automation is of significant importance, since the cost of health care is increasing due to demographic changes and longer life expectancies.

As a motivational example, we consider the following system from [AJH10] that describes a fully automated system, that assists rehabilitation patients. Rehabilitation patients should maintain a certain activity level for a smooth rehabilitation process. A patient is required to carry a small device that measures his activity. The device connects to a smartphone which provides the patient with feedback helping him to maintain his target activity level. The goal is to provide advice in such a way that the patient will follow it. Using data mining techniques the device is able to learn to which (type of) messages the patient is most compliant. In order to overcome the issue of cold start, data mining is applied to patient data of other patients so that the application can be setup in such a way that it provides on average messages to which new patients are likely to comply. More specifically, a decision tree is extracted from old patient data that predicts patients compliance to certain messages in certain circumstances.

Although decision trees may not reveal individual data records, algorithms constructing decision trees require as inputs individual data records. But this leads to privacy issues since patient data is by its nature confidential. Privacy preserving data mining offers a solution. Its goal is to enable data mining on large databases without having access to (some of) the contents. Much research has been done in the field of privacy preserving data mining since the works of Agrawal & Srikant [AS00] and Lindell & Pinkas [LP00]. The solutions can be classified as follows, each having its own advantages and disadvantages [MGA12]: *Anonymization based*, *Pertubation based*, *Randomized Response based*, *Condensation Approach based*, and *Cryptography based*.

Our cryptography based solution will focus on the generation of decision trees using ID3. The cryptography based solutions provide provable security in the framework of multiparty computation, but comes at the cost of time consuming protocols. There are many solutions in the literature that apply multiparty computation techniques to securely evaluate ID3, such as [LP00,VCKP08,DZ02], [XHLS05,SM08,WXSY06,MD08]. All of them require that the database is partitioned in some special way among the computing parties.

In this paper we provide a cryptographic solution for extracting decision trees using ID3 where the database is not partitioned over the parties. In fact, no party is required to have any knowledge of a single entry of the database. We assume that there are $n \geq 3$ parties that wish to evaluate ID3 on a database while having no access to its individual records. Together, they will learn the desired decision tree and nothing more than what can be learned from the tree. We assume that the servers are semi-honest and no more than $n/2$ servers will collude trying to learn additional information.

In contrast to existing secure solutions we assume that no party has knowledge of any record of the database. Nevertheless, the resulting protocols perform well due to the minimal overhead imposed by our approach. In addition, our protocols are designed such that the implementation in VIFF is similar to a straightforward implementation of the original (unsecured) ID3 algorithm. Finally, we show that our protocols are applicable in practice by providing the running times of the protocols on the database used in the rehabilitation application of [AJH10].

### 1.1 Related Work

Privacy preserving data mining using secure multiparty computation for solving real-life problems is first demonstrated in [BTW12], where a secure data aggregation system was built for jointly collecting and analyzing financial data from a number of Estonian ICT companies. The application was deployed in the beginning of 2011 and is still in continuous use. However, their data analysis are limited to basic data mining operations, such as sorting and filtering.

Many results on secure decision tree learning using multiparty computation, however, can be found in the literature. We will briefly describe some of them below.

The first results on secure generation of decision trees using multiparty computation is from Lindell and Pinkas in 2000. In [LP00] they provide protocols for secure ID3, where the database is horizontally partitioned over two parties. They show how to efficiently compute the entropy based information gain by providing two party protocols for computing $x \log x$. Their protocols are based on garbled circuits [Yao86].

Protocols for securely evaluating ID3 over horizontally partitioned data over more than two parties are given in [XHLS05,SM08]. The former provide multiparty protocols computing the entropy based information gain based using threshold homomorphic encryption and the latter applies similar protocols to compute the information gain using the Gini index instead. In the same fashion [MD08] provides protocols for both vertically and horizontally partitioned data using the Gini index, but with a trusted server to provide the parties with shares instead of using homomorphic encryption.

In [DZ02] protocols for secure ID3 over vertically partitioned data over two parties are described and in [WXSY06] protocols over vertically partitioned data over more than two parties are described. Both solutions assume that all parties have the class attribute and show how to gain efficiency by disclosing additional information on the database. These issues have been addressed by [VCKP08], where a secure set of protocols for vertically partitioned data over more than two parties is discussed without disclosing any additional information and where not all parties have the class attribute.

## 2 The ID3 Algorithm

Decision tree learning is a basic concept in data mining. A popular algorithm is the Iterative Dichotomizer 3 (ID3) from [Qui86] that extracts a *decision tree*

**Algorithm 2.1** ID3($T, R$)

```
1: $i^* = \arg\max_i |T \cap S_{0,i}|$
2: if $R = \emptyset$ or $|T| \leq \epsilon|\mathcal{T}|$ or $|T \cap S_{0,i^*}| = |T|$ then
3:     return $\langle c_{i^*} \rangle$
4: else
5:     $k^* = \arg\max_k f(T, A_k)$
6:     return $\langle A_{k^*}, \{\text{ID3}(T \cap S_{k^*,j}, R \setminus \{A_{k^*}\})\}_j \rangle$
```

from a dataset viewed as a table from a structured database. Each row is called a *transaction* and each column corresponds to an attribute. One of the attributes is the target attribute or *class attribute*, which one wants to predict for new transactions given values for the other attributes. For example, in the teletreatment scenario, the attributes include the gender and age of a patient as well as specific attributes such as the advice given to the patient (e.g., " go for a walk right now") and the weather conditions; the class attribute indicates whether or not the patient is compliant with the advice given.

We will use the following notation. Consider database $\mathcal{T}$ with attributes $\mathcal{A} = \{A_k\}$. Let $C = A_0$ denote the class attribute. For each $A_k \in \mathcal{A}$, let $\{a_{kj}\}$ be the set of possible values for attribute $A_k$ and let $\{c_i\} = \{a_{0i}\}$ be the set of possible values for the class attribute $C$. For any $t \in \mathcal{T}$, we denote by $t(A_k)$ the value of attribute $A_k$ in transaction $t$. Let $S_{k,j} = \{t \in \mathcal{T} : t(A_k) = a_{kj}\}$ denote the set of transactions in $\mathcal{T}$ for which attribute $A_k$ has the value $a_{kj}$. Note that $\{S_{k,j}\}_j$ forms a partition of $\mathcal{T}$, which we will call the *partition of* $\mathcal{T}$ *according to* $A_k$.

The overall approach of ID3 is to recursively choose the attribute that best classifies the transactions and partition the database according to the values of that attribute, see Algorithm 2.1. ID3 takes as input a set of transactions $T \subseteq \mathcal{T}$ together with a set of non-class attributes $R \subseteq \mathcal{A} \setminus \{C\}$ over which the decision tree is built. First the algorithm checks whether some stopping criterion is satisfied. There are many common stopping criteria [RM05], each having its own merits. We use the following three stopping criteria. Firstly, if no further partition is possible, i.e., if $R = \emptyset$. Secondly, if the class attribute takes on only one value, i.e., if $|S_{0,i}| = |T|$ for some $i$. And, finally, if the number of transactions in a partition is relatively small, i.e., if $|T|/|\mathcal{T}| \leq \epsilon$, for some small $\epsilon$. In all cases when a stopping criterion is satisfied, ID3 returns a leaf node containing the value for $C$ that occurs most frequently in the transactions in $T$.

If none of the stopping criteria is satisfied, ID3 continues by choosing some attribute $A_{k^*} \in R$ and returning a tree with root $A_{k^*}$ and a subtree generated recursively as ID3$(T \cap S_{k^*,j}, R \setminus \{A_{k^*}\})$ for all possible values $\{a_{k^*j}\}$ for attribute $A_{k^*}$. The main task is to determine which attribute $A_k \in R$ classifies the transactions in $T$ best. This relies on a measure for *goodness of split*. In practice, the goodness of split is represented by some function $f$ for which the value $\{f(T, A_k)\}$ is maximal if $A_k$ classifies the transactions in $T$ best. We will discuss two common choices for $f$ in the next section.

### 2.1 Two Common Splitting Rules

We will use two common splitting rules for generating decision trees, based on entropy and based on the Gini index, respectively. See, e.g., [Bre96].

The goodness of split based on entropy was originally used in the ID3 algorithm [Qui86]. The amount of information needed to identify the class of a transaction in a set $T \subseteq \mathcal{T}$ is given by the entropy:

$$H(T) = -\sum_i \frac{|T \cap S_{0,i}|}{|T|} \log \frac{|T \cap S_{0,i}|}{|T|}.$$

Similarly, the amount of information needed to determine the class of a transaction in a set $T$ given attribute $A_k$ is given by the conditional entropy:

$$H(T|A_k) = \sum_j \frac{|T \cap S_{k,j}|}{|T|} H(T \cap S_{k,j}).$$

ID3 is a greedy algorithm that recursively selects the attribute with maximal *information gain*, which is defined by

$$\mathrm{IG}(A_k) = H(T) - H(T|A_k).$$

The best split is defined as the partition of $T$ according to attribute $A_k$ with the highest information gain, or equivalently, with minimal $H(T|A_k)$.

Computing a logarithm securely is in general a complex task and requires specialized protocols to be applicable in practice [LP00]. Instead of computing a logarithm securely we choose to go a different well known splitting measure to avoid secure computation of logarithms. Our protocols will be based on the Gini index, which is another common splitting measure that can be implemented using simple arithmetic only.

The Gini index measures the probability of incorrectly classifying transactions in $T$ if classification is done randomly according to the distribution of the class values in $T$ [RS00], and is given by

$$G(T) = 1 - \sum_i \left( \frac{|T \cap S_{0,i}|}{|T|} \right)^2.$$

Similarly, the estimated conditional probability of incorrectly classifying transactions in $T$ given attribute $A_k$ is given by

$$G(T|A_k) = \sum_j \frac{|T \cap S_{k,j}|}{|T|} G(T \cap S_{k,j}).$$

One can show that $0 \leq G(T|A_k) \leq G(T)$, such that

$$\mathrm{GG}(A_k) = G(T) - G(T|A_k)$$

defines the reduction of incorrect classifications in $T$ given attribute $A_k$. Again, the best split is defined as the partition $T$ according to attribute $A_k$ with the highest Gini gain, or equivalently, with minimal $G(T|A_k)$.

## 3 Secure Computation Framework

We develop our protocols in a generic framework for secure computation. For simplicity, we assume that all secret values are signed integers ranging over $\mathbb{Z}_p = \{-\lfloor p/2 \rfloor, \ldots, -1, 0, 1, \ldots, \lfloor p/2 \rfloor\}$ for a sufficiently large prime $p$. As a concrete instantiation of a secure computation framework we use the Virtually Ideal Functionality Framework (VIFF), basically using Shamir secret sharing over $\mathbb{Z}_p$ to provide $n$-party computation secure against passive adversaries. Any secret value in $\mathbb{Z}_p$ is thus represented by $n$ shares in $\mathbb{Z}_p$, each party holding one share.

We assume that we have efficient integer arithmetic for secret values. As usual, we take the cost of one multiplication $x * y$ as our basic unit of work. The cost of one addition $x + y$ or subtraction $x - y$ is considered negligibly small compared to the cost of one multiplication. Exact division (that is, $x/y$ where $x$ is an integral multiple of $y$) costs about two multiplications.

Secure integer equality $x = y$ and secure integer comparison $x \leq y$ are also assumed to be available. Both of these operations yield a secret bit value, with 0 representing false and 1 representing true, and are at least an order of magnitude more expensive than secure multiplication. In our protocols, we also use the operation arg max to securely find a location of the maximum value in a given list of $N$ secret values, basically using $N - 1$ secure comparisons.

Furthermore, we will assume that secret subsets of a given finite (ordered) set $V$ are represented as secret bit vectors of length $|V|$. For simplicity, we will identify a secret set $A \subseteq V$ with the bit vector representing it. So, for instance, to securely compute $|A|$ it suffices to sum the entries of the bit vector representing $A$, hence this operation is almost for free. Similarly, the *disjoint* union $A \uplus B$ is obtained securely by taking the entrywise sum $A + B$ of the bit vectors representing $A$ and $B$, and the symmetric difference $A \setminus B$ for $B \subseteq A$ is obtained by taking the entrywise difference $A - B$. Moreover, we see that the intersection $A \cap B$ is obtained securely by taking the entrywise product $A \star B$ of the bit vectors representing $A$ and $B$ (at the cost of $|V|$ secure multiplications). Finally, we note that frequently we need to compute only the *size* of the intersection $|A \cap B|$, for which it suffices to take the dot product $A \cdot B$.

We assume that the dot product can be computed securely at the cost of one or at most a few secure multiplications, independent of the length of the vectors (see, e.g., [CdH10], using similar ideas as in [CDI05]). More precisely, the communication cost of a secure dot product is independent of the length of the vectors (whereas the computational cost is still linear in the length of the vectors). The communication cost is the dominating cost factor in a framework such as VIFF. By using dot products judiciously we are able to reduce the total cost of our protocols considerably.

## 4 Secure ID3 Protocol

We present a secure multiparty protocol based on the recursive ID3 algorithm presented in Sect. 2. The goal is to completely hide the contents of the transac-

**Protocol 4.1** SID3($T, R$)

```
1: foreach $i$ do
2:     $s_i = T \cdot S_{0,i}$
3: $i^* = \arg\max_i s_i$
4: if $R = \emptyset$ or ($|T| \leq \epsilon|\mathcal{T}|$ or $s_{i*} = |T|$) then
5:     return $\langle c_{i^*} \rangle$
6: else
7:     foreach $i$ do
8:         $U_i = T \star S_{0,i}$
9:     foreach $k$ s.t. $A_k \in R$ do
10:        foreach $j$ do
11:            foreach $i$ do
12:                $x_{ij} = U_i \cdot S_{k,j}$
13:            $y_j = \alpha \sum_i x_{ij} + 1$
14:        $D_k = \prod_j y_j$
15:        $\widetilde{G}_k = D_k \sum_j (\sum_i x_{ij}^2)/y_j \div D_k$
16:    $k^* = \arg\max_k \widetilde{G}_k$
17:    return $\langle A_{k^*}, \{\text{SID3}(T \star S_{k^*,j}, R \setminus \{A_{k^*}\})\}_j \rangle$
```

tional database except for the information leaked from the decision tree output by the protocol.

Our recursive SID3 protocol is described below, see Protocol 4.1. Given a database containing a set of transactions $\mathcal{T}$ with attributes in $\mathcal{A}$, a decision tree is obtained by the call SID3($\mathcal{T}, \mathcal{A} \setminus \{C\}$), where $C = A_0$ is the class attribute. In general, the recursive protocol SID3($T, R$) takes sets $T \subseteq \mathcal{T}$ and $R \subseteq \mathcal{A} \setminus \{C\}$ as inputs. The decision tree output by the protocol is public, and therefore set $R$ is not secret either. Set $T$ on the other hand is a secret input, represented as a secret bit vector of length $|\mathcal{T}|$.

We will now give a step-by-step description of the SID3 protocol, assuming that the sets $S_{k,j}$ of transactions for which attribute $A_k$ has value $a_{kj}$ are given as secret bit vectors, all of length $|\mathcal{T}|$.

In lines 1–3 we determine the most frequently occurring class value $c_{i^*}$. First, $s_i$ is computed as the number of transactions in $T$ with class value $c_i$ by taking the dot product of the bit vectors representing $T$ and $S_{0,i}$, respectively. Subsequently, a class value $c_{i*}$ such that $s_{i^*}$ is maximal is determined. The value of $i^*$ is public, but no further information on the secret values $s_i$ is leaked.

Lines 4–5 cover the cases in which the decision tree consists of a single node containing value $c_{i*}$. Whether $R = \emptyset$ holds can be evaluated quickly as $R$ is not secret. If $R \neq \emptyset$ (which is usually the case) the test '$|T| \leq \epsilon|\mathcal{T}|$ or $s_{i*} = |T|$' is evaluated securely as follows. Input $T$ is given as a secret bit vector, hence by summing its entries $|T|$ is obtained as a secret value. The value $s_{i^*}$ is secret as well. Subsequently, using a secure comparison, a secure equality test, and a secure 'or', only the value of the test is revealed. This means, in particular, that if the test evaluates to true, it remains hidden whether $|T| \leq \epsilon|\mathcal{T}|$ holds, whether $s_{i*} = |T|$ holds, or whether both conditions hold.

The remaining lines cover the case of a composite decision tree. Lines 7–15 cover the computation of the secret values $\widetilde{G}_k$ which are used to determine an attribute $A_{k^*}$ of highest Gini index in line 16. The resulting decision tree is then computed in line 17, with $A_{k^*}$ as root value, and with a decision tree for transaction set $T \cap S_{k^*,j}$ as $j$th subtree.

The quantities $\widetilde{G}_k$ are used to approximate the quantities $G_k$ sufficiently well, where

$$G_k = \sum_{j \text{ s.t. } |T \cap S_{k,j}| \neq 0} \frac{\sum_i |T \cap S_{0,i} \cap S_{k,j}|^2}{|T \cap S_{k,j}|}.$$

It can be easily seen that finding an attribute of highest Gini index corresponds to maximizing $G_k$ over $A_k \in R$. However, secure computation of $G_k$ requires that the indices $j$ for which $|T \cap S_{k,j}| = 0$ are not revealed. To this end, we will replace the nonnegative values $|T \cap S_{k,j}|$ by positive values $y_j$ such that the resulting quantity $\widetilde{G}_k$ is sufficiently similar to $G_k$, where

$$\widetilde{G}_k = \sum_j \frac{\sum_i x_{ij}^2}{y_j}.$$

Here, entries $x_{ij} = |T \cap S_{0,i} \cap S_{k,j}|$ form a contingency table, and we set $y_j = \alpha|T \cap S_{k,j}| + 1$ for some sufficiently large integer constant $\alpha \geq 1$. In our experiments in Sect. 6 it turns out that $\alpha = 8$ suffices, as compared to the results for the alternative of setting $y_j = |T \cap S_{k,j}|$ if $|T \cap S_{k,j}| > 0$ and $y_j = 1$ otherwise—in which case we have in fact $\widetilde{G}_k = G_k$. We prefer to use $y_j = \alpha|T \cap S_{k,j}| + 1$ as secure evaluation of the alternative for $y_j$ requires a secure equality test, which has a big impact on the performance; a disadvantage of this choice is that we need to increase the size of the field $\mathbb{Z}_p$, as can be seen from the bit lengths used in Table 6.1.

For each attribute $A_k \in R$, the secret values $x_{ij}$ and $y_j$ are computed efficiently as follows. First, the bit vectors $U_i$ representing the intersections $T \cap S_{0,i}$ are computed as entrywise products of the bit vectors representing $T$ and $S_{0,i}$. Then each $x_{ij}$ is obtained as the dot product of the bit vector $U_i$ and the bit vector representing $S_{k,j}$, and we set $y_j = \alpha \sum_i x_{ij} + 1$.

Finally, to avoid secure arithmetic over rational numbers, we take the common denominator of all terms in the sum $\widetilde{G}_k$:

$$\widetilde{G}_k = \frac{\sum\limits_j \sum\limits_i x_{ij}^2 \prod\limits_{l \neq j} y_l}{\prod\limits_l y_l}.$$

This way, both the numerator and denominator of $\widetilde{G}_k$ are integers, and we can maximize $\widetilde{G}_k$ using integer arithmetic only, as $x \div y \leq x' \div y'$ is equivalent to $xy' \leq x'y$ for $y, y' > 0$. The test $xy' \leq x'y$ is further optimized by actually evaluating $(x, y) \cdot (y', -x') \leq 0$, hence using a single dot product instead of two multiplications. Of course, the terms $\sum\limits_i x_{ij}^2$ are also each computed using a single dot product.

In the actual code used in the experiments of Sect. 6 we have applied some further optimizations throughout. For instance, since $\sum_i U_i = T$, one can save one entrywise product in lines 7–8, which speeds up this part by a factor of two in case the class attribute takes on two values only. Similarly, one entrywise product can be saved in line 17.

## 5 Secure ID3 in Other Settings

We show how minor changes to SID3 allow efficient generation of secret decision trees. In addition, we show that if the database is *horizontally* partitioned between the parties, then minor changes to SID3 allow generation of a public decision tree with communication complexity that is independent of the number of transactions in the database.

### 5.1 Secret Output and Secret Prediction

There are some serious restrictions when hiding the resulting decision tree. Firstly, when any third party is allowed to ask for decisions from the secret tree, it may be able to reconstruct or build an equivalent tree by querying the tree often enough. A strategy could be, for example, to generate its own database by querying the secret tree, and apply ID3 to the generated database.

Secondly, not revealing any information about the decision tree requires hiding the shape of the tree. This would lead to a tree of worst case size, which is exponential. Indeed, a database with $m$ attributes each taking possibly $\ell$ values has at most $\ell^m$ leaves. Moreover, in this case it is useless to apply ID3: one could simply compute the best class for all possible $\ell^m$ paths. The resulting tree is of maximum size as required and can be computed much more efficiently by just partitioning the database into all possible paths along the attribute values. More precisely, one would run SID3S$(T, m, \perp)$, where $\perp$ indicates that there is nothing to output when the original database $T$ is empty, see Protocol 5.1.

In line 4 of SID3S the index $i^*$ of the most frequent class value in $T$ is computed similar to line 3 of SID3. However, $i^*$ should not be revealed. Therefore, we use its secret unary representation, which is a vector containing zeros only, except at position $i^*$, where it contains a 1. Thus, to hide $i^*$ we apply a variant of $\arg\max_i$ that returns a length $|\{c_j\}|$ secret unary representation of the value $i^*$, say $\boldsymbol{i}^*$. Then $c_{i^*}$ can be computed securely and without interaction by the dot product $(c_1, \ldots, c_{|\mathcal{C}|}) \cdot \boldsymbol{i}^*$, since $\{c_i\}$ is public. This is applied in lines 1–4 of SID3S.

As a tradeoff between security and efficiency one could choose to reveal some information on the shape of the tree, e.g., the length of the paths. This avoids exponential growth of the tree. In this case we need to take care of the following two things: Firstly, we cannot reveal the attribute representing the next best split and leaf values as this would leak the entire decision tree. Secondly, we should ensure that all attributes take the same number of values. Indeed, one could learn information about the attribute label of each non-leaf node by observing

**Protocol 5.1** SID3S($T, k, c$)

```
1: if $|T| \neq 0$ then
2:     foreach $i$ do
3:         $s_i = T \cdot S_{0,i}$
4:     $i^* = \arg\max_i s_i$
5:     $c = c_{i^*}$
6: if $k = 0$ then
7:     return $\langle c \rangle$
8: else
9:     return $\langle A_k, \{\mathrm{SID3S}(T \star S_{k,j}, k-1, c)\}_j \rangle$
```

the number of children it has. The latter can be ensured simply by adding dummy values to each attribute.

Thus, ID3 is applied as before, except for opening the values of the leaves and opening the values of the next best split. Not opening the values of the next best split leads to a bit more complicated partitioning of the tree. Fore example, we need to prevent a selected attribute to be selected again in some subsequent call to ID3. Protocol 5.2 computes the secret decision tree for $\mathcal{T}$ and reveals only the depth of each path. We will discuss line by line the changes with respect to SID3.

Firstly, as we observed in SID3S, the index $i^*$ of the most frequent class value in $T$ is computed similar to line 3 of SID3, but should not be revealed. So, in lines 1–5 of SID3T we again apply the variant of $\arg\max_i$ that returns a length $|\{c_j\}|$ secret unary representation of the value $i^*$, such that $c_{i^*}$ can be computed securely and without interaction using a dot product.

Secondly, instead of $R \subseteq \mathcal{A}$ being public it should be secret to avoid revealing which attribute is selected in previous recursions. This will affect lines 4, 16, and 17 of SID3. We let $R$ be represented by a secret bit vector, where its $k$th entry is equal to $[A_k \in R]$ with $[\text{true}] = 1$ and $[\text{false}] = 0$.

In line 4 of SID3 one checks whether $R = \emptyset$. However, since $R$ is secret it cannot be used to perform this check. To check whether $R = \emptyset$ without communication, observe that $R = \emptyset$ if and only if the current path is maximal, or, equivalently, when the recursive call to ID3 is in depth $|\mathcal{A}| - 1$. Therefore, we use a public counter $r$ that is initialized to $|\mathcal{A}| - 1$ and decreases by one after each recursive call to ID3. The condition $R = \emptyset$ is replaced by $r = 0$, see line 4 of SID3T.

Line 16 of SID3 computes and reveals the attribute with the best Gini index among the available attributes given by $R$. To ensure selection among the available attributes in the secret set $R$ we proceed as follows. First we compute $\widetilde{G}_k$ for all $k$, and then we choose attribute $A_{k^*}$ obliviously such that $\widetilde{G}_{k^*} - [A_k \notin R]$ is maximal, see line 16 of SID3T. This ensures selection of an attribute with maximal $\widetilde{G}_k$ that has not been selected already. Indeed, if attribute $A_k$ has already been selected then its value in all transactions con-

**Protocol 5.2** SID3T$(T, R, r)$

```
 1: foreach $i$ do
 2:   $s_i = T \cdot S_{0,i}$
 3: $i^* = \arg\max_i s_i$
 4: if $r = 0$ or $(|T| \leq \epsilon|\mathcal{T}|$ or $s_{i*} = |T|)$ then
 5:   return $\langle c_{i^*} \rangle$
 6: else
 7:   foreach $i$ do
 8:     $U_i = T \star S_{0,i}$
 9:   foreach $k$ do
10:     foreach $j$ do
11:       foreach $i$ do
12:         $x_{ij} = U_i \cdot S_{k,j}$
13:       $y_j = \alpha \sum_i x_{ij} + 1$
14:     $D_k = \prod_j y_j$
15:     $\widetilde{G}_k = D_k \sum_j (\sum_i x_{ij}^2)/y_j \div D_k$
16:   $k^* = \arg\max_k \widetilde{G}_k - [A_k \notin R]$
17:   return $\langle A_{k^*}, \{\text{SID3T}(T \star S_{k^*,j}, R \setminus \{A_{k^*}\}, r-1)\}_j \rangle$
```

sidered by successive recursive calls to ID3 is constant, so that $\widetilde{G}_k = 0$ and $\widetilde{G}_k - [A_k \notin R] = -1 < 0 \leq \widetilde{G}_v - [A_v \notin R]$ for any available attribute $A_v$.

Since $A_{k^*}$ should remain secret, in line 16 we apply again the variant of $\arg\max_i$ that returns a length $|\mathcal{A} - 1|$ secret unary representation of the value $k^*$, say $\boldsymbol{k}^*$. We let $A_{k^*}$ be represented by the secret unary representation of its index $k^*$. To update $T$ by $T \star S_{k^*,j}$, in line 17, we first need to compute $S_{k^*,j}$, which is done using the following dot product

$$S_{k^*,j} = \left(S_{1,j}, S_{2,j}, \ldots, S_{|\mathcal{A}|-1,j}\right) \cdot \boldsymbol{k}^*,$$

which is interactive, since both $S_{i,j}$ and $\boldsymbol{k}^*$ are secret.

Finally, in line 17 of SID3, $R \setminus \{k^*\}$ can be easily computed since $R$ nor $A_{k^*}$ are secret. Updating the secret representation of $R$ in SID3T can be done without interaction by the entrywise subtraction by the secret unary representation of $k^*$. Indeed, $R \setminus \{A_{k^*}\}$ is equivalent to setting the bit $[A_{k^*} \in R]$ to zero. Let $\boldsymbol{k}^*$ be the secret unary representation of $k^*$ then the entrywise subtraction of the secret representation of $R$ by $\boldsymbol{k}^*$ will only affect the $k^*$th entry of the secret bit vector for $R$, where it becomes $[A_{k^*} \in R] - 1$. Since $A_{k^*}$ is selected it was available so that $[A_{k^*} \in R] = 1$, and subtraction by one will result in $[A_{k^*} \in R] = 0$ as required in the next recursive call.

With respect to complexity, selecting the next best attribute requires $|\mathcal{A}| - 2$ secure comparisons in each recursive call as opposed to only $|R| - 1$ secure comparisons in SID3. Computing $S_{k^*,j}$ requires $\ell|\mathcal{T}|$ secure dot products in addition to the $\ell|\mathcal{T}|$ multiplications for computing $T \star S_{k^*,j}$.

Protocol 5.3 computes the class from a secret tree $B$, given a secret transaction $t$ as follows. Let $t$ be a secret transaction, where its $k$th entry is a unary representation of the index of the value $t(A_k)$. So the $j$th value of the $k$th entry

**Protocol 5.3** Class$(t, B)$

```
1: if $B = \langle c \rangle$ then
2:     return $c$
3: else
4:     $m = t \cdot B_1$
5:     return $\sum_j m_j \cdot \text{Class}(t, B_{2j})$
```

of $t$ is equal to 1 if $t(A_k) = a_{kj}$ and it is equal to 0 otherwise. Note that $B = \langle c \rangle$ if $B$ is a single leaf node and $B = \langle B_1, B_2 \rangle = \langle A_k, (B_{21}, \ldots, B_{2\ell}) \rangle$ otherwise, where $B_{2j}$ is the resulting tree of SID3T$(S_{k,j}, R \setminus A_k, |\mathcal{A}| - 2)$. Recall that $A_k$ is secret and is represented by the length $|\mathcal{A}| - 1$ secret unary representation of its index $k$.

If $B = \langle c \rangle$, then $t$ gets class $c$, else $t$ gets class $\sum_j (t \cdot B_1)_j \text{Class}(t, B_{2j})$.

## 5.2 Horizontally Partitioned Database

If the database $\mathcal{T}$ is horizontally partitioned and if the resulting tree is made public, then there is no need to securely split the database by computing a mask. Given a set of transactions, each party can *locally* compute any partition of $\mathcal{T}$ according to some attribute. Hence, the communication complexity will be independent of $|\mathcal{T}|$, which is a significant improvement in practice where $|\mathcal{T}|$ is relatively large compared to $|\mathcal{A}|$. Checking the stopping criteria and computing the Gini index, however, requires knowledge of the entire database and requires interaction.

Let $\{\mathcal{T}_z\}$ be the partition of $\mathcal{T}$ such that each $P_z$ owns $\mathcal{T}_z$. Observe that $\{S_{z:k,j}\}$, where $S_{z:k,j} = \{t \in \mathcal{T}_z : t(A_k) = a_{kj}\}$, is a partition of $S_{k,j}$ where each block $S_{z:k,j}$ can be computed by party $P_z$ locally. Furthermore, if $\{T_z\}$ is a horizontal partition of some $T \subseteq \mathcal{T}$, then $\{T_z \cap S_{z:k,j}\}$ forms a partition of $T \cap S_{k,j}$. To jointly compute $|T \cap S_{k,j}|$, each party $P_z$ computes first $|T_z \cap S_{z:k,j}|$ and shares the result with all other parties. Then all parties locally compute shares of $|T \cap S_{k,j}| = \sum_z |T_z \cap S_{z:k,j}|$ by summing the received shares. This affects lines 2 and 12 in SID3.

Protocol 5.4 shows how to securely compute the decision tree for a horizontally partitioned $\mathcal{T}$. With id we denote the identity of the party running the protocol.

With respect to efficiency, computing the entries of the contingency table $x_{ij}$ requires each party to share their local contingency table. With respect to communication, this is equivalent to performing $|\mathcal{A}-1||\{c_j\}|$ dot products, which is the same as for the computation of the contingency table in ID3. However, splitting the database requires no interaction anymore. This saves $O(|\mathcal{T}|)$ secure multiplications. In fact, the communication complexity of the resulting protocol is independent of $|\mathcal{T}|$.

**Protocol 5.4** SID3P($T, R$)

```
foreach $i$ do
    $s_{\text{id}:i} = \text{Share}(T_z \cdot S_{z:0,i})$
    foreach $P_z \neq P_{\text{id}}$ do
        Receive($s_{z:i}$)
    $s_i = \sum_z s_{z:i}$
$i^* = \arg\max_i \; s_i$
if $R = \emptyset$ or ($|T| \leq \epsilon|\mathcal{T}|$ or $s_{i*} = |T|$) then
    return $\langle c_{i^*} \rangle$
else
    foreach $i$ do
        $U_{\text{id}:i} = T_z \star S_{\text{id}:0,i}$
    foreach $k$ s.t. $A_k \in R$ do
        foreach $i$ do
            foreach $j$ do
                $x_{\text{id}:ij} = \text{Share}(U_{\text{id}:i} \cdot S_{\text{id}:k,j})$
        foreach $j$ do
            foreach $i$ do
                foreach $P_z \neq P_{\text{id}}$ do
                    Receive($x_{z:ij}$)
                $x_{ij} = \sum_z x_{z:ij}$
            $y_j = \alpha \sum_i x_{ij} + 1$
        $D_k = \prod_j y_j$
        $\widetilde{G}_k = D_k \sum_j (\sum_i x_{ij}^2)/y_j \div D_k$
    $k^* = \arg\max_k \; \widetilde{G}_k$
    return $\langle A_{k^*}, \{\text{SID3P}(T \star S_{k^*,j}, R \setminus \{A_k\})\}_j \rangle$
```

## 6 Performance Results

To analyze the performance of our protocols in a practical setting, we have built applications using the Virtual Ideal Functionality Framework (VIFF). VIFF is a general software framework for doing secure multiparty computation [Gei10], which provides researchers and programmers with the basic building blocks (or sub-protocols) as APIs to allow rapid prototyping of new protocols and building practical applications. For improved efficiency, we use the 'boost' extension to VIFF, which greatly improves the performance of VIFF applications [Kel10]. The comparison protocols applied are the probabilistic equality test from [NO07] and the integer comparison from [EFG+09].

We run the protocols for three players on different network ports on a 64-bit Windows 7 PC, with Intel Core i5-3470 CPU @3.20GHz (2 cores, 4 hyper-threads), 16GB memory. Even on such a moderately fast PC and even though the performance overhead of VIFF is intrinsically large, the absolute timings range from a few seconds to a few minutes only, showing the practical feasibility of our approach. A marked advantage of VIFF specifically for implementing secure ID3 protocols is the fact scheduling is done dynamically at runtime, *depending* on the shape of the decision tree as it develops.

Table 6.1: Performance results

| Data | Size | Measure | bit length | SID3 | SID3T | SID3P |
|---|---|---|---|---|---|---|
| SPECT | 267 | $\widetilde{G}_k$ | 41 | 27s | 43s | 24s |
| | | $G_k$ | 32 | 57s | 88s | 54s |
| Scale | 625 | $\widetilde{G}_k$ | 76 | 9s | 17s | 7s |
| | | $G_k$ | 49 | 11s | 17s | 8s |
| Car | 1728 | $\widetilde{G}_k$ | 95 | 18s | 29s | 10s |
| | | $G_k$ | 74 | 20s | 33s | 12s |
| KRKPA7 | 3196 | $\widetilde{G}_k$ | 69 | 46s | 104s | 26s |
| | | $G_k$ | 57 | 73s | 142s | 50s |
| [AJH10] | 2196 | $\widetilde{G}_k$ | 78 | 68s | 185s | 40s |
| | | $G_k$ | 63 | 96s | 255s | 69s |

We have tested the performance of our ID3 protocols with the benchmarking data set from the UCI Machine Learning Repository [FA10] and with the data set from [AJH10]. Table 6.1 shows the performance results of our protocols. The threshold for early stopping is set to $\epsilon = 5\%$ of the data set. The parameter for computing $\widetilde{G}_k$ is set to $\alpha = 8$, which is sufficiently large to ensure that the protocols return basically the same decision trees as obtained using $G_k$ (the decision trees are identical for all data sets, except for SPECT, where some minor differences are visible).

Note that the required size modulus $p$ of the prime field is affected by $\alpha$. Indeed, the size of each $x_{ij}$ is increased by $\log_2(\alpha)$ so that the size $y_j$ (Line 13 of SID3) is increased by at most $\ell \log_2(\alpha)$ bits, where $\ell$ denotes the maximum number of values an attribute from $\mathcal{A}$ takes. This in turn affects the communication complexity of the integer comparisons which are proportional to the given bit length of the inputs. The bit length $b$ in Table 6.1 denotes the number of bits required to simulate integer arithmetic over $\mathbb{Z}_p$. In our experiments, the statistical security parameter is set to 30 bits. As a consequence the prime $p$ is chosen such that $\log_2 p \approx b + 31$.

## Acknowledgements

This work was supported by the Dutch national program COMMIT.

## References

AJH10. H. op den Akker, V.M. Jones, and H.J. Hermens. Predicting feedback compliance in a teletreatment application. In *Proceedings of ISABEL 2010: the 3rd International Symposium on Applied Sciences in Biomedical and Communication Technologies*, Rome, Italy, 2010.

AS00. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2000, pages 439–450, New York, NY, USA, 2000. ACM.

Bre96. L. Breiman. Technical note: Some properties of splitting criteria. *Machine Learning*, 24:41–47, 1996.

BTW12. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. In *Proceedings of the Sixteenth International Conference on Financial Cryptography and Data Security*, FC '12, 2012.

CdH10. Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 134–150, 2010.

CDI05. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 342–362, Berlin, Heidelberg, 2005. Springer-Verlag.

DZ02. Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, pages 1–8. Australian Computer Society, Inc., 2002.

EFG$^+$09. Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 235–253, Berlin, Heidelberg, 2009. Springer-Verlag.

FA10. A. Frank and A. Asuncion. UCI machine learning repository, 2010.

Gei10. Martin Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, Denmark, February 2010.

Kel10. Marcel Keller. VIFF boost extension. http://lists.viff.dk/pipermail/viff-devel-viff.dk/2010-August/000847.html, 2010.

LP00. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54, London, UK, UK, 2000. Springer-Verlag.

MD08. Qingkai Ma and Ping Deng. Secure multi-party protocols for privacy preserving data mining. In Yingshu Li, DungT. Huynh, SajalK. Das, and Ding-Zhu Du, editors, *Wireless Algorithms, Systems, and Applications*, volume 5258 of *Lecture Notes in Computer Science*, pages 526–537. Springer Berlin Heidelberg, 2008.

MGA12. Majid Bashir Malik, M. Asger Ghazi, and Rashid Ali. Privacy preserving data mining techniques: Current scenario and future prospects. In *Proceedings of the 2012 Third International Conference on Computer and Communication Technology*, ICCCT '12, pages 26–32, Washington, DC, USA, 2012. IEEE Computer Society.

NO07. T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Proceedings of the 10th international conference on Practice and theory in public-key cryptography*, PKC'07, pages 343–360, Berlin, Heidelberg, 2007. Springer-Verlag.

Qui86. J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

RM05. Lior Rokach and Oded Maimon. Decision trees. In *The Data Mining and Knowledge Discovery Handbook*, pages 165–192. Springer US, 2005.

RS00. Laura E. Raileanu and Kilian Stoffel. Theoretical comparison between the Gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41:77–93, 2000.

SM08. Saeed Samet and Ali Miri. Privacy preserving ID3 using Gini index over horizontally partitioned data. In *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 645–651. IEEE, 2008.

VCKP08. Jaideep Vaidya, Chris Clifton, Murat Kantarcıoğlu, and A. Scott Patterson. Privacy-preserving decision trees over vertically partitioned data. *ACM Trans. Knowl. Discov. Data*, 2(3):14:1–14:27, October 2008.

WXSY06. Ke Wang, Yabo Xu, Rong She, and Philip S Yu. Classification spanning private databases. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 21, page 293. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

XHLS05. Ming-Jun Xiao, Liu-Sheng Huang, Yong-Long Luo, and Hong Shen. Privacy preserving ID3 algorithm over horizontally partitioned data. In *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, pages 239–243. IEEE, 2005.

Yao86. A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS '86)*, pages 162–167. IEEE Computer Society, 1986.