Privacy Preserving Data Mining

Yehuda Lindell[1] and Benny Pinkas[2*]

[1] Department of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot, ISRAEL. lindell@wisdom.weizmann.ac.il

[2] School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem, ISRAEL. bpinkas@cs.huji.ac.il

Abstract. In this paper we introduce the concept of privacy preserving data mining. In our model, two parties owning confidential databases wish to run a data mining algorithm on the union of their databases, without revealing any unnecessary information. This problem has many practical and important applications, such as in medical research with confidential patient records.

Data mining algorithms are usually complex, especially as the size of the input is measured in megabytes, if not gigabytes. A generic secure multi-party computation solution, based on evaluation of a circuit computing the algorithm on the entire input, is therefore of no practical use. We focus on the problem of decision tree learning and use ID3, a popular and widely used algorithm for this problem. We present a solution that is considerably more efficient than generic solutions. It demands very few rounds of communication and reasonable bandwidth. In our solution, each party performs by itself a computation of the same order as computing the ID3 algorithm for its own database. The results are then combined using efficient cryptographic protocols, whose overhead is only logarithmic in the number of transactions in the databases. We feel that our result is a substantial contribution, demonstrating that secure multi-party computation can be made practical, even for complex problems and large inputs.

1 Introduction

We consider a scenario where two parties having private databases wish to cooperate by computing a data mining algorithm on the union of their databases. Since the databases are confidential, neither party is willing to divulge any of the contents to the other. We show how the involved data mining problem of decision tree learning can be efficiently computed, with no party learning anything other than the output itself. We demonstrate this on ID3, an algorithm widely used and implemented in many real applications.

Confidentiality Issues in Data Mining. A key problem that arises in any en masse collection of data is that of *confidentiality*. The need for secrecy is sometimes due to law (e.g. for medical databases) or can be motivated by business interests. However, sometimes there can be mutual gain by *sharing* of data. A

* Supported by an Eshkol grant of the Israel Ministry of Science.

key utility of large databases today is research, whether it be scientific, or economic and market oriented. The medical field has much to gain by pooling data for research; as can even competing businesses with mutual interests. Despite the potential gain, this is not possible due to confidentiality issues which arise.

We address this question and show that practical solutions are possible. Our scenario is one where two parties P_1 and P_2 own databases D_1 and D_2. The parties wish to apply a data-mining algorithm to the joint database $D_1 \cup D_2$ without revealing any unnecessary information about their individual databases. That is, the only information learned by P_1 about D_2 is that which can be learned from the output, and vice versa. We do not assume any "trusted" third party who computes the joint output.

Very Large Databases and Efficient Computation. We have described a model which is exactly that of multi-party computation. Therefore, there exists a secure solution for *any* functionality (Goldreich et. al. in [13]). As we discuss in Section 1.1, due to the fact that these solutions are generic, they are highly inefficient. In our case where the inputs are very large and the algorithms reasonably complex, they are far from practical.

It is clear that any reasonable solution must have the individual parties do the majority of the computation independently. Our solution is based on this guiding principle and in fact, the number of bits communicated is dependent on the number of transactions by a logarithmic factor only.

Semi-Honest Parties. In any multi-party computation setting, a *malicious* party can always alter his input. In the data-mining setting, this fact can be very damaging as an adversarial party may define his input to be the empty database. Then, the output obtained is the result of the algorithm on the other party's database alone. Although this attack cannot be prevented, we would like to limit attacks by malicious parties to altering their input only. However, for this initial work we assume that the parties are *semi-honest* (also termed *passive*). That is, they follow the protocol as it is defined, but may record all intermediate messages sent in an attempt to later derive additional information. We leave the question of an efficient solution to the malicious party setting for future work. In any case, as was described above, malicious parties cannot be prevented from obtaining meaningful confidential information and therefore a certain level of trust is anyway needed between the parties. We remark that the semi-honest model is often a realistic one; that is, deviating from a specified program which may be buried in a complex application is a non-trivial task.

1.1 Related Work

Secure two party computation was first investigated by Yao [21], and was later generalized to multi-party computation in [13, 2, 5]. These works all use a similar methodology: the function F to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. While this approach is appealing in its generality and simplicity, the protocols it generates depend on the size of the circuit. This size depends on the size of the input (which might be huge as in a data mining application), and on the complexity of expressing F as a circuit (for example, a multiplication circuit is quadratic in the size of its inputs). We stress that secure computation of small circuits with small inputs can be *practical* using the [21] protocol.[1]

There is a major difference between the protocol described in this paper and other examples of multi-party protocols (e.g. [3, 11, 6]). While previous protocols were efficient (polynomial) in the size of their inputs, this property does not suffice for data mining applications, as the input consists of huge databases. In the protocol presented here, most of the computation is done individually by each of the parties. They then engage in a few secure circuit evaluations on very small circuits. We obtain very few rounds of communication with bandwidth which is practical for even very large databases.

Outline: The next section describes the problem of classification and a widely used solution to it, decision trees. Following this, Section 3 presents the security definition and Section 4 describes the cryptographic tools used in the solution. Section 5 contains the protocol itself and its proof of security. Finally, the main subprotocol that privately computes random shares of $f(v_1, v_2) \stackrel{\text{def}}{=} (v_1 + v_2)\ln(v_1 + v_2)$ is described in Section 6.

2 Classification by Decision Tree Learning

This section briefly describes the machine learning and data mining problem of *classification* and ID3, a well-known algorithm for it. The presentation here is rather simplistic and very brief and we refer the reader to Mitchell [15] for an in-depth treatment of the subject. The ID3 algorithm for generating decision trees was first introduced by Quinlan in [19] and has since become a very popular learning tool.

The aim of a classification problem is to classify transactions into one of a discrete set of possible categories. The input is a structured database comprised of attribute-value pairs. Each row of the database is a *transaction* and each column is an *attribute* taking on different values. One of the attributes in the database is designated as the *class* attribute; the set of possible values for this attribute being the classes. We wish to predict the class of a transaction by viewing only the non-class attributes. This can thus be used to predict the class of new transactions for which the class is unknown.

For example, a bank may wish to conduct credit risk analysis in an attempt to identify non-profitable customers before giving a loan. The bank then defines "Profitable-customer" (obtaining values "yes" or "no") to be the class attribute. Other database attributes may include: Home-Owner, Income, Years-of-Credit, Other-Delinquent-Accounts and other relevant information. The bank is then interested in obtaining a tool which can be used to classify a *new* customer as potentially profitable or not. The classification may also be accompanied with a probability of error.

[1] The [21] protocol requires only two rounds of communication. Furthermore, since the circuit and inputs are small, the bandwidth is not too great and only a reasonable number of oblivious transfers need be executed.

2.1 Decision Trees and the ID3 Algorithm

A decision tree is a rooted tree containing nodes and edges. Each internal node is a test node and corresponds to an attribute; the edges leaving a node correspond to the possible values taken on by that attribute. For example, the attribute "Home-Owner" would have two edges leaving it, one for "Yes" and one for "No". Finally, the leaves of the tree contain the *expected* class value for transactions matching the path from the root to that leaf.

Given a decision tree, one can predict the class of a new transaction t as follows. Let the attribute of a given node v (initially the root) be A, where A obtains possible values $a_1, ..., a_m$. Then, as described, the m edges leaving v are labeled $a_1, ..., a_m$ respectively. If the value of A in t equals a_i, we simply go to the son pointed to by a_i. We then continue recursively until we reach a leaf. The class found in the leaf is then assigned to the transaction.

The following notation is used: R: a set of *attributes*; C: the *class* attribute and T: a set of *transactions*. The ID3 algorithm assumes that each attribute is categorical, that is containing discrete data only, in contrast to continuous data such as age, height etc.

The principle of the ID3 algorithm is as follows:

The tree is constructed top-down in a recursive fashion. At the root, each attribute is tested to determine how well it alone classifies the transactions. The "best" attribute (to be discussed below) is then chosen and we partition the remaining transactions by it. We then recursively call ID3 on each partition (which is a smaller database containing only the appropriate transactions and without the splitting attribute). See Figure 1 for a description of the ID3 algorithm.

ID3(R, C, T)

1. If R is empty, return a leaf-node with the class value of the majority of the transactions in T.
2. If T consists of transactions with all the same value c for the class attribute, return a leaf-node with the value c (finished classification path).
3. Otherwise,
   (a) Find the attribute that *best* classifies the transactions in T, let it be A.
   (b) Let $a_1, ..., a_m$ be the values of attribute A and let $T(a_1), ..., T(a_m)$ be a partition of T s.t. every transaction in $T(a_i)$ has the attribute value a_i.
   (c) Return a tree whose root is labeled A (this is the test attribute) and has edges labeled $a_1, ..., a_m$ such that for every i, the edge a_i goes to the tree $\text{ID3}(R - \{A\}, C, T(a_i))$.

Fig. 1. The ID3 Algorithm for Decision Tree Learning

What remains is to explain how the *best* predicting attribute is chosen. This is the central principle of ID3 and is based on information theory. The entropy of the class attribute clearly expresses the difficulty of prediction. We know the class of a set of transactions when the class entropy for them equals zero. The idea is therefore to check which attribute reduces the information of the class-attribute by the most. This results in a greedy algorithm which searches for a small decision tree consistent with the database. As a result of this, decision trees are usually relatively small, even for large databases.

The exact test for determining the best attribute is defined as follows. Let $c_1, ..., c_\ell$ be the class-attribute values. Let $T(c_i)$ be the set of transactions with class c_i. Then the information needed to identify the class of a transaction in T is the entropy, given by:

$$H_C(T) = \sum_{i=1}^{\ell} -\frac{|T(c_i)|}{|T|} \log \frac{|T(c_i)|}{|T|}$$

Let A be a non-class attribute. We wish to quantify the information needed to identify the class of a transaction in T *given* that the value of A has been obtained. Let A obtain values $a_1, ..., a_m$ and let $T(a_j)$ be the transactions obtaining value a_j for A. Then, the conditional information of T given A is given by:

$$H_C(T|A) = \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} H_C(T(a_j))$$

Now, for each attribute A the information-gain[2], is defined by

$$\mathrm{Gain}(A) \stackrel{\mathrm{def}}{=} H_C(T) - H_C(T|A)$$

The attribute A which has the maximum gain over all attributes in R is then chosen.

Since its inception there have been many extensions to the original ID3 algorithm, the most well-known being C4.5. We consider only the simpler ID3 algorithm and leave extensions to more advanced versions for future work.

2.2 The ID3$_\delta$ Approximation

The ID3 algorithm chooses the "best" predicting attribute by comparing entropies that are given as real numbers. If at a given point, two entropies are very close together, then the two (different) trees resulting from choosing one attribute or the other are expected to have almost the same predicting capability. Formally stated, let δ be some small value. Then, for a pair of attributes A_1 and A_2, we say that A_1 and A_2 have *δ-equivalent information gains* if

$$|H_C(T|A_1) - H_C(T|A_2)| < \delta$$

This definition gives rise to an approximation of ID3. Denote by $\mathcal{ID}3_\delta$ the set of all possible trees which are generated by running the ID3 algorithm, and choosing either A_1 or A_2 in the case that they have δ-equivalent information gains. We actually present a protocol for secure computation of a specific algorithm $\mathrm{ID3}_\delta \in \mathcal{ID}3_\delta$, in which the choice of A_1 or A_2 is implicit by an approximation that is used instead of the log function. The value of δ influences the efficiency, but only by a logarithmic factor.

[2] Note that the gain measure biases attributes with many values and another measure called the *Gain Ratio* is therefore sometimes used. We present the simpler version here.

3 Security Definition – Private Computation of Functions

The model for this work is that of general multi-party computation, more specifically between two *semi-honest* parties. Our formal definitions here are according to Goldreich in [12]. We now present in brief the definition for general two-party computation of a functionality with *semi-honest* parties only. We present a formalization based on the simulation paradigm (this is equivalent to the ideal-model definition in the semi-honest case).

Formal Definition. The following definitions are taken from [12]. We begin with the following notation:

- Let $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ be a functionality where $f_1(x,y)$ *(resp.,* $f_2(x,y)$*)* denotes the first *(resp., second)* element of $f(x,y)$ and let Π be a two-party protocol for computing f.
- The view of the first *(resp., second)* party during an execution of Π on (x,y), denoted $\mathrm{view}_1^{\Pi}(x,y)$ *(resp.,* $\mathrm{view}_2^{\Pi}(x,y)$*)*, is $(x,r,m_1,...,m_t)$ *(resp.,* $(y,r,m_1,...,m_t)$*)* where r represents the outcome of the first *(resp., second)* party's *internal* coin tosses, and m_i represents the i'th message it has received.
- The output of the first *(resp., second)* party during an execution of Π on (x,y) is denoted $\mathrm{output}_1^{\Pi}(x,y)$ *(resp.,* $\mathrm{output}_2^{\Pi}(x,y)$*)*, and is implicit in the party's view of the execution.

We note that in the case of $\mathrm{ID3}_\delta$ itself, we have $f_1 = f_2 = \mathrm{ID3}_\delta$ (however, in the subprotocols that we use it is often the case that $f_1 \neq f_2$).

Definition 1 (privacy w.r.t. semi-honest behavior): *For a functionality f, we say that Π* privately computes *f if there exist probabilistic polynomial time algorithms, denoted S_1 and S_2, such that*

$$\{(S_1(x, f_1(x,y)), f_2(x,y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{(\mathrm{view}_1^{\Pi}(x,y), \mathrm{output}_2^{\Pi}(x,y))\right\}_{x,y\in\{0,1\}^*} \quad (1)$$

$$\{(f_1(x,y), S_2(y, f_2(x,y)))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{(\mathrm{output}_1^{\Pi}(x,y), \mathrm{view}_2^{\Pi}(x,y))\right\}_{x,y\in\{0,1\}^*} \quad (2)$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability.

Equations (1) and (2) state that the views of the parties can be simulated by a polynomial time algorithm given access to the party's *input and output only.* We emphasize that the parties here are semi-honest and the view is therefore exactly according to the protocol definition. We note that it is not enough for the simulator S_1 to generate a string indistinguishable from $\mathrm{view}_1^{\Pi}(x,y)$. Rather, the *joint distribution* of the simulator's output and $f_2(x,y)$ must be indistinguishable from $(\mathrm{view}_1^{\Pi}(x,y), \mathrm{output}_2^{\Pi}(x,y))$. See [12] for a discussion on why this is essential.

Composition of Private Protocols. The protocol for privately computing $\mathrm{ID3}_\delta$ is composed of many invocations of smaller private computations. In particular, we reduce the problem to that of privately computing smaller subproblems and show how to compose them together in order to obtain a complete $\mathrm{ID3}_\delta$ solution. This composition is shown to be secure in Goldreich [12].

3.1 Secure Computation of Approximations

Our work takes $\mathrm{ID3}_\delta$ as the starting point and security is guaranteed relative to the approximated algorithm, rather than to ID3 itself. We present a secure protocol for computing $\mathrm{ID3}_\delta$. That is, P_1 can compute his view given D_1 and $\mathrm{ID3}_\delta(D_1 \cup D_2)$ only (likewise P_2). However, this does *not* mean that $\mathrm{ID3}_\delta(D_1 \cup D_2)$ reveals the "same" information as $\mathrm{ID3}(D_1 \cup D_2)$ does. In fact, it is clear that although the computation of $\mathrm{ID3}_\delta$ is secure, *different* information is revealed (intuitively though, no "more" information is revealed)[3].

The problem of secure distributed computation of approximations was introduced and discussed by Feigenbaum et. al. [10]. Their main motivation is a scenario in which the computation of an approximation to a function f might be considerably more efficient than the computation of f itself. The security definition requires that the approximation does not reveal more about the inputs than f does. In addition, the paper presents several general techniques for computing approximations, and efficient protocols for computing approximations of distances.

4 Cryptographic Tools

4.1 Oblivious Transfer

The notion of 1-out-2 oblivious transfer (OT_1^2) was suggested by Even, Goldreich and Lempel [8], as a generalization of Rabin's "oblivious transfer" [20]. This protocol involves two parties, the *sender* and the *receiver*. The sender has two inputs $\langle X_0, X_1 \rangle$, and the receiver has an input $\sigma \in \{0,1\}$. At the end of the protocol the receiver should learn X_σ and no other information, and the sender should learn nothing. Very attractive non-interactive OT_1^2 protocols were presented in [1]. More recent results in [17] reduce the amortized overhead of OT_1^2, and describe non-interactive OT_1^2 of strings whose security is not based on the "random oracle" assumption. Oblivious transfer protocols can be greatly simplified if the parties are assumed to be semi-honest, as they are in the application discussed in this paper.

4.2 Oblivious Evaluation of Polynomials

In the oblivious polynomial evaluation problem there is a sender who has a polynomial P of degree k over some finite field $\mathcal{F}$ and a receiver with an element $x \in \mathcal{F}$. The receiver obtains $P(x)$ without learning anything else about the polynomial P and the sender learns nothing about x. This primitive was introduced in [16]. For our solution we use a new protocol [7] that requires $O(k)$ exponentiations in order to evaluate a polynomial of degree k (where the 'O' coefficient is very small). This is important as we work with low-degree polynomials. Following are the basic ideas of this protocol.

[3] Note that although our implementation approximates many invocations of the ln function, none of these approximations is revealed. The only approximation which becomes known to the parties is the final result of $\mathrm{ID3}_\delta$.

Let $P(y) = \sum_{i=0}^{k} a_i y^i$ and x be the sender and receiver's respective inputs. The following protocol enables the receiver to compute $g^{P(x)}$, where g is a generator of a group in which the Decisional Diffie-Hellman assumption holds. The protocol is very simple since it is assumed that the parties are semi-honest. It can be converted to one which computes $P(x)$ using the the methods of Paillier [18], who presented a trapdoor for computing discrete logs. Security against malicious parties can be obtained using proofs of knowledge. The protocol consists of the following steps:

- The receiver chooses a secret key s, and sends g^s to the sender.
- For $0 \leq i \leq k$, the receiver computes $c_i = (g^{r_i}, g^{s \cdot r_i} g^{x^i})$, where r_i is random. The receiver sends $c_0, \ldots, c_k$ to the sender.
- The sender computes $C = \Pi_{i=0}^{k} (c_i)^{a_i} = (g^R, g^{sR} g^{P(x)})$, where $R = \sum_{i=0}^{k} r_i a_i$. It then chooses a random value r and computes $C' = (g^R \cdot g^r, g^{sR} g^{P(x)} \cdot g^{sr})$ and sends it to the receiver.
- The receiver divides the second element of C' by the first element of C' raised to the power of s, and obtains $g^{P(x)}$.

By the DDH assumption, the sender learns nothing of x^i from the messages $c_0, \ldots, c_k$ sent by the receiver to the sender. On the other hand, the receiver learns nothing of P from C'.

4.3 Oblivious Circuit Evaluation

The two party protocol of Yao [21] solves the following problem. There are two parties, a party A which has an input x, and a party B which has as input a function f and a combinatorial circuit that computes f. At the end of the protocol A outputs $f(x)$ and learns no other information about f, while B learns nothing at all. We employ this protocol for the case that f depends on two inputs x and y, belonging to A and B respectively. This is accomplished by having B simply hardwire his input y into the circuit (that is, B's input is a function $f(\cdot, y)$ and A obtains $f(x, y)$ from the circuit).[4]

The overhead of the protocol involves (1) B sending to A tables of size linear in the size of the circuit, (2) A and B engaging in an oblivious transfer protocol for every input wire of the circuit, and (3) A computing a pseudo-random function a constant number of times for every gate. Therefore, the number of rounds of this protocol is constant (namely, two rounds using non-interactive oblivious transfer), and the main computational overhead is that of running the oblivious transfers.

Computing Random Shares. Note that by defining $r_1 = F(x, (y, r_2)) \stackrel{\text{def}}{=} f(x, y) - r_2$, A and B obtain random shares summing to $f(x, y)$.

5 The Protocol

The central idea of our protocol is that all intermediate values of the computation seen by the players are uniformly distributed. At each stage, the players obtain

[4] In the case that f is known and the parties are semi-honest, Yao's evaluation constitutes a secure protocol for the described problem.

random shares v_1 and v_2 such that their sum equals an appropriate intermediate value. Efficiency is achieved by having the parties do most of the computation independently.

We assume that there is a known upper bound on the size of the union of the databases, and that the attribute-value names are public.[5]

Solution Outline. The "most difficult" step in privately computing $\mathrm{ID3}_\delta$ reduces to oblivious evaluation of the $x \ln x$ function (Section 5.1). (A private protocol for this task is presented separately in Section 6.) Next, we show how given a protocol for computing $x \ln x$, we can privately find the next attribute in the decision tree (Section 5.2). Finally, we describe how the other steps of the $\mathrm{ID3}_\delta$ algorithm are privately computed and show the complete private protocol for computing $\mathrm{ID3}_\delta$ (Section 5.3).

5.1 A Closer Look at $\mathrm{ID3}_\delta$

The part of $\mathrm{ID3}_\delta$ which is hardest to implement in a private manner is step 3(a). In this step the two parties must find the attribute A that best classifies the transactions T in the database, namely the attribute that provides the maximum information gain. This step can be stated as: *Find the attribute A which minimizes the conditional information of T given A, $H_C(T|A)$.* Examine $H_C(T|A)$ for an attribute A with m possible values $a_1, \ldots, a_m$, and a class attribute C with l possible values $c_1, \ldots, c_\ell$.

$$\begin{aligned} H_C(T|A) &= \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} H_C(T(a_j)) \\ &= \frac{1}{|T|} \sum_{j=1}^{m} |T(a_j)| \sum_{i=1}^{\ell} -\frac{|T(a_j, c_i)|}{|T(a_j)|} \cdot \log\left(\frac{|T(a_j, c_i)|}{|T(a_j)|}\right) \\ &= \frac{1}{|T|} \left(-\sum_{j=1}^{m} \sum_{i=1}^{\ell} |T(a_j, c_i)| \log(|T(a_j, c_i)|) + \sum_{j=1}^{m} |T(a_j)| \log(|T(a_j)|) \right) \end{aligned} \tag{3}$$

Note that since the algorithm is only interested in finding the attribute A which minimizes $H_C(T|A)$, the coefficient $1/|T|$ can be ignored. Also, natural logarithms can be used instead of logarithms to the base 2.

The database is a union of two databases, D_1 which is known to P_1 and D_2 which is known to P_2. The number of transactions for which attribute A has value a_j can therefore be written as $|T(a_j)| = |T_1(a_j)| + |T_2(a_j)|$, where $|T_b(a_j)|$ is the number of transactions with attribute A set to a_j in database D_b (likewise $T(a_j, c_i)$ is the number of transaction with $A = a_j$ and the class attribute set to c_i). The values $|T_1(a_j)|$ and $|T_1(a_j, c_i)|$ can be computed by party P_1 independently, and the same holds for P_2. Therefore the expressions that should be compared can be written as a sum of expressions of the form

$$(v_1 + v_2) \cdot \ln(v_1 + v_2),$$

[5] It is clear that the databases must have the same structure with previously agreed upon attribute names.

where v_1 is known to P_1 and v_2 is known to P_2. The main task is, therefore, to privately compute $x \ln x$ and a protocol for this task is described in Section 6. The exact definition of this protocol is provided in Figure 2.

5.2 Finding the Attribute with Maximum Gain

Given the above protocol for privately computing shares of $x \ln x$, the attribute with the maximum information gain can be determined. This is done in two stages: first, the parties obtain shares of $H_C(T|A) \cdot |T| \cdot \ln 2$ for all attributes A and second, the shares are input into a very small circuit which outputs the appropriate attribute. In this section we refer to a field $\mathcal{F}$ which is defined so that $|\mathcal{F}| > H_C(T|A) \cdot |T| \cdot \ln 2$.

Stage 1 (computing shares) : For every attribute A, for every attribute-value $a_j \in A$ and every class $c_i \in C$, P_1 and P_2 use the $x \ln x$ protocol in order to obtain $w_{A,1}(a_j)$, $w_{A,2}(a_j)$, $w_{A,1}(a_j, c_i)$ and $w_{A,2}(a_j, c_i) \in_R \mathcal{F}$ such that

$$w_{A,1}(a_j) + w_{A,2}(a_j) = |T(a_j)| \cdot \log(|T(a_j)|) \mod |\mathcal{F}|$$
$$w_{A,1}(a_j, c_i) + w_{A,2}(a_j, c_i) = |T(a_j, c_i)| \cdot \log(|T(a_j, c_i)|) \mod |\mathcal{F}|$$

Now, define $\hat{H}_C(T|A) \stackrel{\text{def}}{=} H_C(T|A) \cdot |T| \cdot \ln 2$. Then,

$$\hat{H}_C(T|A) = -\sum_{j=1}^{m} \sum_{i=1}^{\ell} |T(a_j, c_i)| \cdot \ln(|T(a_j, c_i)|) + \sum_{j=1}^{m} |T(a_j)| \cdot \ln(|T(a_j)|)$$

Then, P_1 (and likewise P_2) computes his share in $\hat{H}_C(T|A)$ as follows:

$$S_{A,1} = -\sum_{j=1}^{m} \sum_{i=1}^{\ell} w_{A,1}(a_j, c_i) + \sum_{j=1}^{m} w_{A,1}(a_j) \mod |\mathcal{F}|$$

It is clear that $S_{A,1} + S_{A,2} = \hat{H}_C(T|A) \mod |\mathcal{F}|$ and we therefore have that for every attribute A, P_1 and P_2 obtain shares in $\hat{H}_C(T|A)$ (this last step involves local computation only).

Stage 2 (finding the attribute): It remains to find the attribute with the minimum $\hat{H}_C(T|A)$ (and therefore the minimum $H_C(T|A)$). This is done via a Yao circuit evaluation [21]. We note that since $\hat{H}_C(T|A) < |\mathcal{F}|$, it holds that either $S_{A,1} + S_{A,2} = \hat{H}_C(T|A)$ or $S_{A,1} + S_{A,2} = \hat{H}_C(T|A) + |\mathcal{F}|$.

The parties run an oblivious evaluation of a circuit with the following functionality. The circuit input is the shares of both parties for each $\hat{H}_C(T|A)$. The circuit first computes each $\hat{H}_C(T|A)$ (by subtracting $|\mathcal{F}|$ if the sum is larger than $|\mathcal{F}| - 1$ or leaving it otherwise), and then compares the results to find the smallest among them. This circuit has $2|R|$ inputs of size $\log |\mathcal{F}|$ and its size is $O(|R| \log |\mathcal{F}|)$. Note that $|R| \log |\mathcal{F}|$ is a small number and thus this circuit evaluation is efficient.

Privacy: Stage 1 is clearly private as it involves many invocations of a private protocol that outputs random shares, followed by a local computation. Stage 2 is also private as it involves a single invocation of Yao's oblivious circuit evaluation and nothing more.

Note the efficiency achieved above. Each party has to compute the same set of values $|T(a_j, c_i)|$ as it computes in an individual computation of ID3. For each of these values it engages in the $x \ln x$ protocol. (We stress that the number of values here does not depend on the number of transactions, but rather on the number of different possible values for each attribute, which is usually smaller by orders of magnitude.) It sums the results of all these protocols together, and engages in an oblivious evaluation of a circuit whose size is linear in the number of attributes.

5.3 The Private ID3$_\delta$ Protocol

In the previous subsection we showed how each node can be privately computed. The complete protocol for privately computing ID3$_\delta$ can be seen below. The steps of the protocol correspond to those in the original algorithm (see Figure 1).

Protocol 1 (Protocol for Private Computation of ID3$_\delta$:)

Step 1: *If R is empty, return a leaf-node with the class value of the majority of the transactions in T.*
Since the set of attributes is known to both parties, they both publicly know if R is empty. If yes, the parties do an oblivious evaluation of a circuit whose inputs are the values $\langle |T_1(c_1)|, \ldots, |T_1(c_\ell)| \rangle$ and $\langle |T_2(c_1)|, \ldots, |T_2(c_\ell)| \rangle$, and whose output is i such $|T_1(c_i)| + |T_2(c_i)|$ is maximal. The size of this circuit is linear in ℓ and in $\log(|T|)$.

Step 2: *If T consists of transactions with all the same class c, return a leaf-node with the value c.*
In order to compute this step privately, we must determine whether both parties remain with the same single class or not. We define a fixed symbol $\perp$ symbolizing the fact that a party has more than one remaining class. A party's input to this step is then $\perp$, or c_i if it is its one remaining class. All that remains to do is check *equality* of the two inputs. The value causing the equality can then be publicly announced as c_i (halting the tree on this path) or $\perp$ (to continue growing the tree from the current point).
The equality check can be executed in one of two ways: (1) Using the "comparing information without leaking it" protocols of Fagin, Naor, and Winkler [9]. This solution requires the execution of $\log(\ell + 1)$ oblivious transfers. (2) Using a protocol suggested in [16] and which involves the oblivious evaluation of linear polynomials. The overhead of this solution is $O(1)$ oblivious transfers, using the oblivious polynomial evaluation protocol of [7].

Step 3: *(a) Determine the attribute that best classifies the transactions in T, let it be A.*
For every value a_j of every attribute A, and for every value c_i of the class attribute C, the two parties run the $x \ln x$ protocol of Section 6 for $T(a_j)$ and $T(a_j, c_i)$. They then continue as described in Section 5.2 by computing

independent additions and inputting the results into a small circuit. Finally, they perform an oblivious evaluation of the circuit with the result being the attribute with the highest information gain, A. This is public knowledge as it becomes part of the output.

(b,c) Recursively call $\mathrm{ID3}_\delta$ *for the remaining attributes on the transaction sets* $T(a_1), \ldots, T(a_m)$ *(where* $a_1, \ldots, a_m$ *are the values of attribute* A*).*
The result of 3(a) and the attribute values of A are public and therefore both parties can individually partition the database and prepare their input for the recursive calls.

Although each individual step of the above protocol has been shown to be private, we must show that the composition is also private. The central issue in the proof involves showing that the control flow can be predicted from the input and output only.

Theorem 2 *The protocol for computing* $\mathrm{ID3}_\delta$ *is* private.

Proof. In this proof the simulator is described in generic terms as it is identical for P_1 and P_2. Furthermore, we skip details which are obvious Recall that the simulator is given the output decision tree.

We need to show that any information learned by the computation can be learned directly from the input and output. This is done by showing how the views can be correctly simulated based solely on the input and output. The computation of the tree is recursive beginning at the root. For each node, a "splitting" class is chosen (due to it having the highest information gain) developing the tree to the next level. Any implementation defines the order of developing the tree and this order is used by the simulator to write the messages received in the correct order. Therefore according to this order, at any given step the computation is based on finding the highest information gain for a *known* node (for the proof we ignore optimizations which find the gain for more than one node in parallel). We differentiate between two cases: (1) a given node is a leaf node and (2) a given node is not a leaf.

1. The Current Node in the Computation is a Leaf-Node: The simulator checks, by looking at the input, if the set of attributes R at this point is empty or not. If it is *not* empty (this can be deduced from the tree and the attribute-list which is public), then the computation proceeds to Step (2). In this case, the simulator writes that the oracle-answer from the equality call in Step (2) is equal (or else it would not be a leaf). On the other hand, if the list of attributes *is* empty, the computation is executed in Step (1) and the simulator writes the output of the majority evaluation to be the class appearing in the leaf.

2. The Current Node in the Computation is not a Leaf-Node: In this case Step (1) is skipped and the oracle-answer of Step (2) must be not-equal; this is therefore what the simulator writes. The computation then proceeds to Step (3) which involves many invocations of the $x \ln x$ protocol, returning values uniformly distributed in $\mathcal{F}$. Therefore, the simulator simply chooses the correct number of random values (based on the public list of attribute names, values and class values) and writes them. The next step of the algorithm is a local computation (not included in the view) and an oblivious circuit evaluation. The simulator simply looks to see which class is written in the tree at this node and writes the class name as the output from the circuit evaluation.

The computation then continues to the next node in the defined order of traversal. This completes the proof. ■

Remark. It is both surprising and interesting to note that if Steps (1) and (2) of the protocol are switched (as the algorithm is in fact presented in [15]), then it is no longer private. This is due to the equality evaluation in Step (2), which may leak information about the other party's input. Consider the case of a computation in which at a certain point the list of attributes is *empty* and P_1 has only one class c left in his remaining transactions. The output of the tree at this point is a leaf with a class, assume that the class is c. From the output it is impossible for P_1 to know if P_2's transactions also have only one remaining class or if the result is because the majority of both together is c. The majority circuit of Step (1) covers both cases and therefore does not reveal this information. However, if P_1 and P_2 first execute the equality evaluation, this information is revealed.

Complexity. A detailed analysis of the complexity of the protocol is presented in Appendix A. The overhead is dominated by the $x \ln x$ protocol.

6 A Protocol for Computing $x \ln x$

This section describes an efficient protocol for privately computing the $x \ln x$ function, as defined in Figure 2.

- **Input:** P_1's input is a value v_1; P_2's input is v_2.
- **Auxiliary input:** A large enough field $\mathcal{F}$, the size of which will be discussed later.
- **Output:** P_1 obtains $w_1 \in \mathcal{F}$ and P_2 obtains $w_2 \in \mathcal{F}$ such that:
  1. $w_1 + w_2 = (v_1 + v_2) \cdot \ln(v_1 + v_2) \bmod |\mathcal{F}|$
  2. w_1 and w_2 are uniformly distributed in $\mathcal{F}$ when viewed independently of one another.

Fig. 2. Definition of the $x \ln x$ protocol.

There are several difficulties in the design of such a protocol. Firstly, it is not clear how to obliviously compute the natural logarithm efficiently. Furthermore, the protocol must multiply two values together. An initial idea is to use Yao's generic two party circuit evaluation protocol [21] and construct a multiplication circuit. However, the size of this circuit is of the order of the *multiplication of the sizes of its inputs*. This subprotocol is to be repeated many times throughout the complete $\mathrm{ID3}_\delta$ protocol and its efficiency is, therefore, crucial.

The solution requires a linear size circuit and a small number of simple oblivious evaluation protocols. The problem is divided into two parts: First it is shown how to compute shares of $\ln x$ from shares of x. Secondly, we show how to obtain shares of the product $x \ln x$ given separate shares of x and $\ln x$.

6.1 Computing Shares of $\ln x$

We now show how to compute random shares u_1 and u_2 such that $u_1+u_2=\ln x$. The starting point for the solution is the Taylor series of the natural logarithm, namely:

$$\ln(1+\varepsilon)=\sum_{i=1}^{\infty}\frac{(-1)^{i-1}\varepsilon^i}{i}=\varepsilon-\frac{\varepsilon^2}{2}+\frac{\varepsilon^3}{3}-\frac{\varepsilon^4}{4}+\cdots \qquad \text{for } -1<\varepsilon<1$$

It is easy to verify that the error for a partial evaluation of the series is as follows:

$$\left|\ln(1+\varepsilon)-\sum_{i=1}^{k}\frac{(-1)^{i-1}\varepsilon^i}{i}\right|<\frac{|\varepsilon|^{k+1}}{k+1}\cdot\frac{1}{1-|\varepsilon|} \tag{4}$$

As is demonstrated in Section 6.3, the error shrinks exponentially as k grows.

Now, given an input x, let 2^n be the power of 2 which is closest to x (in the ID3$_\delta$ application, note that $n<\log|T|$). Therefore, $x=2^n(1+\varepsilon)$ where $-1/2\le\varepsilon\le 1/2$. Consequently,

$$\ln(x)=\ln(2^n(1+\varepsilon))=n\ln 2+\varepsilon-\frac{\varepsilon^2}{2}+\frac{\varepsilon^3}{3}-\frac{\varepsilon^4}{4}+\cdots$$

Our aim is to compute this Taylor series to the k'th place. Let N be a predetermined (public) upper-bound on the value of n ($N>n$ always). Now, we use a small circuit that receives v_1 and v_2 as input (the value of N is hardwired into it) and outputs shares of $2^N\cdot n\ln 2$ (for computing the first element in the series of $\ln x$) and $\varepsilon\cdot 2^N$ (for computing the remainder of the series). This circuit is easily constructed: notice that $\varepsilon\cdot 2^n=x-2^n$, where n can be determined by looking at the two most significant bits of x, and $\varepsilon\cdot 2^N$ is obtained simply by shifting the result by $N-n$ bits to the left. The possible values of $2^N n\ln 2$ are hardwired into the circuit. As we have described, random shares are obtained by having one of the parties input random values $\alpha_1,\beta_1\in_R\mathcal{F}$ into the circuit and having the circuit output $\alpha_2=\varepsilon\cdot 2^N-\alpha_1$ and $\beta_2=2^N\cdot n\ln 2-\beta_1$ to the other party. The parties therefore have shares α_1,β_1 and α_2,β_2 such that

$$\alpha_1+\alpha_2=\varepsilon 2^N \ \text{ and } \ \beta_1+\beta_2=2^N n\ln 2$$

The second stage of the protocol involves computing shares of the Taylor series approximation. In fact, it computes shares of

$$\mathrm{lcm}(2,...k)\cdot 2^N\left(n\ln 2+\varepsilon-\frac{\varepsilon^2}{2}+\frac{\varepsilon^3}{3}-\cdots\frac{\varepsilon^k}{k}\right)\approx\mathrm{lcm}(2,...k)2^N\ln x \tag{5}$$

(where $\mathrm{lcm}(2,...,k)$ is the lowest common multiple of $\{2,\ldots,k\}$, and we multiply by it to ensure that there are no fractions). In order to do this P_1 defines the following polynomial:

$$Q(x)=\mathrm{lcm}(2,\ldots,k)\cdot\sum_{i=1}^{k}\frac{(-1)^{i-1}}{2^{N(i-1)}}\frac{(\alpha_1+x)^i}{i}-w_1$$

where $w_1 \in_R \mathcal{F}$ is randomly chosen. It is easy to see that

$$w_2 \stackrel{\text{def}}{=} Q(\alpha_2) = \text{lcm}(2,...,k) \cdot 2^N \cdot \left(\sum_{i=1}^{k} \frac{(-1)^{i-1}\varepsilon^i}{i} \right) - w_1$$

Therefore by a single oblivious polynomial evaluation of the k-degree polynomial $Q(\cdot)$, P_1 and P_2 obtain random shares w_1 and w_2 to the approximation in Equation (5). Namely P_1 defines $u_1 = w_1 + \text{lcm}(2,\ldots,k)\beta_1$ and likewise P_2. We conclude that

$$u_1 + u_2 \approx \text{lcm}(2,\ldots,k)2^N \cdot \ln x$$

This equation is accurate up to an approximation error which we bound, and the shares are random as required. Since N and k are known to both parties, the additional multiplicative factor of $2^N \cdot \text{lcm}(2,\ldots,k)$ is public and can be removed at the very end. Notice that *all* the values in the computation are integers (except for $2^N n \ln 2$ which is given as the closest integer number).

The size of the field $\mathcal{F}$. It is necessary that the field be chosen large enough so that the initial inputs in each evaluation and the final output be between 0 and $|\mathcal{F}| - 1$. Notice that all computation is based on $\varepsilon 2^N$. This value is raised to powers up to k and multiplied by $\text{lcm}(2,\ldots,k)$. Therefore a field of size 2^{Nk+2k} is clearly large enough, and requires $(N+2)k$ bits for representation.

We now summarize the $\ln x$ protocol:

Protocol 2 (Protocol $\ln x$)

1. P_1 and P_2 input their shares v_1 and v_2 into an oblivious evaluation protocol for a circuit outputting: (1) Random shares α_1 and α_2 of $\varepsilon 2^N$ (i.e. $\alpha_1 + \alpha_2 = \varepsilon 2^N \bmod |\mathcal{F}|$). (2) Random shares β_1,β_2 such that $\beta_1 + \beta_2 = 2^N \cdot n \ln 2$.
2. P_1 chooses $w_1 \in_R \mathcal{F}$ and defines the following polynomial

$$Q(x) = \text{lcm}(2,\ldots,k) \cdot \sum_{i=1}^{k} \frac{(-1)^{i-1}}{2^{N(i-1)}} \frac{(\alpha_1 + x)^i}{i} - w_1$$

3. P_1 and P_2 then execute an oblivious polynomial evaluation with P_1 inputting $Q(\cdot)$ and P_2 inputting α_2, in which P_2 obtains $w_2 = Q(\alpha_2)$.
4. P_1 and P_2 define $u_1 = \text{lcm}(2,\ldots,k)\beta_1 + w_1$ and $u_2 = \text{lcm}(2,\ldots,k)\beta_2 + w_2$ respectively. We have that $u_1 + u_2 \approx 2^N \text{lcm}(2,\ldots,k) \cdot \ln x$

Proposition 3 *Protocol 2 constitutes a private protocol for computing random shares of $c \cdot \ln x$ in $\mathcal{F}$, where $c = 2^N \text{lcm}(2,\ldots,k)$.*

Proof. We first show that the protocol correctly computes shares of $c \ln x$. In order to do this, we must show that the computation over $\mathcal{F}$ results in a correct result over the reals. We first note that *all* the intermediate values are integers. In particular, $\varepsilon 2^n$ equals $x - 2^n$ and is therefore an integer as is $\varepsilon 2^N$ (since $N > n$). Furthermore, every division by i $(2 \leq i \leq k)$ is counteracted by a multiplication by $\text{lcm}(2,\ldots,k)$. The only exception is $2^N n \ln 2$. However, this is taken care of by having the original circuit output the closest integer to $2^N n \ln 2$ (although the rounding to the closest integer introduces an additional approximation error, it is negligible compared to the approximation error of the Taylor series).

Secondly, the field $\mathcal{F}$ is defined to be large enough so that all intermediate values (i.e. the sum of shares) and the final output (as a real number times $2^N \cdot \mathrm{lcm}(2, \ldots, k)$) are between 0 and $|\mathcal{F}| - 1$. Therefore the two shares uniquely identify the result, which equals the sum (over the integers) of the two random shares if it is less than $|\mathcal{F}|$, or the sum minus $|\mathcal{F}|$ otherwise.

The proof of privacy appears in the full version of the paper. ■

6.2 Computing Shares of $x \ln x$

We begin by briefly describing a simple multiplication protocol that on private inputs a_1 and a_2 outputs random shares b_1 and b_2 (in some finite field $\mathcal{F}$) such that $b_1 + b_2 = a_1 \cdot a_2$.

Protocol 3 (Protocol $Mult(a_1, a_2)$**)**

The protocol is very simple and is based on an oblivious evaluation of a linear polynomial. The protocol begins by P_1 choosing a random value $b_1 \in \mathcal{F}$ and defining a linear polynomial $Q(x) = a_1 x - b_1$. P_1 and P_2 then engage in an oblivious evaluation of Q, in which P_2 obtains $b_2 = Q(a_2) = a_1 \cdot a_2 - b_1$. We define the respective outputs of P_1 and P_2 as b_1 and b_2 giving us that $b_1 + b_2 = a_1 \cdot a_2$.

Proposition 4 *Protocol 3 constitutes a private protocol for computing* Mult *as defined above.*

We are now ready to present the complete $x \ln x$ protocol:

Protocol 4 (Protocol $x \ln x$**)**

1. P_1 and P_2 use Protocol 2 for privately computing shares of $\ln x$ in order to obtain random shares u_1 and u_2 such that $u_1 + u_2 = \ln x$.
2. P_1 and P_2 use two invocations of Protocol 3 in order to obtain shares of $u_1 \cdot v_2$ and $u_2 \cdot v_1$.
3. P_1 (resp., P_2) then defines his output w_1 (resp., w_2) to be the sum of the two *Mult* shares and $u_1 \cdot v_1$ (resp., $u_2 \cdot v_2$).
4. We have that $w_1 + w_2 = u_1 v_1 + u_1 v_2 + u_2 v_1 + u_2 v_2 = (u_1 + u_2)(v_2 + v_2) = x \ln x$ as required.

Theorem 5 *Protocol 4 is a protocol for privately computing random shares of* $x \ln x$.

The correctness of the protocol is straightforward. The proof of the privacy properties appears in the full version of the paper.

Complexity The detailed analysis of the complexity is presented in Appendix A.

6.3 Choosing the Parameter k

Recall that the parameter k defines the accuracy of the Taylor approximation of the "ln" function. Given δ and the database, we analyze which k we need to take in order to ensure that the defined δ-approximation is correctly estimated[6] From here on we denote an approximation of the value z by $\widetilde{z}$.

The approximation definition of $\mathrm{ID3}_\delta$ requires that for all A_1, A_2

$$H_C(T|A_1) > H_C(T|A_2) + \delta \Rightarrow \widetilde{H_C}(T|A_1) > \widetilde{H_C}(T|A_2)$$

This is clearly fulfilled if $\left|H_C(T|A_b) - \widetilde{H_C}(T|A_b)\right| < \frac{\delta}{2}$ for $b = 1, 2$.

We now bound the difference on each $|\ln x - \widetilde{\ln x}|$ in order that the above condition is fulfilled. By replacing $\log x$ by $\frac{1}{\ln 2}|\ln x - \widetilde{\ln x}|$ in Equation (3) computing $H_C(T|A)$, we obtain a bound on the error of $\left|H_C(T|A_1) - \widetilde{H_C}(T|A_1)\right|$. A straightforward algebraic manipulation gives us that if $\frac{1}{\ln 2}|\ln x - \widetilde{\ln x}| < \frac{\delta}{4}$, then the error is less than $\frac{\delta}{2}$ as required. As we have mentioned (Equation (4)), the $\ln x$ error is bounded by $\frac{|\varepsilon|^{k+1}}{k+1}\frac{1}{1-|\varepsilon|}$ and this is maximum at $|\varepsilon| = \frac{1}{2}$ (recall that $-\frac{1}{2} \leq \varepsilon \leq \frac{1}{2}$). Therefore, given δ, we set $\frac{1}{2^k k+1} < \frac{\delta}{4} \cdot \ln 2$ or $k + \log(k+1) > \log\left[\frac{4}{\delta \ln 2}\right]$ (for $\delta = 0.0001$, it is enough to take $k > 12$). Notice that the value of k is *not* dependent on the input database.

References

1. M. Bellare and S. Micali, *Non-interactive oblivious transfer and applications*, Advances in Cryptology - Crypto '89, pp. 547-557, 1990.
2. M. Ben-Or, S. Goldwasser and A. Wigderson, *Completeness theorems for non cryptographic fault tolerant distributed computation*, 20th STOC, (1988), 1-9.
3. D. Boneh and M. Franklin, *Efficient generation of shared RSA keys*, Proc. of Crypto' 97, LNCS, Vol. 1233, Springer-Verlag, pp. 425–439, 1997.
4. R. Canetti, *Security and Composition of Multi-party Cryptographic Protocols.* To appear in the Journal of Cryptology. Available from the Theory of Cryptography Library at http://philby.ucsd.edu/cryptlib, 1998.
5. D. Chaum, C. Crepeau and I. Damgard, *Multiparty unconditionally secure protocols*, 20th Proc. ACM Symp. on Theory of Computing, (1988), 11-19.
6. B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, *Private Information Retrieval*, 36th FOCS, pp. 41–50, 1995.
7. R. Cramer, N. Gilboa. M. Naor, B. Pinkas and G. Poupard, *Oblivious Polynomial Evaluation*, 2000.
8. S. Even, O. Goldreich and A. Lempel, *A Randomized Protocol for Signing Contracts*, Communications of the ACM **28**, pp. 637–647, 1985.
9. R. Fagin, M. Naor and P. Winkler, *Comparing Information Without Leaking It*, Communications of the ACM, vol 39, May 1996, pp. 77-85.

[6] An additional error is introduced by rounding the value $2^N n \ln 2$ to the closest integer. We ignore this error as it is negligible compared to the approximation error of the ln function.

10. J. Feigenbaum, J. Fong, M. Strauss and R. N. Wright, *Secure Multiparty Computation of Approximations*, manuscript, 2000.
11. N. Gilboa, *Two Party RSA Key Generation*, Proc of Crypto '99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 116–129, 1999.
12. O. Goldreich, *Secure Multi-Party Computation*, 1998. (Available at http://philby.ucsd.edu)
13. O. Goldreich, S. Micali and A. Wigderson, *How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority.* In 19th ACM Symposium on the Theory of Computing, pp. 218-229, 1987.
14. J. Kilian, **Uses of randomness in algorithms and protocols**, MIT Press, 1990.
15. T. Mitchell, **Machine Learning.** McGraw Hill, 1997.
16. M. Naor and B. Pinkas, *Oblivious Transfer and Polynomial Evaluation*, Proc. of the 31st STOC, Atlanta, GA, pp. 245-254, May 1-4, 1999.
17. M. Naor and B. Pinkas, *Efficient Oblivious Transfer Protocols*, manuscript, 2000.
18. P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuocity Classes.* Proc. of Eurocrypt '99, LNCS Vol. 1592, pp. 223–238, 1999.
19. J. Ross Quinlan, *Induction of Decision Trees.* Machine Learning 1(1): 81-106(1986)
20. M. O. Rabin, *How to exchange secrets by oblivious transfer*, Tech. Memo TR-81, Aiken Computation Laboratory, 1981.
21. A.C. Yao, *How to generate and exchange secrets*, Proc. of the 27th IEEE Symp. on Foundations of Computer Science, 1986, pp. 162–167.

A Complexity

The communication complexity is measured by two parameters: the number of rounds and the bandwidth of all messages sent. As for the computation overhead, it is measured by the number of exponentiations and oblivious transfers (ignoring evaluations of pseudo-random functions, since they are more efficient by a few orders of magnitude).

Parameters: The overhead depends on the following parameters:

- T, the number of transactions.
- k, the length of the Taylor series, which affects the accuracy.
- $\mathcal{F}$, the field over which the computation is done. This is set as a function of the above two parameters, namely $\log|\mathcal{F}| = (k+2)\log|T|$
- $|R|$, the number of attributes.
- m, the number of possible values for each attribute (to simplify the notation assume that this is equal for all attributes).
- ℓ, the number of possible values for the class attribute.
- $|E|$, the length of an element in the group in which oblivious transfers and exponentiations are implemented. To simplify the notation we assume that $|E| > \log|\mathcal{F}| = k\log|T|$.
- $|S|$, the length of a key for a pseudorandom function used in the circuit evaluation (say, 80 or 100 bits long).
- $|D|$, the number of nodes in the decision tree.

A very detailed analysis of the complexity is given in the full version of the paper. The $\ln x$ protocol (Protocol 2) affects the complexity the most. Its overall overhead is $O(\max(\log|T|, k))$ oblivious transfers. Since $|T|$ is usually large (e.g. $\log|T| = 20$), and on the other hand k can be set to small values (e.g. $k = 12$), the overhead can be defined as $O(\log |T|)$ oblivious transfers. The main communication overhead is incurred by the circuit evaluation and is $O(k \log |T| \cdot |S|)$ bits.

Finding the best attribute for a node. This step requires running the $\ln x$ protocol for every attribute and for every combination of attribute-value and class-value, and evaluating a small circuit. The communication overhead is $O(|R| m\ell k \log |T| \cdot |S|)$ bits and the computation overhead is $O(|R| m\ell \log |T|)$ oblivious transfers. The number of rounds is $O(1)$.

Computing all nodes of the decision tree. All nodes on the same level of the tree can be computed in parallel. We therefore have that the number of rounds equals $O(d)$ where d is the depth of the tree. The value of d is upper bound by $|R|$ but is expected to be much smaller.
Overall complexity:

- **Parameters:** For a concrete example, assume that there are a million transactions $|T| = 2^{20}$, $|R| = 15$ attributes, each attribute has $m = 10$ possible values, the class attribute has $\ell = 4$ values, and $k = 10$ suffices to have the desired accuracy. Say that the depth of the tree is $d = 7$, and that it uses private keys of length $|S| = 80$ bits.
- **Rounds:** There are $O(d)$ rounds.
- **Communication:** The communication overhead is $O(|D| \cdot |R| m\ell k \log |T| \cdot |S|)$. In our example, this is $|D| \cdot 15 \cdot 10 \cdot 4 \cdot 10 \cdot 20 \cdot 80 = 9,600,000|D|$ bits times a very small constant factor. We conclude that the communication per node can be transmitted in a matter of seconds using a fast communication network (e.g. a T1 line with 1.5Mbps bandwidth, or a T3 line with 35Mbps).
- **Computation:** The computation overhead is $O(|D| \cdot |R| m\ell \log |T|)$. In our example, this is an order of $|D| \cdot 15 \cdot 10 \cdot 4 \cdot 20 = 12,000|D|$ exponentiations and oblivious transfers. Assuming that a modern PC can compute 50 exponentiations per second, we conclude that the computation per node can be completed in a matter of minutes.

In the full paper we present a comparison to generic solutions that shows that our protocol achieves a considerable improvement (both in comparison to the complete ID3 protocol and to the $x \ln x$ protocol).