Privacy-Preserving Decision Trees over Vertically Partitioned Data

JAIDEEP VAIDYA
Rutgers University
CHRIS CLIFTON
Purdue University
MURAT KANTARCIOGLU
University of Texas at Dallas
and
A. SCOTT PATTERSON
Johns Hopkins University

Privacy and security concerns can prevent sharing of data, derailing data-mining projects. Distributed knowledge discovery, if done correctly, can alleviate this problem. We introduce a generalized privacy-preserving variant of the ID3 algorithm for vertically partitioned data distributed over two or more parties. Along with a proof of security, we discuss what would be necessary to make the protocols completely secure. We also provide experimental results, giving a first demonstration of the practical complexity of secure multiparty computation-based data mining.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms: Security

Additional Key Words and Phrases: Decision tree classification, privacy

A preliminary version of this article appeared in the *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2008, pp. 139–152.
This research was partially supported by the National Science Foundation under Grants No. CNS-0746943 and IIS-0428168, by MURI award FA9550-08-1-0265 and award FA9550-07-1-0041 from the Air Force Office of Scientific Research, and by a Research Resources Grant from Rutgers Business School.
Authors' addresses: J. Vaidya, Rutgers University, 180 University Ave., Newark, NJ 07102; email: jsvaidya@business.rutgers.edu; C. Clifton, Purdue University, 305 N. University St., West Lafayette, IN 47907; email: clifton@cs.purdue.edu; M. Kantarcioglu, University of Texas at Dallas, 800 W. Campbell Rd, Richardson, TX 75080; email: muratk@utdallas.edu; A. S. Patterson, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218; email: scott.patterson@jhu.edu.

ACM Reference Format:
Vaidya, J., Clifton, C., Kantarcioglu, M., and Patterson, A. S. 2008. Privacy-preserving decision trees over vertically partitioned data. ACM Trans. Knowl. Discov. Data. 2, 3, Article 14 (October 2008), 27 pages. DOI = 10.1145/1409620.1409624 http://doi.acm.org/10.1145/1409620.1409624

1. INTRODUCTION

There has been growing interest in privacy-preserving data mining since the seminal papers in 2000 [Agrawal and Srikant 2000; Lindell and Pinkas 2000]. Classification is one of the most ubiquitous data-mining problems found in real life. Decision tree classification is one of the best-known solution approaches. ID3, first proposed by Quinlan [1986] is a particularly elegant and intuitive solution. This article presents an algorithm for privately building an ID3 decision tree. While this has been done for horizontally partitioned data [Lindell and Pinkas 2002], we present an algorithm for *vertically partitioned* data: A portion of each instance is present at each site, but no site contains complete information for any instance. This problem has been addressed [Wang et al. 2006; Du and Zhan 2002], but the solutions are limited to cases where both parties have the class attribute. (Further differences are discussed in Section 8.) The method presented here works for any number of parties and the class attribute (or other attributes) need be known only to one party. Our method is trivially extendible to the simplified case where all parties know the class attribute.

As an example of the need for such a capability, witness the growing interest in evidence-based medicine [Evidence-Based Medicine Working Group 1992]. While originally applied to manual literature searches in support of individual patient care, the concept that treatments must be based on the conditions of individual patients is of growing importance. For example, some cancer treatments are highly effective but have debilitating side effects, with high variance between populations [Shirao et al. 2004]; use for the majority of the population ruins quality of life for no appreciable benefit. The factors determining the efficacy of such treatments are complex and not fully known; learning a decision tree that could assist in the traditional literature search and treatment process would have significant benefit. Unfortunately, the factors may span many sources (medical practices, hospital records, pharmacy data, insurers), each of which is prevented by privacy laws from disclosing individually identifiable information. (The traditional approach, obtaining patient consent to participate in a clinical trial, is difficult to scale to such a situation. While we do not see this as supplanting the traditional clinical trial to evaluate a treatment, the number of participants required to compare multiple patients across multiple treatments on a wide variety of factors may not be feasible.) The method presented here enables use of historical data, without the disclosure that would run afoul of privacy laws.

Note that this is just one example; many others can be envisioned. Applications run beyond personal privacy. For example, corporations could collaborate across the supply chain while protecting proprietary information [Atallah et al. 2003].

We do not claim that an ID3 decision tree is the right solution to all such problems, but this article also makes a key contribution that goes beyond the specific technique implemented: a demonstration that privacy-preserving distributed data mining is *real* and *usable*. The algorithm has been implemented on top of Weka [Witten and Frank 1999], and we present experimental results on sample datasets, showing the times required to build a decision tree and classify instances. While our protocols leak a small amount of extra information (that does not violate individual privacy), we also discuss what would be required to make the protocols completely secure, along with the added computation and communication complexity.

Privacy preservation can mean many things: protecting specific individual values, breaking the link between values and the individual they apply to, protecting sources, etc. This article aims for a high standard of privacy: Not only individual entities are protected, but to the extent feasible even the schema (attributes and possible attribute values) are protected from disclosure. The goal is that each site disclose as little as possible, while still constructing a valid tree in a time suitable for practical application.

To this end, all that is revealed is the basic structure of the tree (e.g., the number of branches at each node, corresponding to the number of distinct values for an attribute; the depth of each subtree) and which site is responsible for the decision made at each node (i.e., which site possesses the attribute used to make the decision, but not what attribute is used, or even what attributes the site possesses). This allows for efficient *use* of the tree to classify an object; otherwise using the tree would require a complex cryptographic protocol involving every party at every *possible* level to evaluate the class of an object without revealing who holds the attribute used at that level. Each site also learns the count of classes at some interior nodes (although only the class site knows the mapping to actual classes; other sites don't even know if a class with 30% distribution at one node is the same class as one with a 60% distribution at a lower node, except to the extent that this can be deduced from the tree and a site's own attributes). At the leaf nodes, this is desirable: We often want probability estimates, not simply a predicted class. The class site learns transaction counts at leaf nodes, enabling it to compute distributions throughout the tree, so this doesn't disclose much *new* information. Section 7 gives a (somewhat costlier) protocol that eliminates these leaks; since one of the primary contributions of this article is to show a practical, implemented secure multiparty computation approach to privacy-preserving data mining, we start with the version with somewhat greater disclosure.

We now go directly into the algorithm for creating a tree. In Section 3 we describe how the tree (distributed between sites) is used to classify an instance, even though the attribute values of the instance to be classified are also private and distributed between sites. Section 4 formalizes what it means to be secure, and gives a proof that the algorithms presented are secure. We then discuss the computational cost, both analytically (Section 5) and based on actual runtimes

Algorithm 1. ID3(R,C,T) tree learning algorithm

```
Require: $R$, the set of attributes
Require: $C$, the class attribute
Require: $T$, the set of transactions
1: if $R$ is empty then
2:     return a leaf node, with class value assigned to most transactions in $T$
3: else if all transactions in $T$ have the same class $c$ then
4:     return a leaf node with the class $c$
5: else
6:     Determine the attribute $A$ that best classifies the transactions in $T$
7:     Let $a_1, \ldots, a_m$ be the values of attribute $A$. Partition $T$ into the $m$ partitions
       $T(a_1), \ldots, T(a_m)$ such that every transaction in $T(a_i)$ has the attribute value
       $a_i$.
8:     Return a tree whose root is labeled $A$ (this is the test attribute) and has $m$
       edges labeled $a_1, \ldots, a_m$ such that for every $i$, the edge $a_i$ goes to the tree
       $ID3(R - A, C, T(a_i))$.
9: end if
```

Algorithm 2. IsREmpty(): Are any attributes left?

```
Require: $k$ sites $P_i$ (the site calling the function is $P_1$; any other site can be $P_k$),
    each with a flag $AR_i = 0$ if no remaining attributes, $AR_i = 1$ if $P_i$ has attributes
    remaining.
Require: a commutative encryption function $E$ with domain size $m > k$.
1: $P_1$ chooses a random integer $r$ uniformly from $0 \ldots m-1$.
2: $P_1$ sends $r + AR_1$ to $P_2$
3: for $i = 2..k-1$ do
4:     Site $P_i$ receives $r'$ from $P_{i-1}$.
5:     $P_i$ sends $r' + AR_i \bmod m$ to $P_{i+1}$
6: end for
7: Site $P_k$ receives $r'$ from $P_{k-1}$.
8: $r' \leftarrow r' + AR_k \bmod m$
9: $P_1$ and $P_k$ create secure keyed commutative hash keys $E_1$ and $E_k$
10: $P_1$ sends $E_1(r)$ to $P_k$
11: $P_k$ receives $E_1(r)$ and sends $E_k(E_1(r))$ and $E_k(r')$ to $P_1$
12: $P_1$ returns $E_1(E_k(r')) = E_k(E_1(r))$ $\{\Leftrightarrow r' = r \Leftrightarrow \sum_{j=1}^{k} AR_i = 0 \Leftrightarrow 0$ attributes
    remain $\}$
```

(Section 6). Section 7 discusses what would be required to make the algorithms completely secure. Background on other work in privacy-preserving data-mining work is given in Section 8, after which we summarize and conclude the article.

2. PRIVACY-PRESERVING ID3: CREATING THE TREE

The basic ID3 algorithm [Quinlan 1986] is given in Algorithm 1. We will introduce our distributed privacy-preserving version by running through this algorithm, describing pieces as appropriate. We then give the full algorithm in Algorithm 7. Note that for our distributed algorithm, no site knows R; instead,

A_1	A_2	A_3	A_4	A_5	A_6
5	high	?	?	warm	?

Fig. 1. A constraint tuple for a single site.

each site i knows its own attributes R_i. Only one site knows the class attribute C. In vertical partitioning, every site knows a *projection* of the transactions $\Pi_{R_i} T$. Each projection includes a transaction identifier that serves as a join key.

We first check whether R is empty. This is based on Secure Sum [Schneier 1995; Kantarcıoğlu and Clifton 2004], and is given in Algorithm 2. Basically, the first party adds a random r to its count of remaining items. This is passed to all sites, each adding its count. The last site and first then use commutative encryption to compare the final value to r (without revealing either): If they are the same, R is empty.

Line 2 requires determining the majority class for a node, when only one site knows the class. This is accomplished with a protocol for securely determining the cardinality of set intersection. Many protocols for doing so are known [Vaidya and Clifton 2005b; Freedman et al. 2004; Agrawal et al. 2003].

We describe one such protocol in Section 2.1. Each site determines which of its transactions *might* reach that node of the tree. The intersection of these sets with the transactions in a particular class gives the number of transactions that reach this point in the tree, enabling the class site to determine the distribution and majority class; it returns a (leaf) node identifier that allows it to map back to this distribution.

To formalize this, we introduce the notion of a *constraint set*. As the tree is being built, each party i keeps track of the values of its attributes used to reach that point in the tree in a filter $Constraints_i$. Initially, these are all don't-care values ("?"). However, when an attribute A_{ij} at site i is used (lines 6 and 7 of id3), entry j in $Constraints_i$ is set to the appropriate value before recursing to build the subtree. An example is given in Figure 1; we use the well-known (although admittedly not privacy-sensitive) "weather/tennis" example to demonstrate the way the algorithm works. The site has 6 attributes $A_1, \ldots, A_6$. The constraint tuple shows that the only transactions valid for this transaction are those with a value of 5 for A_1, *high* for A_2, and *warm* for A_5. The other attributes have a value of ?, since they do not factor into the selection of an instance. Formally, we define the following functions.

—*Constraints.set(attr, val)*. Set the value of attribute *attr* to *val* in the local constraints set. The special value "?" signifies a don't-care condition.

—*Satisfies*. *x satisfies* $Constraints_i$ if and only if the attribute values of the instance are compatible with the constraint tuple: $\forall i, (A_i(x) = v \Leftrightarrow Constraints(A_i) = v) \vee Constraints(A_i) = \text{"?"}$.

Algorithm 3. DistributionCounts(): Compute class distribution given current constraints

```
Require: $k$ sites $P_i$ with local constraint sets $Constraints_i$
1: for all sites $P_i$ except $P_k$ do
2:    at $P_i$: $Y_i \leftarrow FormTransSet(Constraints_i)$
3: end for
4: for each class $c_1, \dots, c_p$ do
5:    at $P_k$: $Constraints_k.set(C, c_i)$ {To include the class restriction}
6:    at $P_k$: $Y_k \leftarrow FormTransSet(Constraints_k)$
7:    $cnt_i \leftarrow |Y_1 \cap \cdots \cap Y_k|$ using the cardinality of set intersection protocol ([Vaidya
      and Clifton 2005b; Freedman et al. 2004; Agrawal et al. 2003])
8: end for
9: return $(cnt_1, \dots, cnt_p)$
```

—*FormTransSet. Function FormTransSet(Constraints): Return local transactions meeting constraints:*

```
1: $Y = \emptyset$
2: for all transaction id $i \in T$ do
3:    if $t_i$ satisfies $Constraints$ then
4:       $Y \leftarrow Y \cup \{i\}$
5:    end if
6: end for
7: return $Y$.
```

Now, we determine the majority class (and class distributions) by computing for each class $\bigcap_{i=1..k} Y_i$, where Y_k includes a constraint on the class value. This is given in Algorithm 3.

The next issue is determining whether all transactions have the same class (Algorithm 1, line 3). If all are not of the same class, as little information as possible should be disclosed. For efficiency, we do allow the class site to learn the count of classes, even if this is an interior node; since it could compute this from the counts at the leaves of the subtree below the node, this discloses no additional information. Algorithm 4 gives the details: It uses constraint sets and secure cardinality of set intersection in basically the manner described earlier for computing the majority class at a leaf node. If all transactions are in the same class, we construct a leaf node. The class site maintains a mapping from the ID of that node to the resulting class distribution.

Algorithm 4. IsSameClass(): Are all transactions of the same class?

```
Require: $k$ sites $P_i$ with local constraint sets $Constraints_i$
1: $(cnt_1, \dots, cnt_p) \leftarrow DistributionCounts()$
2: if $\exists j$ s.t. $cnt_j \neq 0 \wedge \forall i \neq j, cnt_i = 0$ {only one of the counts is non-zero} then
3:    Build a leaf node with distribution $(cnt_1, \dots, cnt_p)$ {Actually, 100% class $j$}
4:    return ID of the constructed node
5: else
6:    return $false$
7: end if
```

Algorithm 5. AttribMaxInfoGain(): return the site with the attribute having maximum information gain

```
1: for all sites $P_i$ do
2:   $bestgain_i \leftarrow -1$
3:   for each attribute $A_{ij}$ at site $P_i$ do
4:     $gain \leftarrow ComputeInfoGain(A_{ij})$
5:     if $gain > bestgain_i$ then
6:       $bestgain_i \leftarrow gain$
7:       $BestAtt_i \leftarrow A_{ij}$
8:     end if
9:   end for
10: end for
11: return $argmax_j\ bestgain_j$ {Could implement using a set of secure comparisons}
```

The next problem is to compute the best attribute: that with the maximum information gain. The information gain when an attribute A is used to partition the dataset S is

$$Gain(S, A) = Entropy(S) - \sum_{v \in A} \left(\frac{|S_v|}{|S|} * Entropy(S_v) \right).$$

The entropy of a dataset S is given by

$$Entropy(S) = -\sum_{j=1}^{p} \frac{N_j}{N} \log \frac{N_j}{N},$$

where N_j is the number of transactions having class c_j in S and N is the number of transactions in S. As can be seen, this again becomes a problem of counting transactions: the number of transactions that reach the node N, the number in each class N_j, and the same two after partitioning with each possible attribute value $v \in A$. Algorithm 6 details the process of computing these counts; Algorithm 5 captures the overall process.

Once the best attribute has been determined, execution proceeds at that site. It creates an interior node for the split, then recurses.

Algorithm 6. ComputeInfoGain(A): Compute the Information Gain for attribute A

```
1: $S \leftarrow DistributionCounts()$ {Total number of transactions at this node}
2: $InfoGain \leftarrow Entropy(S)$
3: for each attribute value $a_i$ do
4:   $Constraints.set(A, a_i)$ {Update local constraints tuple}
5:   $S_{a_i} \leftarrow DistributionCounts()$
6:   $Infogain \leftarrow Infogain - Entropy(S_{a_i}) * |S_{a_i}|/|S|$ {$|S|$ is $\sum_{i=1}^{p} cnt_i$}
7: end for
8: $Constraints.set(A, `?')$ {Update local constraints tuple}
9: return InfoGain
```

Algorithm 7. PPID3(): Privacy-Preserving Distributed ID3

```
Require: Transaction set $T$ partitioned between sites $P_1, \ldots, P_k$
Require: $p$ class values, $c_1, \ldots, c_p$, with $P_k$ holding the class attribute
1: if $IsREmpty()$ then
2:   Continue at site $P_k$ up to the return:
3:   $(cnt_1, \ldots, cnt_p) \leftarrow DistributionCounts()$
4:   Build a leaf node with distribution $(cnt_1, \ldots, cnt_p)$
5:   $\{class \leftarrow argmax_{i=1..p}\, cnt_i\}$
6:   return ID of the constructed node
7: else if $clsNode \leftarrow$ (at $P_k$ :) $IsSameClass()$ then
8:   return leaf nodeId $clsNode$
9: else
10:  $BestSite \leftarrow AttribMaxInfoGain()$
11:  Continue execution at $BestSite$:
12:  Create Interior Node $Nd$ with attribute $Nd.A \leftarrow BestAtt_{BestSite}$ {This is
     best locally (from $AttribMaxInfoGain()$), and globally from line 8}
13:  for each attribute value $a_i \in Nd.A$ do
14:    $Constraints.set(Nd.A, a_i)$ {Update local constraints tuple}
15:    $nodeId \leftarrow PPID3()$ {Recurse}
16:    $Nd.a_i \leftarrow nodeId$ {Add appropriate branch to interior node}
17:  end for
18:  $Constraints.set(A, \text{'?'})$ {Returning to parent: should no longer filter trans-
     actions with $A$}
19:  Store $Nd$ locally keyed by Node ID
20:  return Node ID of interior node $Nd$ {Execution continues at site owning
     parent node}
21: end if
```

2.1 Cardinality of Set Intersection

Since the cardinality of set intersection protocol is required throughout the process, for the sake of completeness, we now describe a solution for this problem. In fact, there are several solutions to securely finding the size of intersection of sets [Vaidya and Clifton 2005b; Agrawal et al. 2003; Freedman et al. 2004]. We sketch the method of Vaidya and Clifton [2005b]. Consider several parties having their own sets of items from a common domain. The problem is to securely compute the cardinality/size of the intersection of these local sets.

Formally, given p parties $P_0 \ldots P_{p-1}$ having local sets $S_0 \ldots S_{p-1}$, we wish to securely compute $|S_0 \cap \cdots \cap S_{p-1}|$. We can do this using a parametric commutative one-way hash function. One way of getting such a hash function is to use commutative public key encryption, such as Pohlig Hellman, and throw away the decryption keys. An encryption algorithm is commutative if, given encryption keys $K_1, \ldots, K_n \in K$, for any m in domain M, and for any permutation i, j, the following two equations hold.

$$E_{K_{i_1}}(\ldots E_{K_{i_n}}(m)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(m)\ldots) \tag{1}$$

$\forall m_1, m_2 \in M$ such that $m_1 \neq m_2$ and for given k, $\epsilon < \frac{1}{2^k}$

$$Pr(E_{K_{i_1}}(\ldots E_{K_{i_n}}(m_1)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(m_2)\ldots)) < \epsilon \tag{2}$$

All p parties locally generate their public key-pair (E_i, D_i) for a commutative encryption scheme. (They can throw away their decryption keys, since these will never be used.) Each party encrypts its items with its key and passes it along to the other parties. On receiving a set of (encrypted) items, a party encrypts each item and permutes the order before sending it to the next party. This is repeated until every item has been encrypted by every party. Since encryption is commutative (i.e., due to Eq. (1)), the resulting values from two different sets will be equal if and only if the original values were the same (i.e., the item was present in both sets). Thus, we need only count the number of values that are present in *all* of the encrypted itemsets. This can be done by any party. Since no party has all keys needed for decryption, none can learn which items are present in the intersection set.

The complete protocol is shown in Algorithm 8. This protocol is not completely secure; along with the final intersection size, the sizes of intersection of all of the subsets are also revealed. For details and proof, see Vaidya and Clifton [2005b]. In the context of our PPID3 algorithm, whenever the *DistributionCounts*() function is called (at any node of the tree), each party creates its own vector of transactions using the local constraint set and then finds the size of the intersection set using the protocol described before. While getting the overall intersection size, each party can also learn the intersection sizes of all possible combinations of sets. Thus, if there are three parties with vectors X,Y, and Z, respectively, instead of learning just $|X \cap Y \cap Z|$, each party also learns $|X|$, $|Y|$, $|Z|$, $|X \cap Y|$, $|Y \cap Z|$, $|X \cap Z|$. While this disclosure is not necessary (and in particular, the size of individual sets can be obscured at small cost; see Vaidya and Clifton [2005b]), in Section 7.1 we present a completely secure (though less efficient) protocol based on secure scalar product [Goethals et al. 2004]. The protocol we have just presented is sufficient to ensure individual privacy, as the intersection sizes do not reveal individually identifiable information.

3. USING THE TREE

Instance classification proceeds as in the original ID3 algorithm, except that the nodes (and attributes of the database) are distributed. The site requesting classification (e.g., a master site) knows the root node of the classification tree. The basic idea is that control passes from site to site, based on the decision made. Each site knows the transaction's attribute values for the nodes at its site (and can thus evaluate the branch), but knows nothing of the other attribute values. The complete algorithm is given in Algorithm 9, and is reasonably self-explanatory if viewed in conjunction with Algorithm 7.

We now give a demonstration of how instance classification would actually happen in this instance for the tree built with the UCI weather dataset [Blake and Merz 1998]. Assume two sites: The weather observatory collects information about relative humidity and wind, and a second collects temperature and cloud-cover forecast as well as the class ("Yes" or "No"). Suppose we wish to know if it is a good day to play tennis. Neither site wants to share its forecast, but both are willing to collaborate to offer a "good tennis day" service. The

Algorithm 8. Securely Computing Size of Intersection Set

```
Require: $p$ parties, $P_0, \ldots, P_{p-1}$
Require: each party has a local set $S_i$
  Generate the commutative encryption key-pair $(E_i, D_i)$ {Throw away the de-
  cryption keys, since they will not be needed.}
  $M \leftarrow S_i$
  for $p-1$ steps do
    $M' \leftarrow newarray[|M|]$
    $j \leftarrow 0$
    for each $X \in M$ do
      $M'[j++] \leftarrow encrypt(X, E_i)$
    end for
    permute the array $M'$ in some random order
    send the array $M'$ to party $P_{i+1 \bmod p}$
    receive array $M$ from party $P_{i-1 \bmod p}$
  end for
  {Final Encryption}
  $M' \leftarrow newarray[|M|]$
  $j \leftarrow 0$
  for each $X \in M$ do
    $M'[j++] \leftarrow encrypt(X, E_i)$
  end for
  permute the array $M'$ in some random order
  send $M'$ to party $P_{i+1 \bmod p}$
  each party $P_i$ produces array $I_i$ containing only (encrypted) items present in all
  arrays received. (Note all of the computed arrays $I_i$ will be identical).
  Any party, say $P_0$, broadcasts the result $(|I_0|)$
```

Algorithm 9. classifyInstance(instId, nodeId): returns the class/distribution for the instance represented by instId

```
1: {The start site and ID of the root node is known}
2: if $nodeId$ is a LeafNode then
3:   return class/distribution saved in $nodeId$
4: else {$nodeId$ is an interior node}
5:   $Nd \leftarrow$ local node with id $nodeId$
6:   $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
7:   $childId \leftarrow Nd.value$
8:   return $childId.Site.classifyInstance(instId, childId)$ {Actually tail recur-
     sion: this site need never learn the class}
9: end if
```

classification tree is shown in Figure 2, with S1 and S2 corresponding to the site having information on that node. The private information for each site is shown within italics. If today is sunny with normal humidity, high temperature, and weak wind, classification would proceed as follows: We know that site 1 has the root node (we don't need to know anything else). Site 1 retrieves the attribute for from $S1L1$: Outlook. Since the classifying attribute is outlook and site 1 knows the forecast is sunny, the token S2L2 is retrieved. This indicates that the next step is at site 2. Site 2 is called with the token S2L2, and retrieves the attribute for S2L2: Humidity. The humidity forecast is normal, so the token

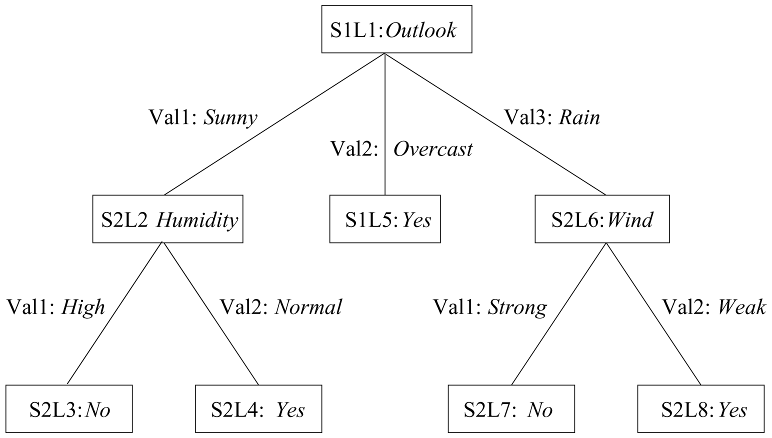


Fig. 2. The privacy-preserving ID3 decision tree on the weather dataset (mapping from identifiers to attributes and values is known only at the site holding the attributes).

S2L4 is retrieved. Since this token is also present at site 2, it retrieves the class value for nodeId S2L4 and returns it: We receive our answer of "Yes".

4. SECURITY DISCUSSION

We first discuss the security of the constituent algorithms, then the security of the complete algorithm. Although it may seem that some of the constituent algorithms leak a large quantity of information, in the context of the full algorithm the leaked information can be simulated by knowing the distribution counts at each node, so overall privacy is maintained.

LEMMA 4.1. *Algorithm* 2 *reveals nothing to any site except whether the total number of attributes left is* 0. *Algorithm* 3 *reveals only the count of instances corresponding to all combinations of constraint sets for each class. Algorithm* 4 *finds if all transactions have the same class, revealing only the aforesaid class distributions.*

PROOF. Refer to Vaidya and Clifton [2005a]. □

LEMMA 4.2. *Algorithm* 6 *reveals nothing except the counts* S, S_{a_i}, *and the constituent subcounts described in Lemma* 4.1 *for each attribute value* a_i *and class* j, *assuming the number of distinct class values is known.*

PROOF. Refer to Vaidya and Clifton [2005a]. □

LEMMA 4.3. *Algorithm* 5 *finds the site with that attribute having the maximum information gain, while revealing only the best information gain at each site and the information discussed in Lemma* 4.2.

PROOF. Refer to Vaidya and Clifton [2005a]. □

Further reduction of the information revealed is possible by using a secure protocol for finding the maximum among a set of numbers. This would reveal only that site having the attribute with the maximum information gain and nothing else.

THEOREM 1. *Algorithm* 7 *computes the decision tree while revealing only:*

—*the distribution subcounts of each node, as described in Lemma* 4.1. *(The full counts, and some of the subcounts, can be computed knowing the distribution counts at the leaves.)*

—*the best information gain from each site at each interior node (as discussed previously, this leak can be reduced.).*

PROOF. Knowing the final tree, the simulator at each site can uniquely determine the sequence of node computations at a site and list the function calls occurring due to this. Given this function-call list, if the messages received in each function call can be simulated, the entire algorithm can be proven secure.

Line 1 is an invocation of Algorithm 2. The result is simulated as either true or false, depending on whether the node in question is a leaf node in the final tree.

Line 3 is an invocation of Algorithm 3. The actual counts are given by the counts *in* the leaf node, which are known to the site P_k that invoked the algorithm. The subcounts revealed by Algorithm 3 are presumed known.

Line 7 is an invocation of Algorithm 4. If the node in question is not a leaf node in the final tree, the result is false. Otherwise, the result is the nodeId of the leaf node.

Line 10 consists of an invocation of Algorithm 5. The result is actually equal to the site which will own the child node. This information is known from the tree structure. The subcounts and information gain values revealed during this step are presumed known.

Line 15 is a recursive invocation that returns a node identifier: a part of the tree structure.

Since all of the aforementioned algorithms have been proven secure, applying the composition theorem, Algorithm 7 is secure. The repeated invocations of the cardinality of set intersection protocol are valid because in each invocation, a new set of keys is chosen. This ensures that messages cannot be correlated across calls. □

THEOREM 2. *Algorithm* 9 *reveals nothing other than the leaf node classifying the instance.*

PROOF. All the computations are local. The only information passed between various sites are node identifiers. This list of node identifiers can be easily simulated from the classification tree once the final leaf is known. □

5. COMPUTATION AND COMMUNICATION ANALYSIS

The communication/computation analysis depends on the number of transactions, number of parties, number of attributes, number of attribute values per attribute, number of classes, and complexity of the tree. Assume that there are n transactions, k parties, c classes, r attributes, p values per attribute (on average), and q nodes in final classification tree. We now give a rough analysis of the cost involved, in terms of the number of set intersections required for building the tree (erring on the conservative side).

At each node in the tree, the best classifying attribute needs to be determined. To do this, the entropy of the node needs to be computed as well as the information gain per attribute. Computing the entropy of the node requires c set intersections (1 per class). Computing the gain of one attribute requires cp set intersections (1 per attribute value and class). Thus, finding the best attribute requires cpr set intersections. Note that this analysis is rough and assumes that the number of attributes available at each node remains constant. In actuality, this number linearly decreases with the depth of the node in the tree (this has little effect on our analysis). In total, every node requires $c(1+pr)$ set intersections. Therefore, the total tree requires $cq(1+pr)$ set intersections.

The intersection protocol requires that the set of each party be encrypted by every other party. Since there are k parties, k^2 encryptions are required and k^2 sets are transferred. Since each set can have at most n transactions, the upper bound on computation is $O(nk^2)$ and the upper bound on communication cost is also $O(nk^2 * bitsize)$ bits. Note that, in general, the size of the sets is likely to be a lot lower than the total size of the database. Due to the filters, only a fraction of the transactions actually reach any node.

Therefore, in total the entire classification process will require $O(cqnk^2(1+pr))$ encryptions and $cqnk^2(1+pr) * bitsize$ bits communication in the worst case. Note that the encryption process can be completely parallelized, reducing the required time by an order of k.

Once the tree is built, classifying an instance requires no extra overhead and is comparable to the original ID3 algorithm.

6. EXPERIMENTAL VALIDATION

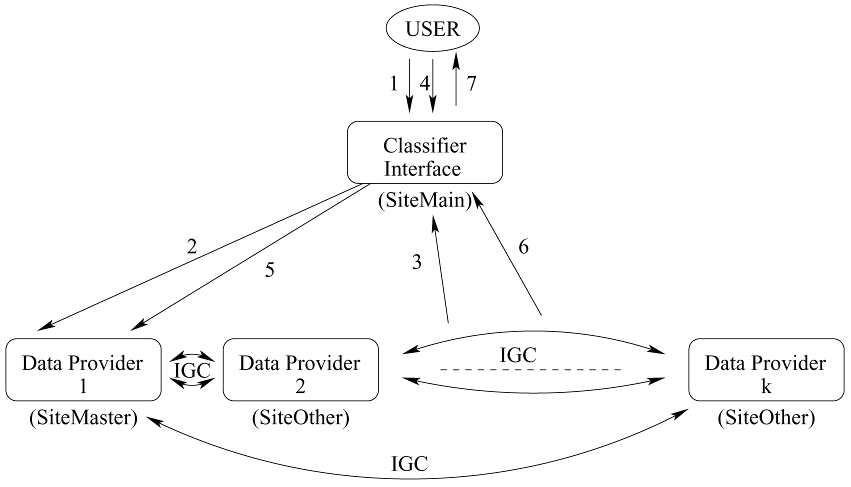
In this section we cover the details of our implementation of the privacy-preserving distributed ID3 algorithm on modern hardware and present experimental results demonstrating the usability of this algorithm. We also show that encryption is the limiting factor of the performance of this algorithm; thus, measuring encryptions is a good way to judge similar protocols.

6.1 Adapting Weka

Weka [Witten and Frank 1999], developed at the University of Waikato in New Zealand, is a collection of machine-learning algorithms for data-mining tasks implemented in Java. Apart from providing algorithms, it is a general implementation framework, along with support classes and documentation. It is extendible and convenient for prototyping purposes. However, the Weka system is a centralized system meant to be used at a single site.

We have implemented our privacy-preserving ID3 algorithm[1] and integrated it into Weka version 3.4. We first developed a general model of operation to extend Weka for privacy-preserving distributed classification. We extended the Weka core classes `Instance` and `Instances`, to provide support for distributed

[1]The current implementation is available at http://www.cs.purdue.edu/ clifton/software/ppid3/.



1: User Requests a classifier
2: Main Site sends request to Master to initiate model construction
IGC: Inter Group Communication (Protocol)
3: Classifier is built
4: User requests classification of an instance
5: Main site asks Master to initiate classification
6: Class of instance returned to Master
7: Class of instance returned to User

Fig. 3. Basic classifier model.

instances. A distributed instance consists of only the key identifier and those site identifiers for the sites that together contain the whole instance.

The general model of privacy-preserving distributed classification is as follows. The user initiates a request to build a classifier and then requests classification of an instance whenever required. The process of building the classifier needs to be coordinated so that the data sites locally construct enough state to enable them to jointly satisfy a classification request. To this end, every centralized classification class must be extended with a distributed class that provides the same functionality; however, the implementation of these functions/messages is done in a distributed manner.

In the current case, we extend the `Id3` class with the new class `distId3` that fulfills the same contracts as promised by `Id3`, in a distributed manner. When called with normal instance(s) the behavior is identical to ID3; when called with distributed instance(s) the algorithm performs in a distributed fashion. There is one global coordinating class/interface that provides access to the classification functionality. We call this the `RmtMain` interface, and it basically provides functionality for two methods: `buildClassifier` and `classifyInstance`. However, since this interface is present at a user site, the class implementing it should have access to no private information. The class implementing this is the `SiteMain` class, which is initialized with the appropriate site information and called by the `distId3` class appropriately. There are two other interfaces: the `RmtMaster` and `RmtSlave` interfaces. One of the local sites is designated as the master site while the others are treated as slave sites. This essentially means that the master site is the one which will be called by the `RmtMain` interface and for the which the `buildClassifier` and `classifyInstance` / `distributionForInstance` methods are called initially. The master site coordinates with the slave sites to perform the required action. Any site can be randomly chosen to be the master. The master interface is implemented by the `SiteMaster` class, while the slave interface is implemented by the `SiteOther` class. Figure 3 demonstrates the basic usage model. Java RMI is used to implement the distributed protocol between all of the sites, `SiteMaster`, and `SiteOther`. The commutative hash function used in the algorithm is implemented with a Pohlig-Hellman cipher written using `libcrypto` from an OpenSSL [Cox et al. 2005] version 2.8-beta, which contains a number of machine code optimizations for exponentiation. This is done through the Java JNI, allowing the Java implementation of the algorithm to take advantage of the native machine code optimizations present in OpenSSL. This is critical to the performance of the algorithm because the encryption routine is the bulk of the algorithm, as demonstrated in Section 6.2. We tried a number of options for implementing the encryption routine. Pure Java proved unusably slow, on the order of days. The OpenSSL implementation is significantly faster, but also varies greatly with architecture. For example, its performance on Sun machines with crypto-accelerators is much slower than Linux on Intel Xeon processors. We speculate that this is due in part to the time taken to buffer the data into the specialized hardware versus keeping them in the primary processor. In addition, differences in integer arithmetic and OpenSSL optimizations will cause large performance variations between platforms.

6.2 Experimental Results

In order to determine the limiting factors of the distributed algorithm and the possible areas for improvement, we conducted various experiments on the running time of building a distributed classification tree. Our hypothesis is that encryption is the primary performance bottleneck of this algorithm, thus any significant speed improvements must be made to the encryption techniques used in the commutative hash function. We ran experiments in two- and three-site configurations, using three identical computers. Each computer consisted of an Intel® Xeon™ 3 GHz processor, with 1GB RAM, hyperthreading enabled, and running the Linux SMP kernel version 2.6.11.10. The time to build the distributed ID3 tree and to classify instances was measured for the car and weather datasets from the UCI repository [Blake and Merz 1998].[2] The UCI weather dataset consists of 4 attributes plus the class, and 14 transactions. The UCI car dataset consists of 6 attributes plus the class, and 1728 transactions. We created three datasets from the standard UCI car dataset: *car100*, *car50*, and *car25*, which correspond to 100, 50, and 25 percent, respectively, of the original car transactions. There are 1728 transactions in *car100*, 864 transactions in *car50*, and 432 transactions in *car25*. Resampling was done at random

[2]While these may not be interesting applications from a privacy point of view, they are commonly used for decision tree evaluation and thus appropriate choices for our goal of performance measurement.

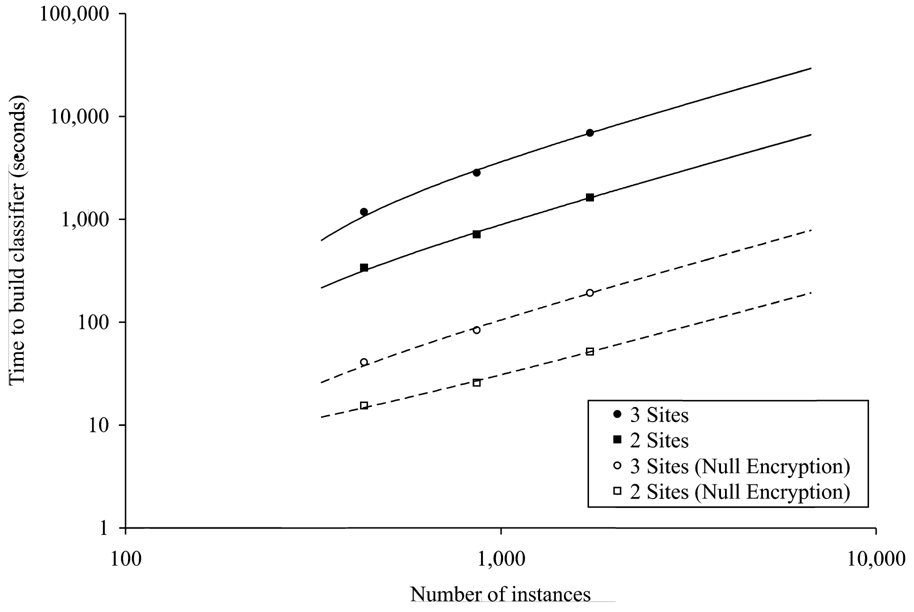


Fig. 4. A log-log scale plot of the average time to build a classification tree versus the number of instances used to create the tree, along with linear interpolations through the data points.

Table I. Classifying an Instance on Real Datasets

Sites:	2	3
car25	0.06 s	0.08 s
car50	0.12 s	0.15 s
car100	0.23 s	0.28 s

to maintain the class distribution, using Weka's supervised resample filter. As expected, the trees created by the distributed algorithm are identical to those of the original ID3 algorithm. The results from these experiments are shown in Figure 4 and Table I.

The ID3 tree for *car100* has 407 nodes, *car50* has 248 nodes, and *car25* has 178 nodes. These are quite complex. We can see that distributed ID3 build time scales linearly in the number of transactions, as expected. Note that the relationship between the instances used to build the trees and the number of nodes in the trees is *not* exactly linear. This is the primary reason why the data points on each plot are also not precisely linear. Increasing the number of parties causes a quadratic expansion in the amount of time required. One of the most important factors affecting the computation time of the protocol is the size of the tree built. Simpler trees are much faster to build. Once the classification tree is built, classifying an instance takes very little time, on par with the undistributed ID3 algorithm. Thus, if the relatively expensive protocol to build the tree has already been executed, it is a computationally trivial task to classify any given instance.

We attempted to fit the theoretical running time given in Section 6 to the experimental data. However, the gap between the two- and three-site data is too large, suggesting that our implementation is not achieving perfect parallelism. As the number of sites grows, the amount of parallelism achieved by our implementation drops off, and there is no factor of k speedup. This causes the gap in running time between two sites versus three to be a bit larger than the theoretical prediction.

In addition to knowing the running time of the algorithm, it is important to understand where the bottlenecks are, and what further speed improvements are possible. There are two primary locations for the possibility of performance bottlenecks in the distributed ID3 algorithm: the communication framework and steps of the protocol itself, or the encryption routines used in the commutative hash. Consider the case where privacy preservation is not needed, but where the data is still vertically partitioned across multiple sites. Performing the ID3 algorithm in this case can be bounded by sending all the data to one site, merging the data, and performing the single-site ID3 algorithm. We performed a simple experiment calculating the sum of the time to send the partitioned car dataset to one site, merging the datasets into one, and using Weka's built-in ID3 algorithm on the new dataset. This takes, on average, less than one second to complete. Next, we looked at the running time of our algorithm independent of commutative encryption. This demonstrates how fast the algorithm itself is, assuming no time is lost to encryption. To do this, we created a null commutative encryption algorithm to drop in place of the normal commutative encryption algorithm. All that the null encryption algorithm does is to pad the data to retain network transfer times, but performing no encryption; this is clearly commutative and requires insignificant running time. The results of this experiment, shown in Figure 4, make it clear that the complexity of this algorithm is in the commutative encryption routine itself, since the null encryption version runs an order of magnitude more quickly. This supports our hypothesis that encryption is the primary bottleneck and encryption type and count are the most important metrics for judging privacy-preserving data-mining algorithms.

A key point is that the actual implementation uses more secure methods than those described in this article. Thus, the method for Algorithm 2 actually uses homomorphic encryption to produce a protocol which is more resistant to collusion.

6.3 Experimental Conclusion

The distributed ID3 algorithm as presented in this article and as a working program is a significant step forward in creating usable, distributed, privacy-preserving data-mining algorithms. The running time of the algorithm, albeit slow compared to the insecure case, is certainly usable on everyday computing hardware. Three sites can securely compute a complex ID3 tree in a couple of hours. These techniques can be adapted to create other usable, privacy-preserving data-mining algorithms. It is clear that encryption speed is the most important factor when looking at the speed of this algorithm and others like it. Architecture and programming language choices can have a large effect on the performance. In order to make significant improvements to this algorithm either the encryption speed must be increased or the number of encryptions used must be decreased. Some possible routes to take are to use specialized hardware to speed-up the encryptions or to use elliptic curve encryption techniques [Cohen et al. 1998] to decrease the key-space necessary.

7. COMPLETELY SECURE ID3

In this section we discuss what would be required to obtain fully secure protocols for ID3 classification. There are two main problems. The first is that the set intersection algorithm reveals too much information (specifically, the sizes of all possible combinations of intersection sets). The second problem lies in the fact that the class counts are revealed for each node of the tree. While we had justified this leakage earlier, it would be nice to have technical solutions to this problem as well. Essentially, a completely secure multiparty dot product protocol could solve both of these problems. We now discuss the specific details.

7.1 Secure Multiparty Dot Product

In this subsection, we discuss a multiparty dot product protocol that is secure in the semi-honest model, even if $t \leq \lfloor \frac{k-1}{2} \rfloor$ of the k parties collude. We now describe the protocol and prove that it is secure under t party collusion in the semi-honest model.

7.1.1 *Homomorphic Public Key Encryption.* An important cryptographic tool used in our protocols is homomorphic public key encryption. In what follows, we briefly describe the properties of the homomorphic encryption.

Let $E_{pk}(.)$ denote the encryption function with public key pk and $D_{pr}(.)$ denote the decryption function with private key pr. A semantically secure public key cryptosystem is called homomorphic if it satisfies the following requirements: (1) Given the encryption of m_1 and m_2, $E_{pk}(m_1)$ and $E_{pk}(m_2)$, there exists an efficient algorithm to compute the public key encryption of $m_1 + m_2$, denoted $E_{pk}(m_1 + m_2) := E_{pk}(m_1) +_h E_{pk}(m_2)$. (2) Given a constant k and the encryption of m_1, $E_{pk}(m_1)$, there exists an efficient algorithm to compute the public key encryption of km_1, denoted $E_{pk}(km_1) := k \times_h E_{pk}(m_1)$.

For sake of completeness, we provide a brief definition of the homomorphic cryptosystem that we use in our experiments. Please refer to Damgard et al. [2003] for details. The Paillier cryptosystem, which is based on the composite residuosity assumption, satisfies the aforesaid properties and can be defined as follows.

—*Key Generation.* Let p and q be prime numbers, where $p < q$ and p does not divide $q - 1$. For the Paillier encryption scheme, we set the public key pk to n, where $n = p \cdot q$ and private key pr to (λ, n), where λ is the least common multiple of $p - 1, q - 1$.

—*Encryption with the Public Key.* Given n, the message m, and a random number r from 1 to $n - 1$, encryption of the message m can be calculated as follows: $E_{pk}(m) = (1+n)^m \cdot r^n \bmod n^2$. Also note that, given any encrypted message, we can get a different encryption by multiplying it with some random r^n.

—*Decryption with the Private Key.* Given n, the cipher text $c = E_{pk}(m)$, we can calculate the $D_{pr}(c)$ as follows: $m = \frac{(c^\lambda \bmod n^2)-1}{n}\lambda^{-1} \bmod n$, where λ^{-1} is the inverse of λ in modulo n.

—*Adding Two Ciphertexts ($+_h$).* Given the encryption of m_1 and m_2, $E_{pk}(m_1)$ and $E_{pk}(m_2)$, we can calculate the $E_{pk}(m_1+m_2)$ as follows.

$$\begin{aligned}
&E_{pk}(m_1) \cdot E_{pk}(m_2) \bmod n^2 \\
&\quad = \big((1+n)^{m_1} r_1^n\big) \cdot \big((1+n)^{m_2} r_2^n\big) \bmod n^2 \\
&\quad = \big((1+n)^{m_1+m_2} \cdot (r_1 \cdot r_2)^n\big) \bmod n^2 \\
&\quad = E_{pk}(m_1+m_2)
\end{aligned}$$

We would like to emphasize that this addition will actually return $E_{pk}(m_1 + m_2 \bmod n)$.

—*Multiplying a Ciphertext with a Constant ($k \times_h E_{pk}(m_1)$).* Given a constant k and the encryption of m_1,$E_{pk}(m_1)$, we can calculate $k \times_h E_{pk}(m_1)$ as follows.

$$\begin{aligned}
k \times_h E_{pk}(m_1) &:= E_{pk}(m_1)^k \bmod n^2 \\
&= \big((1+n)^{m_1} \cdot r_1^n\big)^k \bmod n^2 \\
&= (1+n)^{km_1} \cdot r_1^{kn} \bmod n^2 \\
&= E_{pk}(k \cdot m_1)
\end{aligned}$$

In our protocols, we use a slightly different version of the homomorphic encryption described before. We briefly summarize the threshold decryption next. The implementation details of the threshold version of the Paillier encryption can be found in Damgard et al. [2003] and Cramer et al. [2001].

—*Threshold Decryption.* Given the common public key pk, the private key pr corresponding to pk has been divided into $t+1$ pieces $pr_1, pr_2, \ldots, pr_{t+1}$. There exists an efficient, secure protocol $D_{pr_i}(E_{pk}(a))$ that outputs the random share of the decryption result s_i, along with the noninteractive zero knowledge proof $POD(pr_i, E_{pk}(a), s_i)$ showing that pr_i is used correctly. These shares can be combined to calculate the decryption result. Also, any subset of the private keys pr_i less than size $t+1$ cannot be used to decrypt the ciphertext correctly. In other words, any subset of s_i size less than $t+1$ does not reveal anything about the final decryption result. On the other hand, any subset of size t of different s_i values could be used to construct the original decryption.

Since we assume semi-honest parties in our case, we do not need the zero knowledge proofs associated with secure protocol $D_{pr_i}(E_{pk}(a))$. In the rest of this section, we show how to use such an encryption scheme to construct solutions secure against collusions.

Algorithm 10. Secure $\sum_{t=1}^{T}\left(\prod_{i=1}^{k}(Y_i[t])\right)$ Calculation

```
Require: Each party $P_i$ has a vector $Y_i$
Ensure: Return $\sum_{t=1}^{T}\left(\prod_{i=1}^{k}(Y_i[t])\right)$
  for $P_1$ do
    for all t do
      Choose random $r^n \bmod n^2$
      Set $R_1[t] \leftarrow r^n \bmod n^2$
      if $Y_1[t] = 0$ then
        Set $E_1[t] \leftarrow r^n \bmod n^2$
      else
        Set $E_1[t] \leftarrow (1+n) \cdot r^n \bmod n^2$
      end if
    end for
    Send $E_1$ to $P_2$
  end for
  for $P_i$ where $1 < i \leq k$ do
    Get $E_{i-1}$ from $P_{i-1}$
    for all t do
      Choose random $r^n \bmod n^2$
      Set $R_i[t] \leftarrow r^n \bmod n^2$
      if $Y_i[t] = 0$ then
        Set $E_i[t] \leftarrow r^n \bmod n^2$
      else
        Set $E_i[t] \leftarrow E_i[t] \cdot r^n \bmod n^2$
      end if
    end for
    Send $E_i$ to $P_{i+1}$
  end for
  for $P_k$ do
    Set $s \leftarrow 1$
    for all t do
      $s \leftarrow s \cdot E_k[t] \bmod n^2$
    end for
    Send $s \cdot r^n \bmod n^2$ to all $P_i$ where $1 \leq i \leq k$
  end for
  At least $t+1$ of the parties jointly decrypt the $s$ to learn the final result
```

7.1.2 *Protocol.* Our goal is to securely calculate

$$\sum_{t=1}^{T}\left(\prod_{i=1}^{k}(Y_i[t])\right), \tag{3}$$

where Y_i is the party P_i's 0/1 vector with T elements. Especially, we want to make sure that even if t of the total k parties collude (where $t \leq \lfloor\frac{k-1}{2}\rfloor$), our protocol is secure. Such a protocol could be easily given using a threshold homomorphic encryption that requires at least $t+1$ shares of the private key for decryption. Before we explain our protocol, we would like to note a few facts about Eq. (3).

First, let $S^v[t] = \prod_{i=1}^{v}(Y_i[t])$; note that $S^v[t]$ is equal to 1 if and only if all $Y_i[t]$ values are 1. Also, if $Y_{v+1} = 1$, then $S^{v+1}[t] = S^v[t]$ else $S^{v+1}[t]$ is equal to 0. $\sum_{t=1}^{T} S^k[t]$ is the result we want to learn. These facts imply that, given $E_{pk}(S^v[t])$, we can calculate $E_{pk}(S^{v+1}[t])$ as $E_{pk}(S^v[t]) \cdot r^n \bmod n^2$ for any random r if $Y_{v+1} = 1$ (note that multiplying by any $r^n \bmod n^2$ does not change the decryption result). Similarly, we can calculate $E_{pk}(S^{v+1}[t])$ as $r^n \bmod n^2$ for random r if $Y_{v+1} = 0$ ($E_{pk}(0) = r^n \bmod n^2$ and $E_{pk}(1) = (1+n) \cdot r^n \bmod n^2$ for any (random) r). Also note that $E_{pk}(\sum_{t=1}^{T} S^k[t]) = \prod_i^T E_{pk}(S^k[t]) \bmod n^2$ since, $E_{pk}()$ is homomorphic.

Using preceding observations, it is easy to construct a secure protocol to calculate Eq. (3). First, party P_1 creates $E_{pk}(S^1[t])$ based on $Y_1[t]$ for all t and sends all $E_{pk}(S^1[t])$ to P_2. Later on, P_2 calculates $E_{pk}(S^2[t])$ based on $Y_2[t]$ and $E_{pk}(S^1[t])$ using the preceding observations and sends all $E_{pk}(S^2[t])$ to P_3, and so on.[3] Finally, P_k calculates the $\prod_i^T E_{pk}(S^k[t]) \bmod n^2$ and sends it to every party. Using the threshold encryption all parties jointly decrypt $E_{pk}(\sum_{t=1}^{T} S^k[t])$ to learn the desired result. Algorithm 10 describes the details.

7.1.3 *Security of the Algorithm.* It is easy to prove that the aforesaid protocol is secure in the semi-honest model under at most t party collusions, given that we are using a threshold homomorphic encryption scheme secure against t-party collusions. We need to show that the view of any coalition of t parties can be simulated by their inputs and the output.

Before we define a simulator S that simulates the colluding parties' views, we need to understand the execution view of each party. Note that during the execution of the protocol, each party P_i sees its own input Y_i, E_{i-1}, (i.e., the messages sent by P_{i-1}), R_i, s, and the messages exchanged to decrypt s (let m_i be the concatenation of these messages). Let $view_i^{\Pi}(\bar{x}) = (Y_i, E_{i-1}, R_i, s, m_i)$ denote the view of part P_i. In the following, we define a simulator S to prove that our algorithm is secure.

THEOREM 3. *Algorithm 10 is secure against the collusion of at most t semi-honest parties.*

PROOF. Let us assume that parties $P_{i_1}, P_{i_2}, \ldots, P_{i_t}$ collude. Let $I = (i_1, i_2, \ldots, i_t)$ be the set of the colluding parties. Furthermore, assume that $i_1 < i_2, \cdots < i_t$. We can define simulator S, which takes the $Y_{i_1}, Y_{i_2} \ldots, Y_{i_t}$ values, public key, private keys of the colluding parties, and the final result as inputs, as follows.

```
For each $j = 1$ to $t$ do
  Generate $R_{i_j}$ randomly
  If $(i_j - 1) \notin I$ and $i_j > 1$
    Generate $E_{i_j-1}$ using random numbers mod$n^2$
  If $i_j \neq 1$
    Generate $E_{i_j}$ based on $Y_{i_j}$ and $E_{i_j-1}$
  else
    Generate $E_{i_j}$ based on $Y_{i_j}$
end For
Generate $s$ by using the public key and the final result
Run the simulator for secure joint decryption[4]
Output all $Y_{i_j}$ ,$E_{i_j}$,$R_{i_j}$, $s$ and $m_{i_j}$ values
```

[3]$E_i[t]$ denotes $E_{pk}(S^i[t])$ in Algorithm 10.

To conclude the proof, we need to show that S runs in polynomial time and that the output of S is computationally indistinguishable from the joint views of colluding parties with respect to the original protocol execution. Clearly, S as defined previously runs in polynomial time with respect to its inputs.

Now, we need to prove that the output of the simulator is computationally indistinguishable from the views of the parties in the real protocol execution. First, note that, except for the parts where $(i_j - 1) \notin I$ and the execution of the simulator for joint decryption, S replicates the real execution. In other words, if all E_{i_j-1} and the output of the joint decryption simulator are generated with the same distribution seen in the protocol execution, the output of S would be statistically indistinguishable from the views of the parties in real execution.

The question we need to answer is what happens when we use randomly generated values E_{i_j-1} for $(i_j - 1) \notin I$ and the joint decryption simulator output. If, in this case, the output of the S is *not* computationally indistinguishable from the view of the parties, then either E_{i_j-1} for $(i_j - 1) \notin I$ or the joint decryption simulator output is *not* computationally indistinguishable from uniform random values. Either case would be a contradiction. Our homomorphic threshold encryption scheme is semantically secure against any coalitions of size t; this implies that encrypted values are computationally indistinguishable from uniform random values for any coalition of size t. For the second case, since the joint decryption protocol is secure, the output of the simulator for this protocol should be indistinguishable from the real execution. □

7.1.4 *Computational and Communication Cost.* Before we analyze the computational and communication costs of the protocol, we mention one important optimization technique. Since R_i does not depend on any of the inputs, each party can generate the R_i vectors before execution of the protocol. This optimization implies that calculating E_i from E_{i-1} requires at most T multiplications during the protocol execution. Also we know that the threshold decryption with $t+1$ pieces of secret key requires $2(t+1)$ exponentiations [Damgard et al. 2003].

Using the preceding observations, it is easy to find an upper bound on the total running time and the number of bits transferred. Given the total vector size T and the number of parties k, we need $T \cdot k$ multiplications to compute the E_k, another T multiplications to compute s, and $2(t+1)$ exponentiations to calculate the final result from s. Therefore, total computation cost is $(T+1) \cdot k$ $2 \cdot log(n)$ bit multiplications and $2(t+1)$ exponentiations. Similarly, we need to transfer $T \cdot k \cdot 2 \cdot log(n)$ bits to calculate E_k and $(t+2) \cdot log(n)$ bits to decrypt s. Here, T is equal to the database size n.

7.2 Experimental Comparison

We now look at the computational overhead of the fully secure protocol. We implemented the secure cardinality of set intersection protocol (Algorithm 10) in Java. Experiments were carried out using the same setup described in Section 6. Table II(a) gives the time required for one evaluation of the completely

[4]This simulator exists and runs in polynomial time because the joint decryption protocol is secure.

Table II. Running Time Estimates for Completely Secure Protocol

	Number of Parties		
Vector size	2	3	5
100	0.55	0.80	1.30
1000	5.12	7.66	12.72
10000	50.92	76.24	126.87

(a) cost of one completely secure scalar product (minutes).

	Number of Parties		
Dataset	2	3	5
Car25	12	34	110
Car50	18	50	164
Car100	30	83	273

(b) sample cost of building classification tree (days).

secure protocol for different vector sizes and different numbers of parties. The threshold size (for threshold encryption) is set to 2. However, the number of shares has only negligible contribution to the overall cost and can be ignored. From this we can estimate the overall time required for building the classification tree. Comparing the two protocols, we can see that one scalar product over a vector of length 1,000 between 3 parties takes around 7.6 minutes. The cost is measured assuming no precomputation. The commutative encryption-based set intersection protocol requires about 3.3 minutes with no parallelization. Another key factor for the set intersection protocol is the size of the sets. Here, we assume that all of the sets are full; however, in general each set is only likely to contain a small fraction of the total transactions. This will greatly reduce the cost of the set intersection. For example, if only 10 percent of the transactions are included, the set intersection protocol will be 10 times faster (this optimization cannot apply for the secure scalar-product-based protocol, since both the 1's and the 0's need to be encrypted). Thus, in general, without significant precomputation, it is computationally much more expensive to use the fully secure protocol.

As can be seen from Table II(b), building the classifier for the completely secure protocol takes time on the order of months, as opposed to hours for the optimized protocol of Section 2. While it may be possible to develop a more efficient completely secure protocol, we believe these experiments show that it is possible to achieve sufficient practical levels of privacy at reasonable cost, even when fully private computations are impractical.

7.3 Making the Protocols Completely Secure

Replacing all set intersections with the aforesaid secure threshold multiparty dot product ensure two things: no inherent leakage through the protocol, and elimination of the leakage of counts. Basically, for the distribution counts, the final step of jointly decrypting the dot product is not performed. This leaves the counts split between all parties. For the leaves, this implies that Algorithm 9 (the classification protocol) has to change. When a leaf is reached, to get the classification, the final decryption has to be carried out. This needs to be done for every classification. One point to note, however, is that this might be overkill: If anyone is allowed to use the classification tree (including the participants) they could very easily find out the actual values by posing as customers for all of the tree. The second problem lies with the interior nodes: specifically, finding the attribute with the highest information gain. This requires that the information gain be calculated from the splits present with the parties. This is quite complex, since it involves calculating the entropy (i.e., split log computations, as well as split multiplications). A general circuit could be built to compute this. A more efficient possibility is to instead use the Gini index [Duda and Hart 1973] instead of the information gain. The Gini index only requires computing squares instead of computing logs over split inputs, and therefore is easier to compute. However, whether this is worth the additional computation cost depends on the application domain and security constraints.

8. RELATED WORK AND MODEL

There has been other work in privacy-preserving data mining. In the perturbation approach, pioneered by Agrawal and Srikant [2000], "noise" is added to the data before the data-mining process. Techniques are then used to mitigate the impact of the noise from the data-mining results. Several privacy-preserving data-mining algorithms have since been proposed [Agrawal and Aggarwal 2001; Evfimievski et al. 2002; Rizvi and Haritsa 2002] using this approach. However, recently there has been debate about the security properties of such algorithms [Kargupta et al. 2003; Huang et al. 2005].

The approach to protecting the privacy of distributed sources using cryptographic techniques was first applied in the area of data mining for the construction of decision trees, by Lindell and Pinkas [2000, 2002]. This work closely followed the secure multiparty computation approach discussed next, achieving "perfect" privacy (i.e., nothing is learned that could not be deduced from one's own data and the resulting tree). The key insight was to trade off computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. There has since been work to address association rules in horizontally partitioned data Kantarcioglu and Clifton [2002, 2004], EM clustering in horizontally partitioned data [Lin et al. 2005], K-means clustering in vertically partitioned data [Vaidya and Clifton 2003; Jagannathan and Wright 2005], association rules in vertically partitioned data Vaidya and Clifton [2005b, 2002], and generalized approaches to reducing the number of "online" parties required for computation [Kantarcioglu and Vaidya 2002]. In terms of classification, Vaidya and Clifton [2004] propose a protocol to build a naïve Bayes classifier for vertically partitioned data, while Wright and Yang [2004] propose a similar protocol for learning the Bayesian network structure.

There has been work in distributed construction of decision trees on vertically partioned data. Wang et al. present a solution based on passing the transaction identifiers between sites [Wang et al. 2006]; while this does not reveal specific attribute values, parties learn which transactions follow which path down the tree, enabling, for example, one site to say "These two individuals have the same attribute values." Du and Zhan [2002] do suggest a way of building a privacy-preserving decision tree classifier for vertically partitioned data. Their method is limited to two parties, assumes that both parties have the class attribute, and is unimplemented. These works make trade-offs between efficiency and information disclosure; however, all maintain provable bounds on disclosure.

The methodology for proving the correctness of the algorithm comes from secure multiparty computation [Yao 1986; Goldreich et al. 1987; Goldreich 2004].

Recently, there has been a renewed interest in this field, and a good discussion can be found in Du and Atallah [2001]. Currently, assembling these approaches into efficient privacy-preserving data-mining algorithms, and proving them secure, is a challenging task. This article demonstrates how these methods can be combined to implement a standard data-mining algorithm with provable privacy and information disclosure properties. Our hope is that as the library of primitives and known means for using them grows, standard methods will develop to ease the task of developing privacy-preserving data-mining techniques.

9. CONCLUSIONS

Also, it is possible to extend the protocols developed such that the class of each instance is learned only by the party holding the class attribute (nothing is learned by the remaining parties). In some cases, this might be preferable.

The major contributions of this article are the following.

—It proposes a new protocol to construct a decision tree on vertically partitioned data with an arbitrary number of parties where only one party has the class attribute. (The method is trivially extendible to the case where all parties have the class attribute, and in fact this causes a significant increase in the efficiency of the protocol.)

—The article presents a general framework in which distributed classification would work and addresses how such a system should be constructed.

—The article reports the first results on an actual implementation of a secure multiparty-computation-based protocol. As such, it serves to show that the methods can actually be built and are feasible.

While the tree creation times may seem excessive, they must be compared with the alternative. Privacy-preserving data mining enables applications that would not happen at all without use of these methods (witness the Terrorism Information Awareness program [Lewis 2003]). Even where alternatives exist, such as obtaining consent or Institutional Review Board approval for disclosing individually identifiable data on human subjects, the hours or days of computation pale in comparison to the weeks or months required for approval. This article demonstrates that such applications are feasible.

REFERENCES

Agrawal, D. and Aggarwal, C. C. 2001. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, CA. ACM Press, New York, 247–255.

Agrawal, R., Evfimievski, A., and Srikant, R. 2003. Information sharing across private databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA. ACM Press, New York.

Agrawal, R. and Srikant, R. 2000. Privacy-Preserving data mining. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Dallas, TX. ACM Press, New York, 439–450.

Atallah, M. J., Elmongui, H. G., Deshpande, V., and Schwarz, L. B. 2003. Secure supply-chain protocols. In *Proceedings of the IEEE International Conference on E-Commerce*, Newport Beach, CA. IEEE Computer Society Press, 293–302.

Blake, C. and Merz, C. 1998. UCI repository of machine learning databases. http://citeseer.comp.nus.edu.sg/context/123650/0.

COHEN, H., MIYAJI, A., AND ONO, T. 1998. Efficient elliptic curve exponentiation using mixed coordinates. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*. Springer-Verlag, London, UK, 51–65.

COX, M. J., ENGELSCHALL, R. S., HENSON, S., AND RIE, B. L. 1998–2005. The OpenSSL Toolkit.

CRAMER, R., DAMGARD, I., AND NIELSEN, J. B. 2001. Multiparty computation from threshold homomorphic encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*. Springer-Verlag, London, UK, 280–299.

DAMGARD, I., JURIK, M., AND NIELSEN, J. 2003. A generalization of Paillier's public-key system with applications to electronic voting.

DU, W. AND ATALLAH, M. J. 2001. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of the New Security Paradigms Workshop*. ACM, New York, 11–20.

DU, W. AND ZHAN, Z. 2002. Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, Maebashi City, Japan, C. Clifton and V. Estivill-Castro, Eds. vol. 14. Australian Computer Society, 1–8.

DUDA, R. AND HART, P. E. 1973. *Pattern Classification and Scene Analysis*. John Wiley & Sons, Hoboken, NJ.

EVFIMIEVSKI, A., SRIKANT, R., AGRAWAL, R., AND GEHRKE, J. 2002. Privacy preserving mining of association rules. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, 217–228.

EVIDENCE-BASED MEDICINE WORKING GROUP. 1992. Evidence-Based medicine. A new approach to teaching the practice of medicine. *J. Amer. Medical Assoc. 268,* 17 (Nov.), 2420–2425.

FREEDMAN, M. J., NISSIM, K., AND PINKAS, B. 2004. Efficient private matching and set intersection. In *Proceedings of the 23rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, International Association for Cryptologic Research (IACR), Interlaken, Switzerland. Springer, 1–19.

GOETHALS, B., LAUR, S., LIPMAA, H., AND MIELIKÄINEN, T. 2004. On secure scalar product computation for privacy-preserving data mining. In *Proceedings of the 7th Annual International Conference in Information Security and Cryptology (ICISC)*, New York, C. Park and S. Chee, Eds. vol. 3506, Springer, 104–120.

GOLDREICH, O. 2004. General Cryptographic Protocols, Vol. 2. In *The Foundations of Cryptography*, vol. 2. Cambridge University Press, Cambridge, UK, 599–764.

GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 1987. How to play any mental game—A completeness theorem for protocols with honest majority. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*. ACM, New York, 218–229.

HUANG, Z., DU, W., AND CHEN, B. 2005. Deriving private information from randomized data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Baltimore, MD. ACM Press, New York.

JAGANNATHAN, G. AND WRIGHT, R. N. 2005. Privacy-Preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, IL. ACM Press, New York, 593–599.

KANTARCIOGLU, M. AND CLIFTON, C. 2002. Privacy-Preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, Madison, WI. ACM Press, New York, 24–31.

KANTARCIOGLU, M. AND VAIDYA, J. 2002. An architecture for privacy-preserving mining of client information. In *Proceedings of the IEEE International Conference on Data Mining, Workshop on Privacy, Security, and Data Mining*, Maebashi City, Japan, C. Clifton and V. Estivill-Castro, Eds. vol. 14. Australian Computer Society, 37–42.

KANTARCIOĞLU, M. AND CLIFTON, C. 2004. Privacy-Preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowl. Data Eng. 16,* 9 (Sept.), 1026–1037.

KARGUPTA, H., DATTA, S., WANG, Q., AND SIVAKUMAR, K. 2003. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*. IEEE Computer Society, Los Alamitos, CA.

LEWIS, M. 2003. Department of defense appropriations act, 2004. Title VIII Section 8120. Enacted as Public Law 108-87.

LIN, X., CLIFTON, C., AND ZHU, M. 2005. Privacy preserving clustering with distributed EM mixture modeling. *Knowl. Inf. Syst. 8,* 1 (Jul.), 68–81.

LINDELL, Y. AND PINKAS, B. 2000. Privacy preserving data mining. In *Advances in Cryptology (CRYPTO)*. Springer-Verlag, New York, NY, 36–54.

LINDELL, Y. AND PINKAS, B. 2002. Privacy preserving data mining. *J. Cryptol. 15,* 3, 177–206.

QUINLAN, J. R. 1986. Induction of decision trees. *Mach. Learn. 1,* 1, 81–106.

RIZVI, S. J. AND HARITSA, J. R. 2002. Maintaining data privacy in association rule mining. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, Hong Kong, VLDB Endowment, 682–693.

SCHNEIER, B. 1995. *Applied Cryptography*, 2nd ed. John Wiley & Sons, Hoboken, NJ.

SHIRAO, K., HOFF, P., OHTSU, A., LOEHRER, P., HYODO, I., WADLER, S., WADLEIGH, R., O'DWYER, P., MURO, K., YAMADA, Y., BOKU, N., NAGASHIMA, F., AND ABBRUZZESE, J. 2004. Comparison of the efficacy, toxicity, and pharmacokinetics of a uracil/tegafur (UFT) plus oral leucovorin (LV) regimen between Japanese and American patients with advanced colorectal cancer: Joint United States and Japan study of UFT/LV. *J. Clinical Oncol. 22,* 17 (Sept. 1), 3466–3474.

VAIDYA, J. AND CLIFTON, C. 2002. Privacy-Preserving association rule mining in vertically partitioned data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada. ACM Press, New York, 639–644.

VAIDYA, J. AND CLIFTON, C. 2003. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC. ACM Press, New York, 206–215.

VAIDYA, J. AND CLIFTON, C. 2004. Privacy preserving naïve Bayes classifier for vertically partitioned data. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, Philadelphia, PA, 522–526.

VAIDYA, J. AND CLIFTON, C. 2005a. Privacy-Preserving decision trees over vertically partitioned data. In *the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Storrs, CT. Springer.

VAIDYA, J. AND CLIFTON, C. 2005b. Secure set intersection cardinality with application to association rule mining. *J. Comput. Security 13,* 4 (Nov.), 593–622.

WANG, K., XU, Y., SHE, R., AND YU, P. S. 2006. Classification spanning private databases. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*. AAAI Press, Menlo Park, CA.

WITTEN, I. H. AND FRANK, E. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA.

WRIGHT, R. AND YANG, Z. 2004. Privacy-Preserving Bayesian network structure computation on distributed heterogeneous data. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA. ACM Press, New York.

YAO, A. C. 1986. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, CA, 162–167.

Received September 2007; revised May 2008; accepted August 2008