Cryptographic techniques for privacy-preserving data mining

Benny Pinkas
HP Labs
benny.pinkas@hp.com

ABSTRACT

Research in secure distributed computation, which was done as part of a larger body of research in the theory of cryptography, has achieved remarkable results. It was shown that non-trusting parties can jointly compute functions of their different inputs while ensuring that no party learns anything but the defined output of the function. These results were shown using generic constructions that can be applied to any function that has an efficient representation as a circuit. We describe these results, discuss their efficiency, and demonstrate their relevance to privacy preserving computation of data mining algorithms. We also show examples of secure computation of data mining algorithms that use these generic constructions.

1. INTRODUCTION

Consider a scenario in which two or more parties owning confidential databases wish to run a data mining algorithm on the union of their databases without revealing any unnecessary information. For example, consider separate medical institutions that wish to conduct a joint research while preserving the privacy of their patients. In this scenario it is required to protect privileged information, but it is also required to enable its use for research or for other purposes. In particular, although the parties realize that combining their data has some mutual benefit, none of them is willing to reveal its database to any other party.

Note that we consider here a distributed computing scenario, rather than a scenario where all data is gathered in a central server, which then runs the algorithm against all data. (The central server scenario introduces interesting privacy issues, too, but they are outside the scope of this paper.)

How much privacy?. It is obvious that if a data mining algorithm is run against the union of the databases, and its output becomes known to one or more of the parties, it reveals something about the contents of the other databases. (For example, if a researcher from a medical institution learns that the overall percentage of patients that have a certain symptom is 50%, while he knows that this percentage in his population of patients is 40%, then he also learns that more than 50% of the patients of the other institutions have this symptom.) This leak of information is inevitable, however, if the parties need to learn this output.

Defining privacy. The common definition of privacy in the cryptographic community limits the information that is leaked by the distributed computation to be the information that can be learned from the designated output of the computation. Although there are several variants of the definition of privacy, for the purpose of this discussion we use the definition that compares the result of the actual computation to that of an "ideal" computation: Consider first a party that is involved in the actual computation of a function (e.g. a data mining algorithm). Consider also an "ideal scenario", where in addition to the original parties there is also a "trusted party" who does not deviate from the behavior that we prescribe for him, and does not attempt to cheat. In the ideal scenario all parties send their inputs to the trusted party, who then computes the function and sends the appropriate results to the other parties. Loosely speaking, a protocol is secure if anything that an adversary can learn in the actual world it can also learn in the ideal world, namely from its own input and from the output it receives from the trusted party. In essence, this means that the protocol that is run in order to compute the function does not leak any "unnecessary" information. (Of course, there are partial leaks of information that are harmless. It is hard, however, to decide which type of leakage can be tolerated. The cryptographic community therefore aims at designing protocols that do not reveal any information except for their designated output, and in many case such protocols can in fact be efficiently constructed.)

As an example for the definition of privacy, consider the following problem (described in [8]). Alice and Bob are both teaching the same class, and each of them suspects that one specific student is cheating. None of them is completely sure, though, about the identity of the cheater, and they would therefore like to compare the names of their two suspects. Since they care about their students privacy they wish that, (1) if they both have the same suspect, then they should learn his or her name, but (2) if they have different suspects then they should learn nothing beyond that fact. They therefore have inputs x and y, and wish to compute $f(x, y)$ which is defined as 1 if $x = y$ and 0 otherwise. (Note that if $f(x, y) = 0$ then each party does learn some information, namely that the other party's suspect is different than his/hers, but this is inevitable). There are several efficient solutions for this problem (described in [8; 14]). Note that a seemingly trivial solution, of the parties publishing and comparing the values $H(x)$ and $H(y)$, where H is a one-way function, is insecure (since given $H(x)$ Bob can do an exhaustive search over all students in the class, and find the

student whose name is hashed to this value).

Adversarial behavior. Privacy preserving protocols are designed in order to preserve privacy even in the presence of adversarial participants that attempt to gather information about the inputs of their peers. There are, however, different levels of adversarial behavior. Cryptographic research typically considers two types of adversaries: A *semi-honest* adversary (also known as a *passive*, or *honest but curious* adversary) is a party that correctly follows the protocol specification, yet attempts to learn additional information by analyzing the messages received during the protocol execution. On the other hand, a *malicious* adversary may arbitrarily deviate from the protocol specification. (For example, consider a step in the protocol where one of the parties is required to choose a random number and broadcast it. If the party is semi-honest then we can assume that this number is indeed random. On the other hand, if the party is malicious, then he might choose the number in a sophisticated way that enables him to gain additional information.)

It is of course easier to design a solution that is secure against semi-honest adversaries, than it is to design a solution for malicious adversaries. A common approach is therefore to first design a secure protocol for the semi-honest case, and then transform it into a protocol that is secure against malicious adversaries. This transformation can be done by requiring each party to use zero-knowledge proofs to prove that each step that it is taking follows the specification of the protocol. More efficient transformations are often required, since this generic approach might be rather inefficient and add considerable overhead to each step of the protocol.

We remark that the semi-honest adversarial model is often a realistic one. This is because deviating from a specified program which may be buried in a complex application is a non-trivial task, and because a semi-honest adversarial behavior can model a scenario in which the parties that participate in the protocol are honest, but following the protocol execution an adversary may obtain a transcript of the protocol execution by breaking into a machine used by one of the participants.

Note that we do not consider adversaries that change their inputs in order to gain more information about the inputs of the other parties. For example, an adversary could set its input to be the empty set, run the distributed computation, and then learn the result of applying the data mining algorithm to the input of the other parties (since its own input adds nothing to the union of the inputs). The operation of this adversary is legitimate according to our definition, since it learns exactly the same information it could learn by sending an empty input to the trusted party in the ideal scenario. One might wonder why our model does not cover such attacks. The main reason is that such attacks are hard to model, and that possible solutions are complex and tend to be application oriented – for example, we could require the parties to prove in zero-knowledge that their databases consist of data they gathered from users, but such a solution must be tailored to the specific scenario, and is not very efficient. We therefore assume that the authenticity of the inputs is assured by some external process. For example, parties might be interested in inputing their true data to the protocol since they are more interested is obtaining meaningful output from the distributed algorithm than they are in learning information about the input of their peers.

2. CRYPTOGRAPHIC RESULTS: SECURE FUNCTION EVALUATION

We describe here results of a body of cryptographic research that shows how separate parties can jointly compute any function of their inputs, without revealing any other information. As we argued above, these results achieve maximal privacy that hides all information except for the designated output of the function. This body of research attempts to model the world in a way which is both realistic and general. While there are some aspects of the "real world" that are not modeled by this research, the privacy guarantees and the generality of the results are quite remarkable.

2.1 The main building block - oblivious transfer

Oblivious transfer is a basic protocol that is the main building block of secure computation. It might seem strange at first, but its role in secure computation should become clear later. (In fact, it was shown by Kilian [11] that oblivious transfer is sufficient for secure computation in the sense that given an implementation of oblivious transfer, and no other cryptographic primitive, one could construct any secure computation protocol.)

The notion of 1-out-2 oblivious transfer was suggested by Even, Goldreich and Lempel [7] (as a variant of a different but equivalent type of oblivious transfer that has been suggested by Rabin [17]). The protocol involves two parties, the *sender* and the *receiver*. The sender's input is a pair (x_0, x_1) and the receiver's input is a bit $\sigma \in \{0, 1\}$. At the end of the protocol the receiver learns x_σ (and nothing else) and the sender learns nothing. In other words, if we use the notation $(input_A, input_B) \rightarrow (output_A, output_B)$ to define the result of a function, then oblivious transfer is the function $((x_0, x_1), \sigma) \rightarrow (\lambda, x_\sigma)$, where λ is the empty output.

It is known how to design oblivious transfer protocols based on virtually all known constructions of trapdoor functions, i.e. public key cryptosystems. In the case of semi-honest adversaries, there exist simple and efficient protocols for oblivious transfer [7; 9]. One straightforward approach is for the receiver to generate two random public keys, a key P_σ whose decryption key he knows, and a key $P_{1-\sigma}$ whose decryption key he does not know. The receiver then sends these two keys to the sender, who encrypts x_0 with the key P_0 and encrypts x_1 with the key P_1, and sends the two results to the receiver. The receiver can then decrypt x_σ but not $x_{1-\sigma}$. It is easy to show that the sender does not learn anything about σ, since the only message that she receives includes two random public keys, and she cannot find which one of them has a private key that is known to the receiver. As for the sender's privacy, if the receiver follows the protocol he only knows one private key and can therefore only decrypt one of the inputs, and if the encryption scheme is secure he cannot gain information about the other input. If we consider also malicious adversaries, then this oblivious transfer protocol must ensure that the receiver chooses the public keys appropriately. This can be done using zero-knowledge proofs that are used by the receiver to prove that he chooses the keys correctly. Fortunately, there are very efficient proofs for this case, see e.g. [15].

Oblivious transfer is often the most computationally intensive operation of secure protocols, and is repeated many times. Each invocation of oblivious transfer typically re-

quires a constant number of invocations of trapdoor permutations (i.e. public-key operations, or exponentiations). It is possible to reduce the amortized overhead of oblivious transfer to one exponentiations per a logarithmic number of oblivious transfers, even for the case of malicious adversaries [15].

2.2 Oblivious polynomial evaluation

The problem of "oblivious polynomial evaluation" (OPE) involves a sender and a receiver. The sender's input is a polynomial Q of degree k over some finite field $\mathcal{F}$ and the receiver's input is an element $z \in \mathcal{F}$ (the degree k of Q is public). The protocol is such that the receiver obtains $Q(z)$ without learning anything else about the polynomial Q, and the sender learns nothing. That is, the problem considered is the private computation of the function $(Q, z) \mapsto (\lambda, Q(z))$. This problem was introduced in [14], where an efficient solution was also presented. The overhead of that protocol is $O(k)$ exponentiations (using methods suggested in [15]). (Note that this protocol maintains privacy in the face of a malicious adversary. In the semi-honest case a simpler OPE protocol can be designed based on any homomorphic encryption scheme, with an overhead of $O(k)$ computation and $O(k|\mathcal{F}|)$ communication.)

The main motivation for using OPE is to utilize the fact that the output of a k degree polynomial is $(k+1)$-wise independent. Another motivation is that polynomials can be used for approximating functions that are defined over the Real numbers.

2.3 The two party case

In [19], Yao presented a constant-round protocol for privately computing any probabilistic polynomial-time function (where the adversary may be either semi-honest or malicious). Denote the parties as Alice (A) and Bob (B), and denote their respective inputs by x and y. Let f be the function that they wish to compute (for simplicity, assume that Bob should learn the value $f(x, y)$). The protocol is based on expressing f as a combinatorial circuit with gates defined over some fixed base $\mathcal{B}$. For example, $\mathcal{B}$ can include all the functions $g : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$. The bits of the input are entered into input wires and are propagated through the gates. Note that it is known that any polynomial-time function can be expressed as a combinatorial circuit of polynomial size (see, e.g. [18]).

Encoding the circuit. Loosely speaking, Yao's protocol works by having one of the parties (say Alice) first generate an "encrypted" or "garbled" circuit computing f and send its representation to Bob. The encrypted circuit is generated in the following way: First, Alice "hardwires" her input into the circuit, generating a circuit computing $f(x, \cdot)$. She then assigns to each wire i of the circuit two random ("garbled") values (W_i^0, W_i^1) corresponding to values 0 and 1 of the wire (the random values should be long enough to be used as keys to a pseudo-random function, e.g. 80-128 bits long).

Consider a gate g which computes the value of the wire k as a function of wires i and j. Alice prepares a table T_g that encrypts the garbled value of the output wire using the output of a pseudo-random function F keyed by the garbled values of the input wires i and j. The table therefore has four entries, one entry for every combination of input values. (Note that pseudo-random functions are usually realized using private-key primitives such as block ciphers or hash functions, and are therefore very efficient.) The table enables computation of the *garbled* output of g, from the garbled inputs to g. Given the two garbled inputs to g, the table does not disclose information about the output of g for any other inputs, nor does it reveal the values of the actual input bits.

The representation of the circuit includes the wiring of the original circuit (namely, a mapping from inputs or gate outputs to gate inputs), the tables T_g, and tables that translate the garbled values of the output wires of the circuit to actual 0/1 values. In this form the representation reveals nothing but the wiring of the circuit, and therefore Bob learns nothing from this stage. (We assume that the wiring of the circuit is not secret, which is obviously the case if the function f is public and the only secret information of Alice is her input x. Even if f is secret and is known only to Alice, it can be represented as being part of Alice's input and the parties can evaluate a universal circuit, i.e. a circuit whose input is $((f, x), y)$ and whose output is $f(x, y)$.)

Encoding Bob's input. The tables described above enable the computation of the garbled output of every gate from its garbled inputs. Therefore given these tables and the garbled values of the input wires of the circuit, Bob is able to compute the garbled values of its output wires and then translate them to actual values. In order for Bob to obtain the garbled values of the input wires, Alice and Bob engage, for each input wire, in a 1-out-of-2 oblivious transfer. In this protocol Alice is the sender, and her inputs are the two garbled values of this wire, and Bob is the receiver, and his input is his input bit. As a result of the oblivious transfer protocol Bob learns the garbled value of his input bit and nothing about the garbled value of the other bit, and Alice learns nothing.

Computing the circuit. At the end of the oblivious transfer stage Bob has sufficient information to compute the output of the circuit by his own. After computing $f(x, y)$, he can send this value to Alice if she requires it.

To show that the protocol is secure it should be proved that the parties learn nothing that cannot be computed based on the input and output only. The main observation regarding the security of each gate is that every masking value (e.g. output of the pseudo-random function F) is used only once, and that the pseudo-randomness of F ensures that without knowledge of the correct keys, i.e. garbled values of input wires, its output values look random. Therefore knowledge of one garbled value of each of the input wires discloses only a single garbled output value of the gate; while Bob cannot distinguish the other garbled value from random.

As for the security of the complete circuit, the oblivious transfer protocol ensures that Bob learns only a single garbled value for each input wire, and Alice does not learn which value it was. Inductively, Bob can compute only a single garbled output value of each gate, and in particular of the circuit. The method in which the tables were constructed hides the values of intermediate results (i.e. of gates inside the circuit).

It is possible to adapt the protocol for circuits in which gates have more than two inputs, and even for wires with more

than two possible values (which are possible since there is no need for a physical realization of the circuit, and might enable the construction of more compact circuits). The size of the table for a gate with ℓ inputs, which each can have d values, is d^ℓ.

Overhead. The overhead of the protocol involves: (1) Alice and Bob engaging in an oblivious transfer protocol for every input wire of the circuit, (2) Alice sending to Bob tables of size linear in the size of the circuit, and (3) Bob computing a pseudo-random function a constant number of times for every gate (this is the cost incurred in evaluating the gates). The number of rounds of the protocol is therefore constant (namely, two rounds using the oblivious transfer of [7; 9; 15]). The computation overhead is dominated by the oblivious transfer stage, since the evaluation of the gates uses pseudo-random functions which are very efficient compared to the oblivious transfer protocol.
A common belief with regard to Yao's protocol is that it is inherently inefficient, since it uses a circuit representation of the function. Let us examine the overhead more carefully.

- The computational overhead of the protocol is roughly linear in the size of Bob's input. To be more specific, the oblivious transfer stage requires one exponentiation (e.g. public key encryption) per bit of Bob's input. The amortized overhead can be reduced at the cost of increasing the communication overhead, see [15]. It is therefore reasonable to assume that about a hundred oblivious transfers can be computed per second [6].

- The communication overhead is linear in the size of the circuit. More accurately, a table of about 320-512 bits (40-64 bytes) is generated and communicated for every gate (assuming that all gates have two inputs and one output). The oblivious transfer stage requires communication linear in the number of input bits, of about three modular values per oblivious transfer (for the protocol of [2]).

The major factor dominating the overhead is, therefore, the size of the circuit representation of f. There are many functions for which we do not know how to create linear size circuits (e.g. functions computing multiplications or exponentiations, or functions that use indirect addressing). However, there are many other functions, notably those involving additions and comparisons, which can be computed by linear size circuits. The size of the input itself should also be reasonable. For example, we cannot expect that two parties, each of them holding a database with millions of entries, could run the protocol for computing a function whose inputs are the entire databases.

2.4 The multi-party case

In the multi-party scenario, there are protocols that enable the parties to compute any joint function of their inputs without revealing any other information about the inputs. That is, compute the function while attaining the same privacy as in the ideal model. This was shown to be possible in principle by Goldreich, Micali and Wigderson [10], Ben-Or, Goldwasser and Wigderson [3], and by Chaum, Crepau and Damgard [4], for different scenarios. These constructions, too, are based on representing the computed function as a circuit and evaluating it. The constructions do have, however, some additional drawbacks, compared to the two-party case:

- The computation and communication overhead of the protocol is linear in the size of the circuit, and the number of communication rounds depends on the depth of the circuit[1], unlike the two-party case where the number of rounds is constant. Furthermore, the protocol that is run for every gate of the circuit is more complex than the computation of a gate in the two-party case, especially in the malicious party scenario, and requires public-key operations (although the overhead is still polynomial).

- The multi-party protocols require each pair of parties to exchange messages (in order to compute each gate of the circuit). The required communication graph is, therefore, a complete graph, whereas a sparse communication graph could have been sufficient if no security was required. In many applications, for example applications run between a web server and many clients, it is impossible to require all pairs of parties to communicate.

- The security of the multi-party protocols is assured as long as there is no corrupt coalition of more than one half or one third of the parties (depending on the scenario). In many situations, however, it is impossible to ensure that the number of corrupt parties is smaller than such a threshold (for example, consider a web application in which anyone can register and participate, and which, therefore, enables an adversary to register any number of corrupt participants). In such cases the security of the protocol is not guaranteed.

These drawbacks prevent most applications from using the *generic* solutions for secure distributed computation.

2.5 Recommended reading

A preferred alternative to reading the original papers of secure computation is to read Ronald Cramer's lecture notes that provide an elementary introduction to the methods of secure computation [5], or Oded Goldreich's manuscript details a rigorous introduction to secure multi-party computation [9].

3. THE TWO-PARTY CASE: COMPUTING ID3

Yao's two-party protocol is pretty efficient, as long as the size of the inputs, and the size of the circuit computing the function, are reasonable. In fact, for many functions the efficiency of Yao's generic protocol is comparable to that of protocols that are targeted for computing the specific function. We describe here a distributed scenario of computing the ID3 algorithm, where Yao's protocol is obviously too costly. On the other hand, a specialized protocol can be designed for computing this algorithm, which uses Yao's protocol as a primitive.

[1]The only exception is the protocol of Beaver, Micali and Rogaway [1] that requires a constant number of communication rounds, but this protocol uses general zero-knowledge proofs that are inefficient.

Classification, decision trees and ID3. Classification is a classic problem in data mining, which is commonly solved using decision trees. ID3 is a basic algorithm for constructing decision trees. The input to a classification problem 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 (for example, each row could represent a patient, and each column a different symptom). One of the attributes in the database is designated as the *class* attribute (e.g., it could denote whether the patient has a certain disease). The goal is to use the database in order to predict the class of a new transaction by viewing only the non-class attributes.

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. 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 by traversing the nodes from the root down, following the edges that correspond to the attribute values of the transaction. The value of the leaf is the expected class value of the new transaction.

The ID3 algorithm is used to design a decision tree based on a given database. 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 defined below) is then chosen and the remaining transactions are partitioned by it. ID3 is then recursively called on each partition (which is a smaller database containing only the appropriate transactions and without the splitting attribute).

The central principle of ID3 is to choose the *best* predicting attribute based on information theory. The idea is to check which attribute reduces the information of the class-attribute to the greatest degree. Namely, to choose the attribute that provides the maximal information gain, where this value is defined as the difference between the entropy of the class attribute, and the entropy of the class attribute given the value of the chosen attribute. This decision rule results in a greedy algorithm that searches for a small decision tree consistent with the database. (Note that we only discuss the basic ID3 algorithm, and assume that each attribute is categorical and has a fixed set of possible values.)

Privacy preserving distributed computation of ID3. We are interested in a scenario involving two parties, each one of them holding a database of different transactions, where all the transactions have the same set of attributes (this scenario is also denoted as a "horizontally partitioned" database). The parties wish to compute a decision tree by applying the ID3 algorithm to the union of their databases. An efficient privacy preserving protocol for this problem was described in [12]. We describe its basic details below, and refer the readers to [12] for the complete solution.

Obstacles. A naive approach for implementing a privacy preserving solution is to apply the generic Yao protocol to the ID3 algorithm. This approach encounters two major obstacles. First, the size of the databases is typically very large. As each transaction can have many attributes, and there might be millions of transactions, the encoding of each party's input might require hundreds of millions of bits. This means that the computational overhead of running an oblivious transfer per input bit might be very high.

In addition, the circuit representation of ID3 is very large. Note that the basic step of the algorithm, which is repeated many times per node, involves computing the information gain, which is defined as the difference between two entropy values. Each entropy is computed as the sum of values of the form $p_i \log(p_i)$, where each p_i is the fraction of transactions in which the class attribute, and possibly other attributes, have certain values. This means that the protocol should compute the logarithm function, which is defined over the Real numbers. Most cryptographic protocols, however, compute functions over finite fields. Even if the circuit computes an approximation to the logarithm, this computation involves evaluating polynomials and therefore requires computing multiplications and exponentiations.

An additional problem is that running ID3 involves many rounds. The part of the circuit computing the ith round depends on the results of the previous $i-1$ rounds. A naive implementation could require an encoding of many copies of this step, each one of them corresponding to a specific result of the previous rounds.

Computing ID3. A key observation is that each node of the tree can be computed separately, with the output made public, before continuing to the next node. In general, private protocols have the property that intermediate values remain hidden. However, in the case of ID3 some of these intermediate values (specifically, the assignments of attributes to nodes) are actually part of the output and may therefore be revealed. Once the attribute of a given node has been found, both parties can separately partition their remaining transactions accordingly for the coming recursive calls. This means that private distributed ID3 can be reduced to privately finding the attribute with the highest information gain. (This is a slightly simplified argument as the other steps of ID3 must also be carefully dealt with. However, the main issues arise within this step.)

Computing information gains. Let T be a set of transactions. The exact test for determining the best attribute is defined as follows. Let $c_1, ..., c_\ell$ be the class-attribute values and let $T(c_i)$ denote 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 C be the class attribute and A be some 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 equals $H_C(T|A) = \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} H_C(T(a_j))$.

Now, for each attribute A the information-gain is defined as $\text{Gain}(A) = H_C(T) - H_C(T|A)$. The attribute A which has the maximum gain (or equivalently minimum $H_C(T|A)$ value) over all attributes is then chosen.

Notice that the algorithm needs only to find the name of the attribute A which minimizes $H_C(T|A)$; the actual value is irrelevant. Therefore, the coefficient $1/|T|$ can be ignored, and natural logarithms can be used instead of logarithms to base

2. Let T_A and T_B be the transactions in Alice's and Bob's databases, respectively. The values $|T_A(a_j)|$ and $|T_A(a_j, c_i)|$, which are a function of the first database alone, can be computed by Alice independently, and a similar argument holds for Bob. Therefore the value $H_C(T|A)$ can be written as a sum of expressions of the form $(v_A + v_B) \cdot \ln(v_A + v_B)$, where v_A is known to Alice and v_B is known to Bob (e.g., $v_A = |T_A(a_j)|$, $v_B = |T_B(a_j)|$).

The protocol computes the information gain of every attribute, such that at the end of the computation Alice and Bob hold two random shares, whose sum is equal to the information gain. None of the parties learns the information gains themselves, but they can later compare the sum of the different shares and find the attribute with the maximum gain. The main technical component of the protocol is a private computation of $x \ln x$ using a protocol that receives private inputs x_A and x_B such that $x_A + x_B = x$ and outputs random shares of an approximation of $x \ln x$. The shares are elements of a field $\mathcal{F}$ where it holds that $x \ln x < |\mathcal{F}|$.

The $x \ln x$ protocol. The protocol first computes random shares of $\ln x$. The first step is computing random shares of the values n and ε such that $x = 2^n(1 + \varepsilon)$ and $-1/2 \leq \varepsilon \leq 1/2$. Note that $\ln x = n \ln 2 + \ln(1 + \varepsilon)$. This computation is done using Yao's protocol, and is efficient since the circuit required for computing this function is very small (it basically sums the shares and checks for the location of the most significant bit of x).

The shares of the value ε that are output from the previous step are used to privately compute the Taylor series for $\ln(1 + \varepsilon)$, using the equation $\ln(1 + \varepsilon) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\varepsilon^i}{i} = \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3} - \frac{\varepsilon^4}{4} + \cdots$. This computation is done up to the power k that makes the approximation error sufficiently small. The parties therefore need to compute a polynomial of degree k and can do this using the oblivious polynomial evaluation primitive. All computations are done in $\mathcal{F}$. In order to avoid dealing with fractions, the parties compute the results multiplied by the least common multiple of $1, \ldots, k$ and therefore all intermediate values are integers (this means that the parties actually compute $\mathrm{lcm}(1, \ldots, k)x \ln x$, but this does not matter since they are only interested in *comparing* the entropies).

After obtaining shares of $\ln x$, the parties compute shares of $x \ln x$ using the oblivious polynomial evaluation primitive. To see how this is done, denote the shares of $\ln x$ as l_A and l_B. Therefore, $x \ln x = (x_A + x_B)(l_A + l_B) = x_A l_A + x_A l_B + x_B l_A + x_B l_B$. Alice can define two linear polynomials $P_1(y) = x_A y + r_1$ and $P_2(y) = l_A y + r_2$, where r_1, r_2 are random. Bob runs oblivious polynomial evaluation protocols to obtain $P_1(l_B)$ and $P_2(x_B)$, and sets his share to be $P_1(l_B) + P_2(x_B) + x_B l_B$. Alice sets her share to be $x_A l_A - r_1 - r_2$.

Finding the best attribute. Given shares for the different $x \ln x$ values, the parties should find the attribute with the best information gain. Each party first sums his or her shares that correspond to the same entropy value, in order to obtain a share of this entropy value. The parties then use Yao's protocol to compute the different information gain values and compare them, and output the index of the best attribute. Note that this is a simple circuit that has to perform one addition per attribute and then compare the results. Once the best attribute is found, the parties partition their databases according to the values of this attribute and run the algorithm recursively.

Note that this protocol actually does not compute the result of ID3, but rather an approximation: If the difference between two information gain values is smaller than the effect of the approximation errors that are generated by the Taylor approximations, then the protocol might choose either one of the two attributes as the best attribute.

4. THE MULTI-PARTY CASE

The multi-party case involves three or more parties that wish to compute some function of their inputs without leaking any unnecessary information. As we have described above, there are generic constructions for this task [10; 3; 4]. Compared to the two-party case, however, it is harder to apply the generic constructions to actual scenarios. To illustrate this point we consider the case of running a secure computation for computing the result of an auction, where there is an obvious motivation for privacy and security, and also certain restrictions on the operation of the parties. The auction application, discussed in [16], is not related to data mining, but it does exemplify some of the difficulties of the multi-party case. The discussion below applies for any function that can be computed by a circuit of reasonable size.

The auction scenario is that of a "sealed bid" auction, and consists of an auctioneer and many bidders. Each bidder submits a single secret bid (i.e. the bid is sealed in an envelope). There is a known decision rule, whose inputs are the submitted bids, and whose output is the identity of the winning bidder and the amount that this bidder has to pay. For example, in an "English auction" the winning bidder is the bidder who offered the highest bid, and he has to pay the amount of his bid. In the second-price, or Vickrey, type of auction (which has some nice properties that are outside the scope of this paper) the winner is the highest bidder and he has to pay the amount of the *second highest* bid. Bidding is allowed until some point in time, and at that stage the decision rule is applied to the submitted bids.

In the physical world bids are submitted in sealed envelopes that are kept secure until the end of the bidding period, and are then opened by the auctioneer. In the virtual world we would like to keep the bids secret during the bidding period, but we could also attempt to hide all information *afterwards*, except for the identity of the winning party and the amount he has to pay. For example, in the case of a Vickrey auction the auctioneer's output could be limited to the identity of the highest bidder (but not the value of his bid), and the value of the second highest bid (but not the identity of the second highest bidder). This is more privacy than can be achieved in the physical world. (In fact, some of the suggested explanations for the unpopularity of second-price auctions are based on possible attacks that a malicious auctioneer can mount if he learns the bid value of the highest bidder. This phenomenon is inevitable in the real world, but can be avoided if a privacy preserving protocol is used to compute the result of the auction.)

The difficulty of applying the generic constructions. A natural approach for achieving privacy is to run a secure multi-party computation involving the auctioneer and all bidders, where the inputs are the respective bids of the par-

ties, and the output is the result of the auction. This approach seems promising in the auction scenario since the circuit that computes the result of the auction is typically small, as it involves only comparisons.
The drawbacks of the generic solutions for the multi-scenario case, which we have described in Section 2.4, become apparent when we consider applying these constructions to auctions. The computation overhead per gate is high, the protocol requires each pair of bidders to exchange several rounds of messages (which might be unacceptable, for example, in an Internet environment where different bidders do not have mutual relationships, and are not even online at the same times), and security and privacy are only assured if less than, say, one half of the parties collude (whereas in an environment where there are no long term trust relationships between the parties an adversary could register the majority of the bidders that participate in the auction).

Applying the two-party solution in the multi-party scenario. Privacy preserving multi-party computation can be reduced to the two-party case. Namely, it is possible to use the generic two-party protocol to compute a function in the multi-party scenario. Such a reduction is described in [16]. Before describing the highlights of the reduction we first describe the advantages of this approach.

- *Trust:* In order to use the two-party construction it is assumed that there are two special parties, and privacy is preserved as long as these two parties do not collude. Namely, a collusion of any number of parties (even a majority of the parties) that does not include *both* special parties does not affect the privacy and security of the protocol.

  Protocols with this security assurance might seem weaker than protocols that are secure against collusions of say, any coalition of less than one half of the parties. After all, there is a coalition of just two parties – the two special parties, is able to break the security of the system. Consider however a scenario where most of the parties are users (e.g. bidders) that have not established trust relationships between themselves, and there are one or more central parties that are more established. For example, in the auction scenario we can assume that the two special parties are the auctioneer and another party which we denote as the "issuer", and which can be, for example, an accounting firm. We know that an adversary can register many fake bidders in order to control a majority of the participating parties. It seems harder, though, for the adversary to be able to control insiders of *both* special parties, i.e. in the auctioneer's organization and in the accounting firm.

- *Communication:* We can design the reduction such that each of the "simple" participating parties should only communicate with one of the special parties (e.g. the auctioneer), and should only send a single message to this party. This property greatly simplifies the required communication infrastructure, and enables to run the protocol without requiring all parties to be online at the same time (in fact, compared to a protocol that provides no security at all, the only new communication channel that is introduced by the secure protocol is the channel between the two special parties). When all the "simple" parties finish sending their messages, the two special parties run a short protocol to complete the computation of the function.

- Efficiency: The protocol evaluates a circuit representation of the function. The overhead per gate and per input bit is as in the two-party construction, and is lower than in the multi-party constructions.

The protocol is run with the two special parties taking the roles of the two parties in the two-party case. The issuer prepares a circuit for computing the function. This circuit might have many inputs of different parties – for example, the inputs might be the bids of the different bidders. The issuer encodes the circuit as in the two-party case, by choosing garbled values for the wires and preparing tables for every gate. The other special party (the auctioneer) is responsible for computing the result of the circuit. In order to do that it should receive the tables that were prepared by the issuer, and one garbled value for every input wire, namely the value that corresponds to the input bit associated with that wire. Once it receives the garbled values of all input wires it can compute the output of the circuit.
Note that the inputs are, and should remain, unknown to the auctioneer, yet it should be able to obtain the correct garbled value for each input wire. In order to do that the parties run a protocol called "proxy oblivious transfer", which was introduced in [16]. This protocol is similar to oblivious transfer, but involves three parties: the chooser (which is the party who knows the value of the input wire), the sender (which is the issuer), and the receiver (the auctioneer). The input of the chooser is a bit σ, and the input of the sender consists of two items x_0, x_1. At the end of the protocol the receiver should learn x_σ, and no information about $x_{1-\sigma}$ or σ, and the other parties should learn nothing. An implementation of this protocol, which has an overhead comparable to that of plain oblivious transfer, and does not require direct communication between the chooser and the sender, is described in [16].
Given the proxy oblivious transfer protocol, the rest of the implementation is simple. Each bidder engages in a proxy oblivious transfer for each of its input bits. The input of the bidder to this protocol is the value of the input bit. The sender is the issuer, and its two inputs are the two garbled values that are associated with the corresponding input wire. The receiver is the auctioneer, and it learns the garbled value that corresponds to the input bit. This protocol consists of a single message that is sent from the bidder to the auctioneer, and then a round of communication between the auctioneer and the issuer. The auctioneer can actually wait until it receives messages from all the bidders before it runs the round of communication with the issuer in parallel for all input bits. The main computational overhead of the protocol is incurred by the proxy oblivious transfers, and is the same as in the two-party case – a proxy oblivious transfer must be executed for every input wire. Estimates in [16] show that this method can be used to securely implement Vickrey auctions that involve hundreds of bidders.

5. CONCLUSIONS

This paper was intended to demonstrate basic ideas from a large body of cryptographic research on secure distributed computation, and their applications to data mining. We

described in brief the definitions of security, and the generic constructions for the two-party and multi-party scenarios. We showed that it is easier to design an implementation based on the constructions for the two-party case than it is to design one based on the multi-party constructions. The main parameter that affects the feasibility of implementing a secure protocol based on the generic constructions is the size of the best combinatorial circuit that computes the function that is evaluated. The main computational bottleneck of the constructions is the oblivious transfer protocol, and any improvement in the overhead of this protocol should directly affect the overhead of secure computation.

6. REFERENCES

[1] D. Beaver, S. Micali and P. Rogaway, *The round complexity of secure protocols*, Proc. of 22nd ACM Symposium on Theory of Computing (STOC), pp. 503-513, 1990.

[2] M. Bellare and S. Micali, *Non-Interactive Oblivious Transfer and Applications*, *Advances in Cryptology - CRYPTO '89.* Lecture Notes in Computer Science, Vol. 435, Springer-Verlag, 1997, pp. 547-557.

[3] M. Ben-Or, S. Goldwasser and A. Wigderson, *Completeness theorems for non cryptographic fault tolerant distributed computation*, Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC), ACM, 1988, pp. 1–9.

[4] D. Chaum, C. Crepeau and I. Damgard, *Multiparty unconditionally secure protocols*, Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC), ACM, 1988, pp. 11–19.

[5] R. Cramer, *Introduction to Secure Computation*, 2000. Available at `http://www.brics.dk/~cramer/papers/CRAMER_revised.ps`.

[6] Wei Dai, *The Crypto++ library*, benchmark of Nov. 3, 2002, `http://www.eskimo.com/ weidai/cryptlib.html`.

[7] S. Even, O. Goldreich and A. Lempel, *A Randomized Protocol for Signing Contracts*, Communications of the ACM, vol. **28**, 1985, pp. 637–647.

[8] R. Fagin, M. Naor and P. Winkler, *Comparing Information Without Leaking It*, Communications of the ACM, 39(5), pp. 77-85, 1996.

[9] O. Goldreich, Secure Multi-Party Computation, manuscript, 2002. Available at `http://www.wisdom.weizmann.ac.il/ oded/pp.html`.

[10] O. Goldreich, S. Micali and A. Wigderson, *How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority*, Proceedings of the 19th Annual Symposium on the Theory of Computing (STOC), ACM, 1987, pp. 218–229.

[11] J. Kilian, *Founding cryptography on oblivious transfer*, ACM STOC '88, pp. 20-31.

[12] Y. Lindell and B. Pinkas, *Privacy Preserving Data Mining*, Journal of Cryptology, Vol. 15, No. 3, pp. 177-206, 2002.

[13] M. Luby, **Pseudorandomness and Cryptographic Applications**, Princeton Computer Science Notes, 1996.

[14] M. Naor and B. Pinkas, *Oblivious Transfer and Polynomial Evaluation*, Proceedings of the 31th Annual Symposium on the Theory of Computing (STOC), ACM, 1999, pp. 245–254.

[15] M. Naor and B. Pinkas, *Efficient Oblivious Transfer Protocols*, Proceedings of 12th SIAM Symposium on Discrete Algorithms (SODA), January 7-9 2001, Washington DC, pp. 448–457.

[16] M. Naor, B. Pinkas and R. Sumner, *Privacy Preserving Auctions and Mechanism Design*, Proc. of the 1st ACM conference on Electronic Commerce, November 1999.

[17] M. O. Rabin, *How to exchange secrets by oblivious transfer*, Technical Memo TR-81, Aiken Computation Laboratory, 1981.

[18] J.E. Savage, *Computational work and time on finite machines*, Journal of the ACM, 19(4), pp. 660-674, 1972.

[19] A. C. Yao, *How to generate and exchange secrets*, Proceedings 27th Symposium on Foundations of Computer Science (FOCS), IEEE, 1986, pp. 162–167.