Secure set intersection cardinality with application to association rule mining

by Christopher Clifton
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Accepted for publication in Journal of Computer Security.

Secure Set Intersection Cardinality with Application to Association Rule Mining

Jaideep Vaidya
+1-765-494-6408
jsvaidya@cs.purdue.edu

Chris Clifton
+1-765-494-6005
clifton@cs.purdue.edu

Department of Computer Sciences
Purdue University
250 N University St
West Lafayette, IN 47907-2066

March 15, 2004

Abstract

There has been concern over the apparent conflict between privacy and data mining. There is no inherent conflict, as most types of data mining produce summary results that do not reveal information about individuals. The process of data mining may use private data, leading to the potential for privacy breaches. Secure Multiparty Computation shows that results can be produced without revealing the data used to generate them. The problem is that general techniques for secure multiparty computation do not scale to data-mining size computations. This paper presents an efficient protocol for securely determining the size of set intersection, and shows how this can be used to generate association rules where multiple parties have different (and private) information about the same set of individuals.

1 Introduction

Privacy advocates have been challenging attempts to bring more and more information into integrated collections. Attempts to combine data have even resulted in public protest, witness Japan's creation of a national registry containing information previously held by the prefectures[29]. Data mining in particular has come under siege, such as the introduction of U.S. Senate Bill 188, the "Data-Mining Moratorium Act of 2003"[15]. While aimed specifically at the Total Information Awareness program[30], the bill as introduced would have forbidden data-mining (including research and development) by the entire U.S. Department of Defense, except for searches of public information or searches based on particular suspicion of an individual. In addition, all U.S. government agencies would have been required to report to congress on how their data-mining activities protect individual privacy.

The irony is that most data mining techniques generate high-level rules or summaries – the objective is to generalize across populations, rather than reveal information about individuals. The problem is that data mining works by evaluating individual data that is subject to privacy concerns.

Much of this information has already been collected, however it is held by various organizations. Separation of control and individual safeguards prevent correlation of this information, providing acceptable privacy in practice. However, this separation also makes it difficult to use the information for purposes that would benefit society, such as identifying criminal activity. Proposals to share information across agencies, most recently to combat terrorism, would eliminate the safeguards imposed by separation of the information.

This problem is not insurmountable, provided we make the following two assumptions:

1. Putting different information under the control of different entities provides sufficient privacy safeguards, and

2. The desired results to be computed from the combined information do not by themselves violate privacy.

The first assumption is already being made, e.g., the TIA program spoke of making better use of existing information rather than new data collection. The second assumption is reasonable for a wide class of computations. Yao showed how a function could be computed without revealing the inputs [33], this has grown into the field of Secure Multiparty Computation [16]. Their work showed that it is possible to address privacy issues; what remains is to show that practical and efficient techniques exist.

This paper presents an efficient and provably secure method for computing the size of the intersection of sets of items held by different parties. We also show how this method can be used to compute association rules in an environment where different information holders have different types of information about a common set of entities, e.g., the multiple government agency problem described above.

1.1 Problem Definition and Method Overview

Consider several parties having 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 k parties $P_1, \ldots, P_k$ having local sets $S_1, \ldots, S_k$, we wish to securely compute $|S_1 \cap \ldots \cap S_k|$.

The basic idea is to use a parametric commutative one way hash function to encrypt all items. (Commutative public key encryption produces one such hash function.) Each party encrypts all their items with their own key. Each set is passed to the next party, who encrypts the items and passes the set on until all parties have encrypted all items. Since encryption is commutative, the encrypted values from 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. The encryption prevents any party from knowing which of the items are present in the intersection.

There are additional complexities required to achieve a protocol with provable security properties, details and the proof are given in Section 3. Clever ordering of operations allows an algorithm with asymptotically improved performance, this is described in Section 4. In Section 5, we show how association rules can be securely computed using the set intersection algorithm, and compare this with other proposals for privately computing association rules. We will begin with a discussion of relevant background from the fields of Secure Multiparty Computation and data mining.

2 Background and Related Work

The general problem of distributed computation has been to compute a function f where the argument data resides at several sites, with low communication cost and high parallelism relative to computing f if all data were centralized. This ignores the issue of *why* the data is distributed, often distribution results from autonomy of data sources. The autonomous sources often have privacy or secrecy concerns preventing arbitrary sharing of data, even if all sources agree that they are willing to share the result of the global function f.

2.1 Secure Multiparty Computation

There has been work in cooperative computation between entities that mutually distrust one another. Secure two party computation was first investigated by Yao [33] and was later generalized to multiparty computation. The seminal paper by Goldreich proves that there exists a secure solution for *any* functionality[16]. The approach used is as follows: the function f to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. Every participant gets (randomly chosen) shares of the values of the input and output for each gate; the exclusive or of the shares is the actual value. One share alone carries no information about the input or function value, as which party gets which share is determined randomly. At the end, the parties exchange their shares, enabling each to compute the final result. This protocol has been proven to give the desired result without disclosing anything other than the result. The work in [16] establishes what is meant by secure; we will give necessary definitions later as part of the proof of security of our set intersection protocol. This approach, though appealing in its generality and simplicity, means that the size of the protocol depends on the size of the circuit, which depends on the size of the input.

The cost of circuit evaluation for large inputs has resulted in several algorithms for more efficiently computing specific functionalities. Examples include secure summation[28], scalar product[6, 18, 31], matrix operations[11], secure set union[21], private permutation[12], and computing entropy[13]. This paper adds a new operation to this toolkit that is particularly useful for operations on vertically partitioned data, i.e., data where different attributes of a single entity are held at different sites.

2.2 Data Mining

Data mining is a rapidly growing field, providing new methods to analyze large data sets. A classic problem is mining association rules from transaction databases. A transaction is a set of items, e.g., items purchased in a store visit. A rule is of the form $AB \Rightarrow C$, where A, B, and C correspond to items in the store. Rules are deemed interesting when they occur in many transactions (support), and where transactions that contain the left hand side are likely to contain the right hand side (confidence). The association rule mining problem is to find all such rules, i.e., all A, B, and C that form rules with the desired confidence and support. The key component is to find sets of items that occur frequently, from these the rules can be determined.

Cheung et al. proposed a method for horizontally partitioned data[10], i.e., data where different sites contain different transactions. Distributed solutions have been suggested for several other types of data mining as well[27, 7, 8, 25, 9, 32]. However, none of this work addresses the privacy and security concerns of the data sources.

There has been interest in addressing privacy and security issues in the data mining process. One approach is to add "noise" to the data before the data mining process, and using techniques that mitigate the impact of the noise on the data mining results[5, 1, 14, 26]. This gives a good approximation, but assuming the data sources are willing to cooperate to obtain the data mining result allows us to obtain exact results and maintain higher standards of privacy.

The first work on privacy preserving data mining from distributed sources addressed the construction of decision trees[23]. This work closely followed the secure multiparty computation approach, trading off computation and communication cost for accuracy while achieving "perfect" privacy There has since been work to address association rules in horizontally partitioned data[21, 19], EM Clustering in Horizontally Partitioned Data[22], association rules in vertically partitioned data for two parties[31], and generalized approaches to reducing the number of "on-line" parties required for computation[20]. Some of this work makes tradeoffs between efficiency and information disclosure, however all maintain provable privacy of individual information and bounds on disclosure, and disclosure is limited to information that is unlikely to be of practical concern.

The method for computing association rules of [31] works only for two parties, we now present a method for more than two. In addition, [31] makes use of secure scalar product: an item is represented as a vector with a 1 for each transaction containing the item and a 0 for transactions not containing the item. The scalar product of a set of vectors gives the support (frequency) of the corresponding set of items. In addition to the method of [31], secure scalar product protocols have been presented in [6, 18]. Their use for association rule mining shares a common failing: They are subject to *probing* attacks. Probing attacks involve one party generating false input to find out something about the other party's data from the output. For scalar product, generating a vector with a single 1 enables one party to find out the value of the corresponding element in the other party's vector. Ordinarily, a protocol can be aborted when one party is observed sending spurious input. However, with secure evaluation none of the other parties should find out what the input of any other party is. This makes it difficult to detect probing. We will demonstrate how the secure size of set intersection protocols presented in this paper can be used to compute association rules in a way that is both safe from probing attacks and provably avoids disclosing input.

3 Securely Computing the Size of Set Intersection

Assume $k > 2$ parties, $P_0, \dots, P_{k-1}$. Each party P_i has set $S_i \subseteq U$ chosen from a common global universe. The problem is to securely compute the size of the intersection set, $| \cap_{i=0}^{k-1} S_i|$.

3.1 Algorithm Idea

The key idea behind the algorithm is simple. It is not necessary to have the actual set elements to compute the cardinality of the intersection set. Instead, the parties jointly generate a mapping from U that no party knows in its entirety. The mapping is used to transform the sets S_i, then the intersection is performed on the transformed sets. Since no party knows the mapping, they cannot reverse the mapping to find the value of any element.

A secure keyed commutative hash function can be used to perform such a mapping, and has other properties that will be useful in proving the security properties of the algorithm. We now formally define such a hash function.

3.2 Commutative One-way Hash Functions

The definitions of properties used below are collated from [24].

Definition 3.1 *A* commutative keyed one way hash function *(CKHF) is a function h_k, parameterized by a secret key k, with the following properties:*

1. ease of computation - *for a known function h_k, given a value k and an input x, it is easy to compute $h_k(x)$.*

2. 2nd-preimage resistance - *it is computationally infeasible to find a second input that has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$.*

3. collision resistance - *it is computationally infeasible to find any two distinct inputs x, x' that hash to the same output, i.e., $h(x) = h(x')$. A stronger form of collision resistance is to require $\forall x \neq x', h(x) \neq h(x')$.*

4. commutative hashing - *given two instances of a keyed hash function h_k parameterized with 2 different keys k_1 and k_2 and an input x, $h_{k_1}(h_{k_2}(x)) = h_{k_2}(h_{k_1}(x))$.*

5. key non-recovery - *given one or more text-hash pairs $(x_i, h_k(x_i))$ for the same k, it must be computationally infeasible to recover k.*

A commutative public key encryption scheme such as Pohlig-Hellman can be used to generate a hash function satisfying all our requirements. Each party generates its own keypair (E_i, D_i). The length in bits for the keypair is commonly agreed upon and known to all parties (1024 bits is common today). The hash function h_{k_i} is simply encryption with E_i. The decryption keys are not needed.

3.3 Algorithm

There are three stages to the protocol: hashing, initial intersection, and final intersection. In the hashing stage, each party hashes (encrypts) its own items with its own key. The parties then pass the set to their neighbor to be hashed. This continues until all sets have been encrypted by all keys. Since hashing is commutative, the resulting values from two different sets will be equal if the original values were the same (i.e., the item was present in both sets). Collision resistance ensures that this will happen only if the original values were the same.

In the initial intersection stage, each party sends the set it has to all parties except its right neighbor, the original owner of the set. All parties then compute the intersection of all the sets received. If the size of this intersection is less than a user-determined threshold, the protocol is aborted. This is to avoid probing attacks. In the final stage, the intersections are sent to the right neighbor. The final result is the size of the intersection of the received set with the one generated in the initial intersection stage. A complete description of the algorithm is given as Protocol 1.

We now give additional clarification of the algorithm.

Hashing In this stage the sets of all the parties are hashed by all parties. Since each party hashes with a key known only to itself, and the order of items is randomly permuted, no other party can determine the mapping performed by the previous party.

Protocol 1 Securely computing size of intersection set

```
Require: $k > 2$ sites, each having a local set $S_i$
Require: Maximum local set size $m$, and threshold $r$ used to protect against probing
  for all sites $i$ {Parallel operations} do
    Generate the hash key $E_i$
    for $j = |S_i|$ to $m$ do
      $S_i \leftarrow S_i \cup \{\text{prefix\_not\_in\_}U.i.j\}$ {Pad $S_i$ with items that will be unique to that site and cannot contribute to the intersection.}
    end for

    {Stage 1 - Hashing}
    $M \leftarrow EncryptAndPermute(S_i, E_i)$
    send $M$ to site $i + 1 \bmod k$

    for $p = 1 \ldots k - 2$ do
      $M' \leftarrow$ receive from site $i - 1 \bmod k$
      $M'' \leftarrow EncryptAndPermute(M', E_i)$
      send $M''$ to site $i + 1 \bmod k$
    end for

    $M' \leftarrow$ receive from site $i - 1 \bmod k$
    $M'' \leftarrow EncryptAndPermute(M', E_i)$
    send $M''$ to all sites except site $i + 1 \bmod k$

    {Stage 2 - Initial Intersection of sets and check for probing}
    $TS_j \leftarrow$ receive from site $j$, $j \neq i - 1$
    $TS'_i \leftarrow \cap_{p=0, p \neq i-1}^{k-1} TS_p$
    if $|TS'_i| < r$ then
      broadcast ABORT {Detect/prevent probing}
    else

      {Stage 3 - Final Intersection to compute Final Result}
      Send $TS'_i$ to party $i + 1 \bmod k$
      Receive $TS'_{i-1 \bmod k}$ from party $i - 1 \pmod k$
      $TS'_i \leftarrow TS'_i \cap TS'_{i-1 \bmod k}$
      return $|TS'_i|$
    end if
  end for
```

```
Function EncryptAndPermute(Set M, Key E_k)
Require: $M$ is the input array to be hashed, $E_k$ is the hash key
  $C \leftarrow \emptyset$
  for all $j \in M$ do
    $C \leftarrow C \cup \{E_k(j)\}$ ;
  end for
  randomly permute $C$ to prevent tracking values
  return $C$
```

Initial Intersection In this stage, every party finds the intersection of all sets except its own. The hashing prevents learning the actual values corresponding to the hashed items received. The reason a site does not get its own set is to prevent probing attacks: a site could initially generate a singleton set to probe if that item existed at another site, i.e., if the intersection of its set with that of another site is empty or of size 1. Aborting prevents probes for sets of less than size r.

This also shows the reason that we require $k > 2$ parties. With two parties, no intersection could be performed without access to the hashed values of one's own set. This prevents the probe detection/prevention.

Final Intersection Each party sends the remaining piece of the puzzle to its left neighbor. This enables all parties to compute the final intersection and find the final result, viz. the cardinality of the total intersection set.

The collision resistance property of the hash function ensures that no collisions can occur. Thus the algorithm clearly generates the correct result for the size of the intersection set.

3.4 Proof of Security

Before proving the security of the protocol, we must state what we mean by *secure*. The definitions we use come from the field of Secure Multi-party Computation, specifically [16].

3.4.1 Definition of Secure Multi-party Computation

We start with the semi-honest model. A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security.

A formal definition of private two party computation in the semi-honest model is given below. Computing a function privately is equivalent to computing it securely. The formal proof of this can be found in [17].

Definition 3.2 *(privacy w.r.t. semi-honest behavior):[17]*

Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ *be a probabilistic, polynomial-time functionality, where* $f_1(x,y)$*(resp.,* $f_2(x,y)$ *denotes the first (resp., second) element of* $f(x,y)$*). Let* Π *be two-party protocol for computing* f*.*

The view *of the first (resp. second) party during an execution of* Π *on* (x,y)*, denoted* $VIEW_1^{\Pi}(x,y)$ *(resp.,* $VIEW_2^{\Pi}(x,y)$*) is* $(x, r, m_1, \ldots, m_t)$ *(resp.,* $(y, r, m_1, \ldots, m_t)$*), where* r *represent the out-*

come of the first (resp., second) party's internal coin tosses, and m_i represents the i^{th} message it has received.

The output *of the first (resp., second) party during an execution of Π on (x, y), denoted $OUTPUT_1^{\Pi}(x, y)$ (resp., $OUTPUT_2^{\Pi}(x, y)$), is implicit in the party's own view of the execution.*

Π privately computes f if there exist probabilistic polynomial time algorithms S_1 and S_2 such that

$$\{(S_1(x, f_1(x, y)), f_2(x, y))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(VIEW_1^{\Pi}(x, y), OUTPUT_2^{\Pi}(x, y)\right)\right\}_{x,y\in\{0,1\}^*}$$

$$\{(f_1(x, y), S_2(x, f_1(x, y)))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(OUTPUT_1^{\Pi}(x, y), VIEW_2^{\Pi}(x, y)\right)\right\}_{x,y\in\{0,1\}^*}$$

where $\equiv^C$ denotes computational indistinguishability.

Privacy by Simulation The above definition says that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated given the input and the output of that party. Thus, in all of our proofs of security, we only need to show the existence of a simulator for each party that satisfies the above equations.

This does not quite guarantee that private information is protected. Whatever information can be deduced from the final result obviously cannot be kept private. For example, if an association rule $A \Rightarrow B$ is supported by three entities, and A only holds for three entities, it is clear that B holds for those same three entities. The result reveals information to the site having A. The key to this definition is that nothing *beyond* the results is learned.

In summary, secure multi-party protocol will not reveal more information to a particular party than the information that can be induced by looking at that party's input and the final output.

3.4.2 Security Analysis

The intersection algorithm described above clearly calculates the size of the intersection set without revealing what items are present in any set. However, it is not quite secure based on the standard of Definition 3.2. In addition to the size of the complete set intersection, the parties can learn the size of the intersection of subsets of the entire group. Specifically, for any set $C \subseteq \{0, \ldots, k-1\}^*$ such that $i \notin C, |C| \geq 2$, party i can compute $|\cap_{j\in C} S_j|$.

By acknowledging that these subset intersection sizes are revealed, we can prove that nothing else is disclosed by Protocol 1. We effectively augment the result with the "leaked information". We then show how to build a polynomial time simulator for the view of every party using only the augmented output for that party and their own input.

Theorem 3.1 *Protocol 1 privately computes the size of the intersection set $|S| = |\cap_{p=0}^{k-1} S_p|$. Site i learns at most $|\cap_{p\in C} S_p|, \forall C \subseteq \{0, \ldots, k-1\}^*$ such that $i \notin C, |C| \geq 2$. If $|\cap_{p=0,\ldots,i-2,i,\ldots,k-1} S_p| \geq r$ it learns that value and the final result $|S|$.*

Proof. Since the protocol is symmetric, proving that the view of one party can be simulated with its own input and output suffices to prove it for all parties. We now show how to simulate the messages received by party i. Given these, it uses its own input and hash key to simulate the rest of its view by running the appropriate parts of Protocol 1. The protocol consists of three stages. The initialization phase can be done based on i's own input, so we begin with the messages received in Stage 1.

Stage 1 At each step of this stage, party i receives a new local set from part $i-1 \bmod k$. However, each item in each of these sets has been hashed (encrypted). The preimage resistance, collision resistance, and key non-recovery properties combine to ensure that the distribution of the hashed values (as the key changes) is independent of the distribution of the data. This allows us to state that the values in these sets are computationally indistinguishable from numbers chosen from a uniform random distribution. We can simulate the received set M' by randomly choosing m values uniformly distributed over the domain of the hash function E.

This allows us to simulate a single M'. Each M' seen by i is hashed by a different set of keys, i.e., the first is hashed with $E_{i-1}(x)$, the second by $E_{i-1}(E_{i-2}(x))$, etc. (For brevity we drop the $\bmod k$, it should be assumed in all index computations.) Regardless of any relationship between items in S_{i-1} and S_{i-2}, the different hashes and permutation ensure that i sees no relationship. Therefore, the argument that randomly choosing values allows us to simulate M' extends to the set of all $k-1$ M' sets.

Stage 2 Party i now receives $k-1$ sets TS_j, corresponding to the fully hashed sets from all but party i. TS_i is the last M'' generated in Stage 1, and is simulated with the final M'' generated in the simulation of Stage 1. While the hashing/encryption guarantees that any single item in these sets is equally likely to come from anywhere in the domain of E, we can not simply generate random values for the other TS_j. The problem is the need to simulate intersections. Since all parties have hashed all items, and because the hashing is commutative, if S_g and S_h have an item in common, then TS_{g-1} and TS_{h-1} will also have an item in common.

To simulate this, we take advantage of knowing $|\cap_{p \in C} S_p|$. The simulator first generates a directed acyclic graph from these intersection sizes, and uses this to calculate the number of items that should be common at each node. It then does a breadth-first traversal, generating the required number of items at each node and placing the items in the leaf sets reachable from that node. "Generating" an item happens in two ways: When TS_i is one of the reachable leaves, an item is chosen (without replacement) from TS_i. Otherwise, a random value from the domain of E is used.

A detailed description of this process is given as Simulator 1. A demonstration of the simulation algorithm for three parties is given in Figure 1.

Party i can now generate TS'_i and determine if it should send an ABORT message. It also knows if it should receive an ABORT, as this is part of the result. If the result is not an ABORT, we must simulate Stage 3.

Stage 3 Party i now receives TS'_{i-1}. Since $|TS'_{i-1}| \geq r$, i is allowed to learn the size of this set (i is not probing). This set is simulated by choosing $|S|$ items from TS'_i, and randomly choosing $|TS'_i| - |S|$ from the domain of E. The encryption arguments used for Stage 1 still hold to protect the *value* of the items, and the known "leaked" information is sufficient to perform the simulation.

Definition 3.2 requires that this simulation be polynomial time. Stages 1 and 3 are clearly polynomial. Stage 2 requires construction and breadth-first traversal of a graph consisting of all powersets of $k-1$ nodes. The graph is exponential in k, an apparent conflict. However, the requirement is that the simulation be polynomial in the size of the input m, we can treat k as fixed. In the graph traversal of Algorithm 1, we generate $k * m$ items to fill the leaves. Since generating each item (either choosing an item from TS_i or randomly generating one) is polynomial, and we perform one such operation for each item in the input, the simulation is polynomial. □

Simulator 1 GenInput: Generating Input Sets for Party i

Generate the hierarchical directed graph G connecting all of the intersection sets to their immediate descendents.

- $\{0, \ldots, i-1, i+1, \ldots, k-1\}$ is the root,
- All sets with $k-2$ elements are level 2,
- $\ldots$
- All 2-sets are at level $k-2$,
- $\{1\}, \ldots, \{i-1\}, \{i+1\}, \ldots, \{k-1\}$ (i.e. sets for all parties other than i) are leaves at level $k-1$,
- An edge is added from p to c if c is a subset of the set represented by p obtained by removing one number.

Each node n is assigned the size of the corresponding intersection set $|\cap_{p \in n} S_p|$, nodes at level $k-1$ are assigned m.

```
for $l = 1..k-1$ do
  for all nodes $p$ at level $l$ do
    if $i \in p$ then
      $items \leftarrow$ remove $p.size$ items from $M''$
    else
      $items \leftarrow$ remove $p.size$ items from (domain of $E - M''$)
    end if
    for all $TS_j, j \in p$ do
      $TS_j \leftarrow TS_j \cup items$
    end for
    for all $c$ child of $n$ do
      $c.size \leftarrow c.size - p.size$
    end for
  end for
end for
```

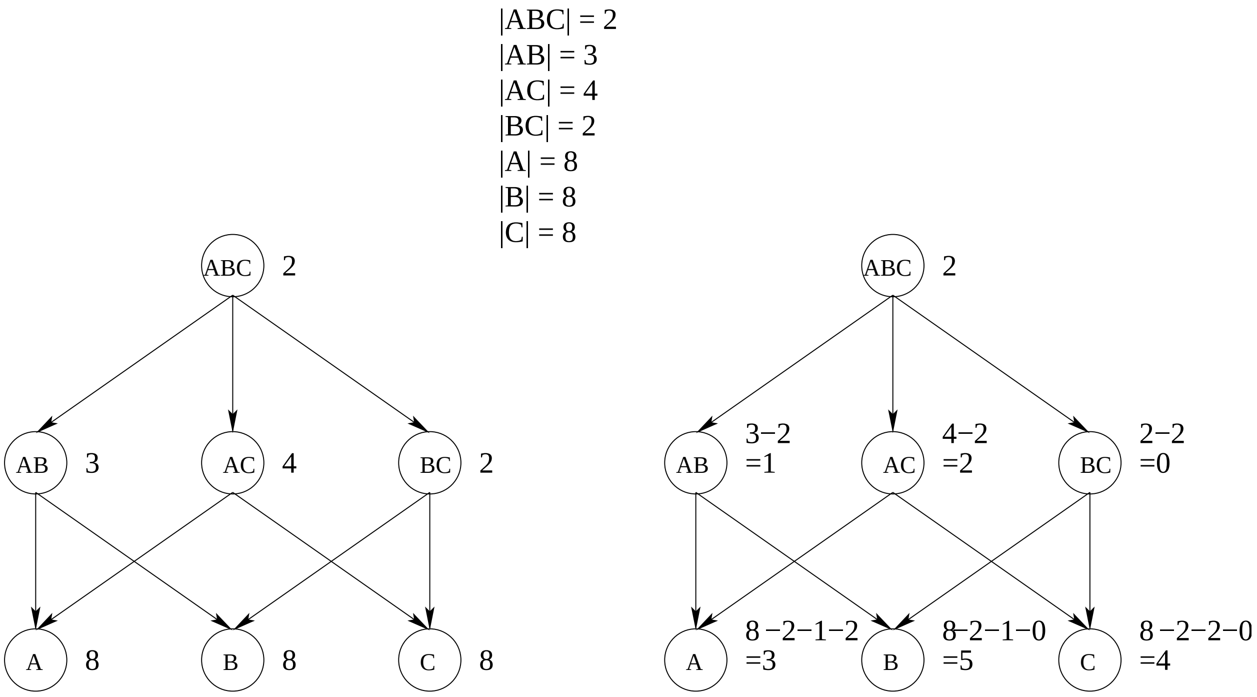


Figure 1: Building the simulated input set.

The definitions we have given are for the semi-honest model: parties follow the protocol, but may try to learn additional details from what they see during the execution. The malicious model for Secure Multiparty Computation looks at the case where parties may not play by the rules. Protocol 1 does not quite meet malicious standards, as a malicious party can cause incorrect results without detection. From the proof of Theorem 3.1 we can see that the disclosure properties *do* hold in the face of a malicious party. No party sees information hashed with the same set of keys twice, so altering an outgoing message to learn how it is hashed would not enable learning anything from an incoming message. This is true as long as there is no collusion between parties. However, if two parties collude, they could jointly mount a probing attack by returning each party's fully hashed items to that party.

3.5 Cost of Protocol 1

Communication protocols of this type are generally analyzed either based on the communication cost or number of encryptions performed. The encryption cost is entirely contained in Stage 1: Each party hashes every item once, giving $k^2 * m$ encryptions. The inherent parallelism gives a factor of k speedup, for a computation time cost of $k * m$.

The communication cost for a single party is as follows. In stage 1, a set of size m is transmitted in each of the first $k-1$ rounds. In round k, a set is sent to $k-2$ of the other parties. Assuming no multicast this requires $2k-3$ messages of $m * \textit{hashed item size}$ bits. Stage 2 requires no transmission (except possibly a broadcast ABORT). In Stage 3, a message is sent containing the intersection of all but one party. This message is at most size m hashed items, but would typically be closer to the lower bound of $|S|$ items. Thus, neglecting abort, each site sends $2k-2$ message of at most size $m * \textit{hashed item size}$ bits.

The entire protocol is symmetric, so all of the parties transmit equal amounts of data. So, to calculate the total communication cost, we simply multiply the single party cost by k. Thus, the upper bound on the total communication cost of the algorithm is

$$\begin{array}{rl} k*(2k-2) = O(k^2) & \text{messages} \\ k*(2k-2)*m*\textit{hashed item size} = O(k^2m) & \text{bits} \\ k & \text{rounds} \end{array}$$

The factor of m can be reduced to $|S_i|$, at the cost of revealing the size of each site's set. This is done by skipping the padding to size m step at the beginning of Protocol 1. As this is much more likely to be sensitive information than the sizes of intersections, we have detailed the more secure version.

Although this cost may seem high, the problem inherently requires $O(k * |S_i|)$ worst-case communication even if security is not an issue. An obvious non-secure solution would be for all the sites to send their sets to a third party that would compute and return the result, requiring transfer of all of the sets once. It is impossible to determine if an item is in the global intersection based purely on local information, preventing asymptotic worst-case improvements on this simple solution. Our protocol has an additional multiplicative factor dependent on the number of sites.

4 A More Efficient Set Intersection Protocol

The symmetric algorithm we have presented is simple and proven effective at controlling the disclosure of information. We now present a more complex variant that gives asymptotically improved performance in number of rounds, number of messages, and total number of bits transmitted. It also provides a practical improvement in information disclosure; the same total information is disclosed, but each party only sees a piece of that information.

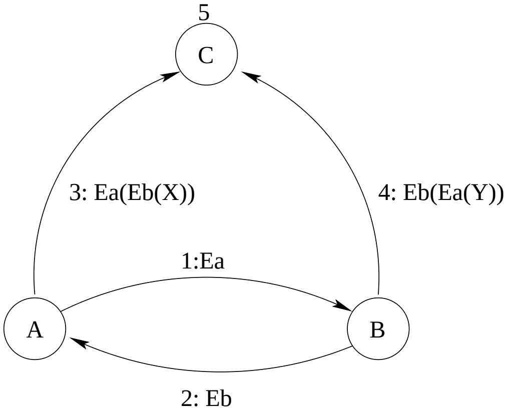
The key insight behind this protocol is to overlap the hashing and intersection phases. Note that any arbitrary parenthesization of the intersection expression still gives the same result.

$$\begin{gathered} S_0 \cap S_1 \cap \ldots \cap S_k \\ \equiv \\ (\ldots(S_0 \cap S_1) \cap S_2) \cap \ldots \cap S_k) \\ \equiv \\ (S_0 \cap S_1) \cap (S_2 \cap S_3) \cap \ldots \cap (S_{k-1} \cap S_k) \end{gathered}$$

The second observation is that it is not necessary to hash every set with all keys before intersecting the sets. Any time two items have been hashed by the same set of keys, they can be tested for equality. With careful ordering of the hashing we can perform the innermost intersections early. Repeating this at each level, the intersections can be carried out in the form of a binary tree, reformulating the intersection as

$$(\ldots(\log k) - 1 \ldots((S_0 \cap S_1) \cap (S_2 \cap S_3)) \cap \ldots \cap (S_{k-1} \cap S_k) \ldots \log k \ldots)$$

The difficulty is with carrying out intersections of two sets. As pointed out in Section 3.3, a party that sees the hashed results of its own set can probe, requiring at least three parties to perform the intersection. The solution is to use a party from the opposite side of the tree as this



1: A sends its key, Ea, to B
2: B sends its key, Eb, to A
3: A encrypts and sends its set to C
4: B encrypts and sends its set to C
5: C intersects both sets it receives to get the result

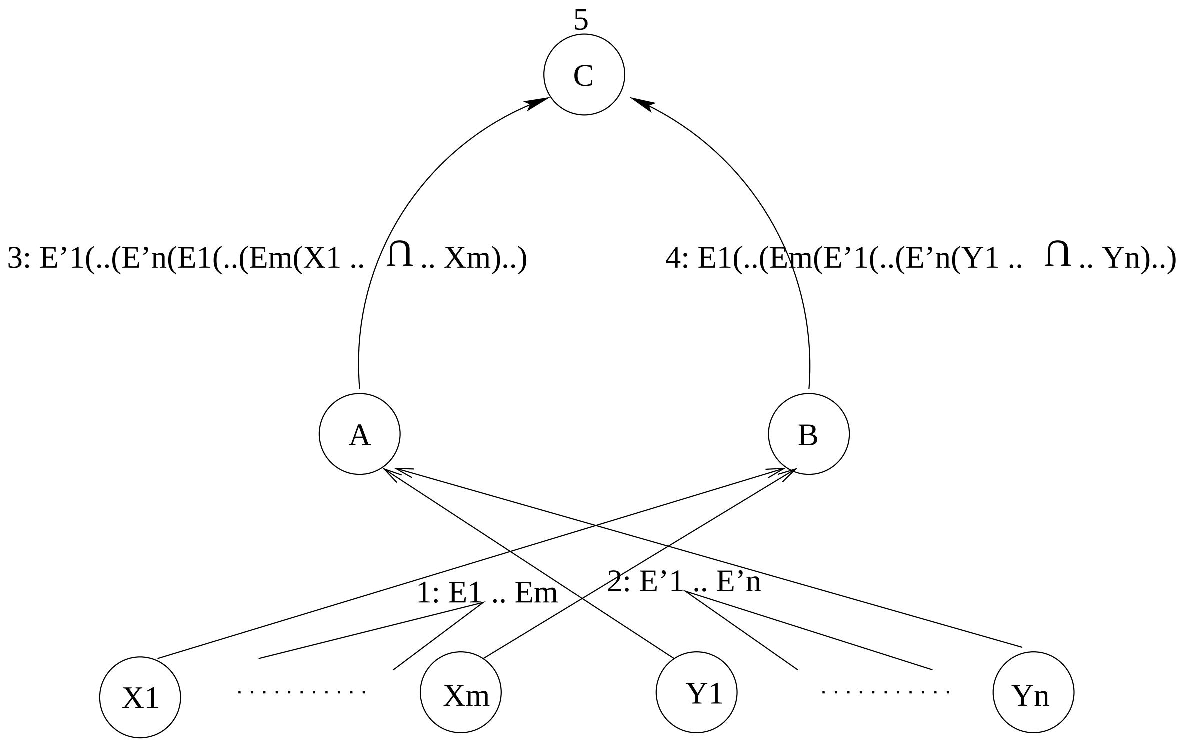
Figure 2: A single binary set intersection using a third party

third party. Each party hashes its set and sends it to its "intersection partner". The partner hashes it, and both send them to the parent third party. The third party performs the intersection. An example of this is given in Figure 2.

Each parent now has the intersection of the sets of its children, hashed with the keys of its children. To compute the intersection with its sibling, it must hash these items with the keys of its sibling's children. Since it does not see any items from its sibling's children, it gains no information by having these keys. Once this is complete, the siblings can send their intersections up the tree to compute the next level of intersection. This process is repeated until to root is reached, giving the final intersection. This process is illustrated in Figure 3. The complete algorithm is given in Protocol 2, and depicted graphically in Figure 4.

Interior nodes are assigned so that no node is on the path from itself (as a leaf) to the root. This avoids any information leak based on knowing the size of the intersection of one's own set with any subset of other nodes. (The root is the only node to learn the size of a subset containing its own set, but this subset contains half the nodes.) If the number of parties is a power of two, they form a complete tree with each party acting as both a leaf and at most once as an interior node. Only $k - 1$ parties are needed to act as interior nodes. If the tree is not complete, we can still make such an assignment provided $k \geq 4$. Leaf nodes whose sibling is not a leaf hash their own set with their own key, and with the keys of their sibling's children. The protocol then proceeds as normal. This eliminates the need for one interior node from the leaves on the other side of the tree. Balancing the tree with respect to these singleton nodes enables an assignment such that no node is on its own path to the root.

Since the sets each parent receives are hashed with the same set of keys, the commutative hashing property guarantees that the intersection of those sets will be of the correct size. Associativity of


5
C
3: E’1(..(E’n(E1(..(Em(X1 .. ∩ .. Xm)..)
4: E1(..(Em(E’1(..(E’n(Y1 .. ∩ .. Yn)..)
A
B
1: E1 .. Em
2: E’1 .. E’n
X1
Xm
Y1
Yn

1: m leaves send their encryption keys E1 .. Em to B
2: n leaves send their encryption keys E’1 .. E’n to A
3: A applies all encryption keys to its set and sends it to C
4: B applies all encryption keys to its set and sends it to C
5: C intersects what it has received from both A and B to get the result

Figure 3: Higher-level set intersection

Protocol 2 Tree-based protocol for computing size of set intersection

```
Require: $k > 3$ sites numbered $0 \dots k-1$, each having a local set $S_i$
Require: Maximum local set size $m$, threshold $r$ used to protect against probing
  for all sites $i$ do
    Generate a binary tree with $k$ leaves at levels $\log_2(k)$ and $\log_2(k) - 1$. Even leaves are to the left of root, odd to the right. If the tree is not complete, the lowest numbered nodes form the lowest level. Number non-leaf nodes as follows. 0 is root. The left-hand side is numbered with odd numbers using a preorder traversal, the right side with even numbers.
    {Each site now has an identical view of the tree.}
    Generate the hash key $E_i$
    for $j = |S_i|$ to $m$ do
      $S_i \leftarrow S_i \cup \{\text{prefix\_not\_in\_}U.i.j\}$ {Pad $S_i$ with unique items.}
    end for
    $M \leftarrow EncryptAndPermute(S_i, E_i)$
    if the sibling of $i$ is a leaf then
      Send $E_i$ to sibling
      $E \leftarrow$ receive from sibling
      $M \leftarrow EncryptAndPermute(M, E)$
    end if
    Send $\{E_i\}$ to the sibling of the parent of leaf $i$
    if the sibling of $i$ is a leaf then
      Send $M$ to parent
    end if
  end for
  {Nodes now act based on their "interior" position in the tree. For nodes whose sibling is not a leaf, the interior and leaf positions are the same.}
  for all sites $i > 0$ do
    $KeySet_i \leftarrow$ receive key set from left child of interior node $i$ of sibling
    $KeySet_i \leftarrow KeySet_i \cup$ receive key set from right child of sibling
    if the sibling of $i$ is a leaf {site $i$ is also an interior node} then
      $M_l \leftarrow$ receive from left child
      $M_r \leftarrow$ receive from right child
      $M \leftarrow M_l \cap M_r$
    end if
    if $|M| < r$ then
      Broadcast ABORT {Potential Probe}
    else
      for all $E \in KeySet_i$ do
        $M \leftarrow EncryptAndPermute(M, E)$
      end for
      if parent of $i$ is not 0 then
        Send $KeySet_i$ to sibling of parent of interior node $i$
      end if
      Send $M$ to parent of interior node $i$
    end if
  end for
  if site is 0 then
    $M_l \leftarrow$ receive from left child, $M_r \leftarrow$ receive from right child
    Broadcast result $|M_l \cap M_r|$
  end if
```

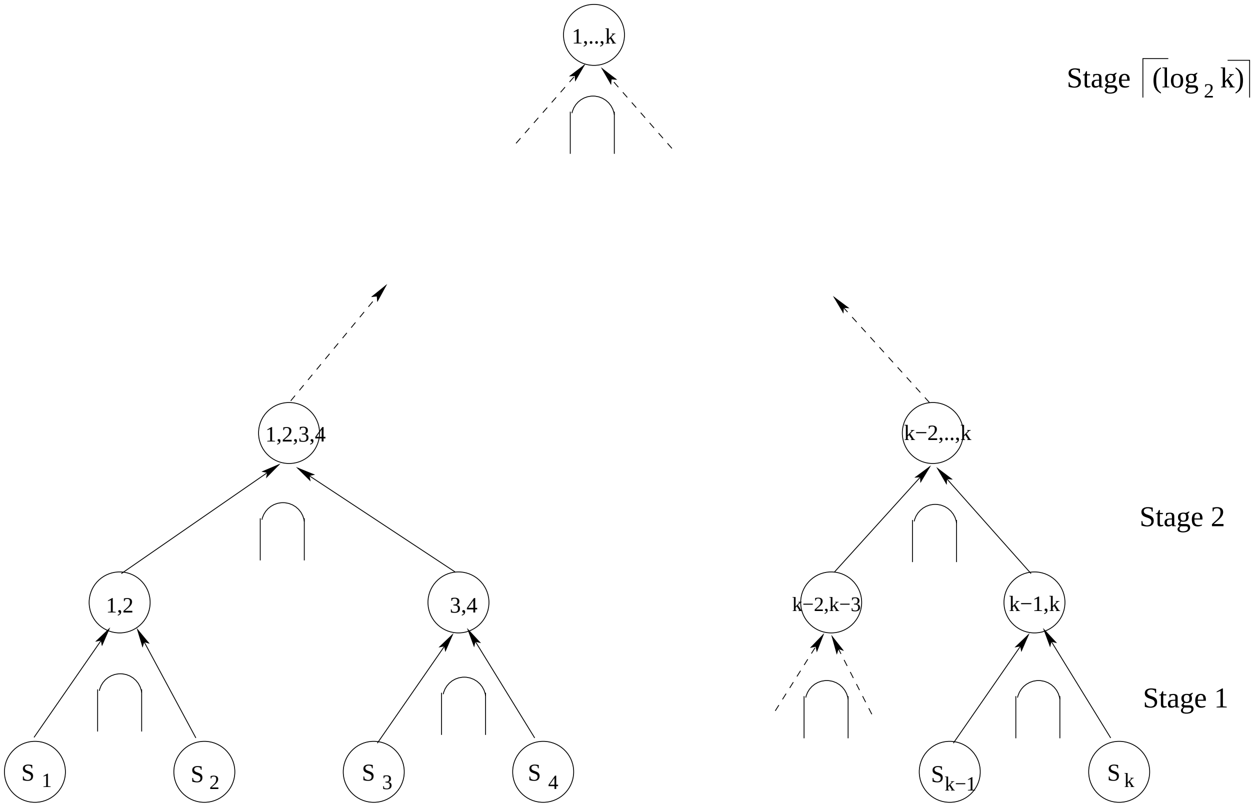


Figure 4: A more efficient protocol

intersection ensures that the order does not affect the result. An inductive argument shows that the protocol generates the correct result.

Protocol 2 demonstrates another optimization. Instead of sending sets to be hashed by other sites, a site sends its key. The numbering of the tree ensures that no site sees items hashed with any key it knows (except root, which knows only its own key and sees items hashed with that plus several others.) Thus, in the absence of collusion, sending a key gives the receiver no additional information.

4.1 Security Analysis

Protocol 1 is symmetric, and reveals to all parties the intersections of any subset of items except its own. Protocol 2 reveals the same type of information, however, each site learns the intersection size of at most three subsets:

- $|\bigcap_{p \in descendants} S_p|$
- $|\bigcap_{p \in left\ descendants} S_p|$
- $|\bigcap_{p \in right\ descendants} S_p|$

The total information revealed is considerably less, $O(k)$ intersections as opposed to $O(2^k)$. In addition, the limited amount revealed to any party enables an assignment of parties to nodes based on trust and which specific intersections can be disclosed. This gives considerable flexibility in meeting specific security policy goals.

Theorem 4.1 *Protocol 4 privately computes the size of the intersection set $|S| = |\bigcap_{p=0}^{k-1} S_p|$. Each site learns the final result, and if it serves as an interior node in the tree it learns:*

- $|\bigcap_{p \in descendants} S_p|$
- $|\bigcap_{p \in left\ descendants} S_p|$
- $|\bigcap_{p \in right\ descendants} S_p|$

Proof. The proof proceeds by simulating the view of a party i as it proceeds through Protocol 2. The simulator effectively runs Protocol 2, all we need to show is that the received messages can be simulated.

First, let us look at the case of an "extra" node: a leaf whose sibling is not a leaf. These nodes (at most two, one odd and one even) serve in the same spot as a leaf and interior node. If i is one of these nodes, it will receive the keys E_{i-2} and E_{i-4}. This is the only message it receives. Since these were not used to generate any information i will receive, they can be simulated by randomly generating keys for the hash function E. Protocol 2 generates the rest of the view for these nodes, except for receiving the final result. As the result is known to the simulator, generating it is trivial.

The remaining non-root nodes are slightly more complex, as they receive three sets of messages. The first message received is the key of their sibling in their position as a leaf. This is simulated by randomly generating a key for the hash function E. The next are the two sets of keys of their sibling's descendants based on their position as an interior node. These are simulated by randomly generating keys for the hash function E, the number of keys to generate is known from i's position in the tree. The final messages received by i are the intersection sets of i's left and right descendants. To simulate these, i takes advantage of three facts:

1. i knows the sizes $|\cap_{p \in left\ descendants} S_p|$ and $|\cap_{p \in right\ descendants} S_p|$ of the received sets,
2. i knows the number of items the two sets have in common $|\cap_{p \in descendants} S_p|$, and
3. i has *no* knowledge of the keys used to hash the items in the sets.

Using fact 2, the simulator for i generates $|\cap_{p \in descendants} S_p|$ items, by randomly selecting items from a uniform distribution over the domain of E, and places them in both the simulated M_l and M_r. Fact 1 is then used to complete M_l and M_r, by generating $|\cap_{p \in left\ descendants} S_p| - |\cap_{p \in descendants} S_p|$ to insert into M_l and $|\cap_{p \in right\ descendants} S_p| - |\cap_{p \in descendants} S_p|$ to insert into M_r.

The simulated sets M_l and M_r are now the same size as those seen in the actual protocol, and their intersection contains the same number of items as those in the actual protocol. Since i has no knowledge of the keys used to hash the items (fact 3), security of hashing/encryption guarantees values in hashed sets are computationally indistinguishable from values chosen from a uniform random distribution over the domain of the hash function E. Therefore, the simulated view is computationally indistinguishable from that seen by i during the actual execution of the protocol.

The argument for site 0, the root, is slightly different. The simulator is the same as other interior nodes. This site receives only E_2, the key of its sibling. It does see items hashed with E_2, as well as its own key E_0, so fact 3 does not hold. However, by the time it sees items hashed with E_0 and E_2, they have also been hashed with (at least) E_1. Since 0 does not know E_1, the computational indistinguishability argument still holds.

The simulator requires one key generation or selection of a random value from the domain of E for each item in the received messages. The number of items is bounded by the maximum set size m. Assuming key generation or random value selection is polynomial in the size of the input, and that Protocol 2 runs in polynomial time (see Section 4.2), the view seen by any site i can be simulated in time polynomial in the size of the input. □

Absent collusion, Protocol 2 is as effective as Protocol 1 with malicious parties. A malicious party can alter what it sends, however, since it never sees anything based on messages it has sent except the final output it can only gain information from the final result. This constitutes a probing attack, and the information gain possible is restricted by the minimum size threshold r. Site 0 does see information based on its first set of messages, however the intermediate hashing and threshold test prevents it from gaining additional information from what it sent as a leaf.

Collusion poses a significant problem. Collusion between the parent of a leaf node and its right child can give it both the hash key E_l and the hashed set M_l of the left child. It can now probe for the existence of any item I in that set, by testing if $E_l(I) \in M_l$. Collusion with its own sibling or the sibling of its ancestors also gives it this key. Even if some sites were not trusted (i.e., they may collude with some other sites), it would often be possible to assign sites to tree nodes in such a way that the untrusted sites would not gain by colluding.

4.2 Cost of Protocol 2

At first glance, the encryption cost appears similar to Protocol 1. Every item in the final intersection is hashed with every key. However, duplicate items are filtered out at higher levels. This reduces the number of items to be hashed. Leaf nodes perform $2m$ encryptions. Lowest level parent nodes perform 2 additional encryptions on every item in the intersection of their children; at most $2m$.

The next level must hash with 4 keys. The level below the root ends up with $k/2$ keys. Multiplying by the number of nodes at each level gives $k * m$ encryptions at each level, for a worst case total of $(2k + k\log_2(k)) * m$ encryptions. Parallelism gives some benefit, but since the upper levels perform more encryptions the encryption time is still $O(k * m)$.

More important is the savings in number of rounds and messages. The number of rounds of messages is one for the leaf key exchange, and $\lceil \log_2(k) \rceil$ rounds of sending key sets and hashed sets up the tree. The key exchange requires each leaf to send one message of the number of bits in the hash key. (The one or two "extra" nodes whose sibling is not a leaf are spared sending this message.) For each edge in the tree there is one "hashed set" message and (except for the root) a corresponding key set message, for a total of $2k - 4$ messages. Each hashed set message is at most $m * \textit{hashed item size}$ bits. The key set messages grow as they grow up the tree; a total of $k * \textit{hash key size}$ bits are sent at each level.

The overall communication cost of Protocol 2 is then:

$$\begin{array}{cl} 3k - 4 = O(k) & \text{messages} \\ k\log_2(k) * \textit{hash key size} + 2(k-1)m * \textit{hashed item size} \approx O(k * m) & \text{bits} \\ \lceil \log_2(k) \rceil = O(\log k) & \text{rounds} \end{array}$$

This is a substantial improvement over the $O(k^2)$ messages, $O(k^2 m)$ bits, and $O(k)$ rounds of Protocol 1. However, as we shall see in the next Section, there are practical situations where Protocol 1 has advantages.

The number of bits and number of encryptions is in practice a pessimistic estimate. It is likely that the size of intersections will be significantly smaller than m, and will rapidly approach $|S|$. While the asymptotic results do not change, the effect of parallelism on the encryption cost is likely to improve significantly, as the majority of encryptions occur only at the low levels. The set message sizes at higher levels will also shrink, although each key transmission message will grow. Thus, the effective total time to run the algorithm should be closer to $O(m + \log k)$ than $O(m \log k)$.

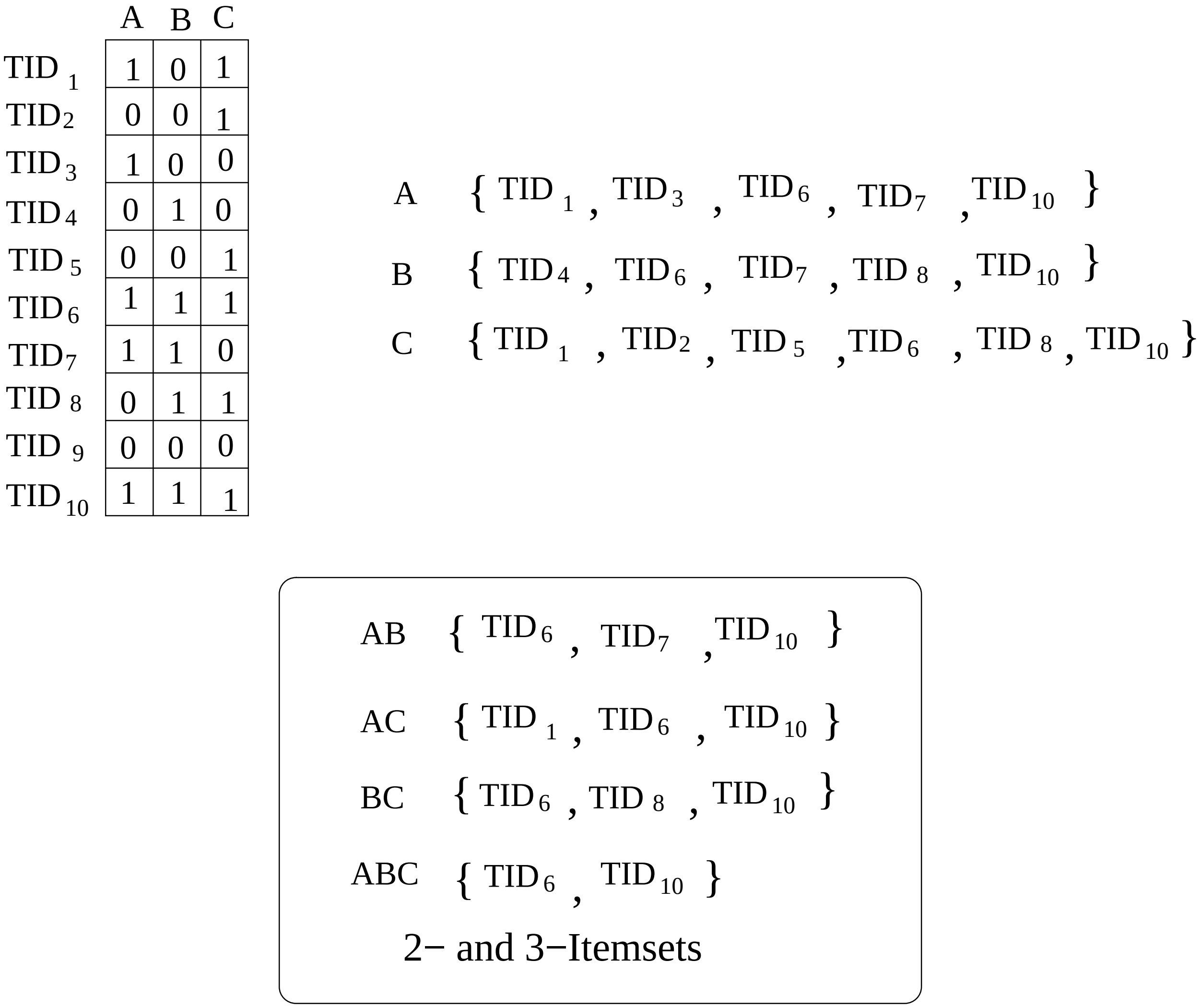
5 Association Rules

We now show how the set intersection protocol enables secure mining of association rules in vertically partitioned data.

5.1 Problem Definition

Let there be k parties $P_1, P_2, \ldots, P_k$. We consider an extended form of the heterogeneous database scenario considered in [9], a vertical partitioning of the database between the k parties. The association rule mining problem can be formally stated as follows[2]: Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Associated with each transaction is a unique identifier, called its *TID*. We say that a transaction T *contains* X, a set of some items in $\mathcal{I}$, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with **confidence** c if c% of transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has **support** s in $\mathcal{D}$ if s% of the transactions in $\mathcal{D}$ contain $X \cup Y$.

A transaction database can be represented in several ways. One is to describe it as a $m \times n$ boolean matrix, where m represents the number of items/attributes/features and n represents the number of transactions. A 1 denotes the presence of an item in the transaction, a 0 represents its absence.



	A	B	C
TID_1	1	0	1
TID_2	0	0	1
TID_3	1	0	0
TID_4	0	1	0
TID_5	0	0	1
TID_6	1	1	1
TID_7	1	1	0
TID_8	0	1	1
TID_9	0	0	0
TID_{10}	1	1	1

A $\{TID_1, TID_3, TID_6, TID_7, TID_{10}\}$

B $\{TID_4, TID_6, TID_7, TID_8, TID_{10}\}$

C $\{TID_1, TID_2, TID_5, TID_6, TID_8, TID_{10}\}$

AB $\{TID_6, TID_7, TID_{10}\}$

AC $\{TID_1, TID_6, TID_{10}\}$

BC $\{TID_6, TID_8, TID_{10}\}$

ABC $\{TID_6, TID_{10}\}$

Figure 5: An example database in its various representations

A second representation is the transaction identifier (TID) list approach. Every attribute/feature has a transaction identifier (TID) list associated with it. This TID-list contains the identifiers of transactions that contain the attribute. This is a more compact representation in practice, as most transactions are sparse (contain few items). It also lends itself well to secure processing with our size of set intersection protocol.

A clearer explanation of these representations is given by the following example. Consider a database having ten transactions $TID_1, TID_2, \ldots, TID_{10}$, and three attributes A, B, and C. The figure 5 illustrates the boolean matrix view and the TID-list view of the database. The figure also gives the TID-list of all combinations of attributes. It is clear that a transaction supports an itemset (set of attributes) if and only if its TID is present in the TID-list for all the attributes in the itemset. To find the number of transactions that support a given itemset we need only find the cardinality of the intersection set of the TID-lists of these attributes.

Sections 3 and 4 present an efficient way to compute the cardinality of the intersection set without disclosing the items in that set. We now show how to generalize these protocols from a k-itemset where each party has one of the attributes to general association rules. This generalization shares no information other than through computing the size of the set intersections.

There are two possibilities:

- a single party contributes no attributes to the itemset.
- a single party contributes more than one attribute to the itemset.

A party that has none of the attributes does not participate in the protocol. This reveals that it does not have the attribute, but the mapping of attributes to sites is presumed to be public. It is the transactions, and which attributes are in which transaction, that must be kept private. If a party contributes more than one attribute, it locally computes the intersection before participating in the protocol.

For example, suppose we want to compute if a particular 5-itemset is frequent, with

- P_1 having 2 of the attributes, A_{11}, A_{13},
- P_2 having 2 of the attributes, A_{22}, A_{23}, and
- P_3 having the remaining 1 attribute, A_{31}.

P_1, P_2 and P_3 want to know if the itemset $l = \langle A_{11}, A_{13}, A_{22}, A_{23}, A_{31} \rangle$ is frequent. P_1 locally generates the set $S_1 = S_{11} \cap S_{13}$, P_2 locally generates the set $S_2 = S_{22} \cap S_{23}$ and P_3 generates the set $S_3 = S_{31}$. $|S_1 \cap S_2 \cap S_3|$ is the frequency of the itemset.

The full procedure for computing frequent itemsets is given in Algorithm 1.

Algorithm 1 Privacy Preserving Association Rule Mining Algorithm

```
 1: $L_1 = \{$large 1-itemsets$\}$
 2: for $(k = 2; L_{k-1} \neq \emptyset; k++)$ do
 3:    $L_k = \emptyset$
 4:    $C_k =$ apriori-gen$(L_{k-1})$;
 5:    for all candidates $c \in C_k$ do
 6:       if all the attributes in $c$ are entirely at any one party $P_l$ then
 7:          party $P_l$ independently calculates $c.count$
 8:       else
 9:          let $P_1$ have $l_1$ of the attributes, ..., $P_k$ have $l_k$ attributes $(\sum_{i=1}^{k} l_i = |c|)$
10:          construct $S_1$ on $P_1$'s side, ..., $S_k$ on $P_k$'s side,
11:          where $S_i = S_{i1} \cap \ldots \cap S_{il_i}$, $1 \leq i \leq k$
12:          compute $c.count = |\cap_{j=1..k} S_j|$ using protocol 1 or 2
13:       end if
14:       $L_k = L_k \cup c | c.count \geq minsup$
15:    end for
16: end for
17: Answer $= \cup_k L_k$
```

In step 4, the function apriori-gen takes the set of large itemsets L_{k-1} found in the $(k-1)th$ pass as an argument and generates the set of candidate itemsets C_k. This is done by generating a superset of possible candidate itemsets and pruning this set. [3] discusses the function in detail. Given the counts and frequent itemsets, we can compute all association rules with *support* $\geq$ *minsup*.

Two-itemsets pose a special problem, as the set intersection protocols only work for three or more parties. Assuming we have at least three parties overall, we can use one as an "untrusted third party" to compute the size of 2-itemsets. This is analogous to the leaf actions of Protocol 2: the two parties exchange keys, hash their items with both keys, and send the hashed sets to the third party. The third party reports the size of the set if it is greater than the threshold. The same argument as in the proof of Protocol 2 demonstrates the security of this approach.

5.2 Proof of correctness

Candidate itemsets are generated by a straightforward application of the Apriori-gen procedure. For the proof of correctness of that procedure refer to [4]. As long as the input to the procedure is correct, the C_k sets are generated correctly.

We show by induction that the L_j sets are generated correctly. For the basis step with $j = 1$, L_1 is correctly generated directly from the data. Assume that L_{k-1} has been correctly generated. Hence C_k is generated correctly from L_{k-1}. Assuming that step 12 computes the count correctly, L_k is correctly computed.

The critical step is computing *c.count*, step 12. The correctness of Protocols 1 and 2 have already been demonstrated. Thus, the entire association rule mining algorithm gives correct results.

5.3 Security Analysis

Only steps 1, 12 and 14 require exchanging information. Since the final result $\cup_k L_k$ is known to both parties, steps 1 and 14 reveal no additional information. In Sections 3 and 4, we show how to compute step 12 revealing only limited information.

Theorem 5.1 *Algorithm 1 privately computes the association rules present in the database without revealing any information other than the support of all the possible itemsets.*

Proof. The security of the protocol is based upon the security of steps 1, 12 and 14. The final result $\cup_k L_k$ is known to both parties, so steps 1 and 14 can be simulated directly from the result. To prove the security of step 12, we first present the composition theorem from [17]:

Theorem 5.1 *(Composition Theorem for the semi-honest model): Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.*

Proof. Refer to [17]. □

The composition theorem states that if a protocol is shown to be composed of many invocations of smaller subproblems, and if the subproblems are computable privately, then the overall protocol is secure. Treating Protocols 1 and 2 as f, we can show that the association rule computation, g, is computed privately. Setting the threshold r to *minsupport* causes the protocol to abort if the itemset is not supported, so we only learn the support of supported itemsets. (Parties can refuse to participate if they feel *minsupport* is so low as to enable probing attacks.) Protocols 1 and 2 also reveal the sizes of some (or all) subsets of the itemset being tested. However, a candidate

k-itemset is only generated if all its subsets are frequent. The result $\cup_k L_k$ of Algorithm 1 includes all frequent subsets. If we also include the support of the supported itemsets in the result, as is commonly done in association rule mining, the size of these frequent subsets is also known. These sizes can be provided as input to the simulator to prove the security of Protocols 1 and 2. This demonstrates that the association rule mining algorithm 1 is fully secure under the Secure Multiparty Computation definitions.

5.4 Communication Analysis

The communication analysis critically depends on the number of times step 12 is called. For each call to step 12, we incur the cost of the set intersection algorithm. If we let r be the maximum size of a frequent itemset (i.e., no frequent $r+1$-itemsets are found), and let $C_i, (i = 1 \ldots r)$ represent the number of candidate itemsets at each round, the total communication using Protocol 2 is:

$$\begin{array}{cl} \sum_{i=1}^{r} C_i * (3k-4) & \text{messages} \\ \sum_{i=1}^{r} C_i * 2(k-1) * (m * \mathit{hashed\ item\ size} + k \log_2(k) * \mathit{hash\ key\ size}) & \text{bits} \end{array}$$

Surprisingly, a much more efficient association rule mining can be constructed using the Protocol 1. The goal of association rule mining is to find all frequent itemsets. If we use Protocol 1 to immediately find the size of $| \cap_{i=1..k} S_k|$, each party also has enough information to find the intersection sizes, or support, of all smaller itemsets that do not include its own items. The “flaw” in Protocol 1 becomes a benefit. Each party compute all frequent itemsets based on the hashed sets received in the initial intersection stage. If any of the frequent itemsets involve its neighbor’s set, the corresponding hashed sets are forwarded to its neighbor (since these are the only ones the neighbor does not have). Now, every site has complete information and can compute all association rules, but still prevents probing. The new association rule mining algorithm can be defined as follows:

```
Call Protocol 1, with each attribute treated as a different site (even if both are at the same site.)
Perform local association rule mining on the $TS_j$ to determine frequent itemsets not involving
ones own attributes
if Any frequent itemsets involve ones left or right neighbor then
  Send the support of those itemsets to the neighbor
  Now each site can compute the remaining association rules involving its own attributes
end if
```

Since the only action carried out involves calling the set intersection protocol, the security analysis of this algorithm is completely reducible to the security of the set intersection protocol. The one caveat is that each site learns the support of itemsets in which it does not participate, even if they are below the support threshold. This is less secure than Algorithm 1, but still does not reveal individual itemsets or allow probing.

Since only one call to the set intersection algorithm takes place, the communication analysis is straightforward. Note that all the attributes that are frequent 1-itemsets are sent for intersection. No local intersection is carried out. The communication cost is therefore

$$\begin{array}{cl} k * (2k-2) = O(k^2) & \text{messages} \\ K * (2K-2) * m * \mathit{encrypted\ item\ size} = O(K^2 m) & \text{bits} \\ k & \text{rounds} \end{array}$$

where K is the number of frequent attributes rather than the number of sites.

Number of items encrypted	Key Size			Transfer Time
	256	512	1024	
1k	< 0.0001s	5s	29s	0.0027s
10k	10s	47s	286s	0.007s
100k	90s	467s	2827s	0.04s
1M	900s	4660s	28762s	0.41s

Table 1: Computation and Communication Cost of encryption

5.5 Practical Cost

We have run experiments to evaluate what the actual cost would be for a number of different cases. The experiments were run on a SUN Blade 1000 workstation with a 900Mhz processor and 1GB of RAM. First, we tabulate the pure encryption cost for different key sizes. An encryption key size of 512 bits is sufficient for typical applications. It can be seen that the computation cost rises linearly with the number of item to be encrypted (as expected). Note that encryption can proceed at different sites in parallel. Thus Table 1 gives the encryption time per round.

We also measured the transfer time required to send the encrypted data from one site to another over a 100Mb network. The encrypted data required comparable size in the GNU GMP raw bit format regardless of key size.

Using the data generated in the prior table we can easily estimate the extra cost incurred by privacy while doing association rule mining in a particular situation (characterized by the number of transactions, attributes and parties). Table 2 estimates the computation cost assuming that the encryption key size is 512 bits. Table 3 estimates the communication cost assuming communication is over a 100Mb network. Both assume that attributes can have at most $100k$ transactions. We give a worst case scenario estimate assuming that all the attributes are frequent 1-itemsets and also encrypting and communicating the entire attribute. In practice, the cost would be much lower (at least an order of magnitude), since all attributes may not be frequent and even the frequent attributes are present in only a fraction of the total number of transactions. The cost for other values of key size and communication bandwidth can be easily extrapolated using the data provided above. It is clear from this data that the computation cost greatly exceeds the communication cost. Computation cost can be drastically reduced by optimizing the code (we used the generic variant of GNU gmp), or through widely-available special-purpose encryption hardware. Note that the cost described here is the additional cost of assuring privacy. We still need to compute the association rules at each site. However, this cost is comparable to simply running APRIORI and is thus quite feasible.

Though this means that the entire database will have to be transferred a number of times (depending upon the number of sites), even a non-secure solution requires on the order of transferring the entire database once. It is important to note that this communication cost is dependent only upon the number of attributes and the number of sites. It is independent of the number of iterations of the APRIORI algorithm. This is likely to lead to huge cost savings in extremely high dimensional databases with many transactions.

Number of attributes	Number of Sites				
	2	3	5	10	20
10	9340s	14010s	23350s	46700s	-
50	13hr	19.5hr	32.5hr	65hr	130hr
100	26hr	39hr	65hr	130hr	260hr
200	52hr	78hr	130hr	260hr	520hr

Table 2: Worst-case added computation to achieve privacy

Number of attributes	Number of Sites				
	2	3	5	10	20
10	1.6s	3.6s	10s	40s	-
50	8s	18s	50s	200s	800s
100	16s	36s	100s	400s	1600s
200	32s	72s	200s	800s	3200s

Table 3: Worst-case communication cost increase to achieve privacy

6 Conclusions and Further Work

We have presented two protocols for securely computing the cardinality of an intersection of sets at distributed sites. The security properties of both have been rigorously proven. We have also presented distributed association rule mining as an application to motivate these protocols, and compared the protocols for creating an efficient privacy-preserving association rule mining algorithm.

As we pointed out in Section 2.2, there have been other efforts in privacy-preserving data mining. One problem is that the number of types of data mining continues to grow; each new type generates a need for several privacy-preserving data mining algorithms (depending on how data is partitioned, privacy constraints, etc.) We have presented a primitive that is useful for association rule mining, one area for future work is to see what other types of data mining may be addressed with this primitive. Our long-range goal is to develop a toolkit of secure computations that can be combined to form a variety of privacy-preserving data mining techniques.

A key challenge is ensuring that the security properties are maintained as these tools are combined. The association rule mining presented here had the serendipitous effect that the information "leaked" by the intersection protocol was part of the mining result. We cannot count on this for other types of mining. Simply applying one secure algorithm to the results of the other does not guarantee security; we must prevent disclosure of the intermediate result or show that it does not compromise privacy. Iterative data mining algorithms pose a particular challenge in this respect.

Both of the protocols presented are vulnerable to collusion between parties. Also, though our protocols go somewhat beyond a semi-honest trust model, they are definitely not applicable in a fully malicious setting. More work is required to develop protocols which are resistant to collusion as well as applicable in a malicious setting.

Another direction for future research is experimental evaluation. We have provided experimental estimates but having a working system would allow use of the intersection protocol in other tasks. It would also be quite useful to be able to provide tight bounds on the amount of communication required for truly private computation. An easy lower bound can be provided by the size of the database, but it should be possible to find much tighter lower bounds.

Acknowledgments

We wish to acknowledge the contributions of Mike Atallah, discussions with whom led to the original k-fold hashing idea. We also thank Murat Kantarcioglu, who has provided valuable input into the proof mechanisms, giving clear bounds on the information released.

References

[1] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, Santa Barbara, California, USA, May 21-23 2001. ACM.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 26–28 1993.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 12-15 1994. VLDB.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. Technical Report RJ9839, IBM Research, June 1994.

[5] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, Dallas, TX, May 14-19 2000. ACM.

[6] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In *Seventh International Workshop on Algorithms and Data Structures (WADS 2001)*, Providence, Rhode Island, USA, August 8-10 2001.

[7] Philip Chan. *An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 1996.

[8] Philip Chan. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8:5–28, 1997.

[9] Rong Chen, Krishnamoorthy Sivakumar, and Hillol Kargupta. Distributed web mining using bayesian networks from multiple data streams. In *The 2001 IEEE International Conference on Data Mining*, San Jose, California, November 29 - December 2 2001. IEEE.

[10] David Wai-Lok Cheung, Vincent Ng, Ada Wai-Chee Fu, and Yongjian Fu. Efficient mining of association rules in distributed databases. *IEEE Trans. Knowledge Data Eng.*, 8(6):911–922, December 1996.

[11] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative scientific computations. In *14th IEEE Computer Security Foundations Workshop*, pages 273–282, Nova Scotia, Canada, June 11-13 2001.

[12] Wenliang Du and Mikhail J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, December 10-14 2001.

[13] Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, volume 14, pages 1–8, Maebashi City, Japan, December 9 2002. Australian Computer Society.

[14] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy preserving mining of association rules. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, Edmonton, Alberta, Canada, July 23-26 2002.

[15] Mr. Feingold, Mr. Corzine, Mr. Wyden, and Mr. Nelson. Data-mining moratorium act of 2003. U.S. Senate Bill (proposed), January 16 2003.

[16] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.

[17] Oded Goldreich. Secure multi-party computation, September 1998. (working draft).

[18] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *The 2002 International Conference on Parallel Processing*, Vancouver, British Columbia, August 18-21 2002.

[19] Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, Madison, Wisconsin, June 2 2002.

[20] Murat Kantarcioglu and Jaideep Vaidya. An architecture for privacy-preserving mining of client information. In Chris Clifton and Vladimir Estivill-Castro, editors, *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, volume 14, pages 37–42, Maebashi City, Japan, December 9 2002. Australian Computer Society.

[21] Murat Kantarcıoğlu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowledge Data Eng.*, 16(4), July 2004.

[22] Xiaodong Lin, Chris Clifton, and Michael Zhu. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems*, to appear 2004.

[23] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, August 20-24 2000.

[24] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996.

[25] Andreas Prodromidis, Philip Chan, and Salvatore Stolfo. *Advances in Distributed and Parallel Knowledge Discovery*, chapter 3: Meta-learning in distributed data mining systems: Issues and approaches. AAAI/MIT Press, 2000.

[26] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 682–693, Hong Kong, August 20-23 2002. VLDB.

[27] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of 21st International Conference on Very Large Data Bases*, pages 432–444. VLDB, September 11-15 1995.

[28] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, 2nd edition, 1995.

[29] Doug Struck. Don't store my data, Japanese tell government. *International Herald Tribune*, page 1, August 24-25 2002.

[30] Total information awareness (TIA) system.

[31] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Alberta, Canada, July 23-26 2002.

[32] Rüdiger Wirth, Michael Borth, and Jochen Hipp. When distribution is part of the semantics: A new problem class for distributed knowledge discovery. In *Ubiquitous Data Mining for Mobile and Distributed Environments workshop associated with the Joint 12th European Conference on Machine Learning (ECML'01) and 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'01)*, Freiburg, Germany, September 3-7 2001.

[33] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.