Privacy-Preserving Data Mining

C. Clifton, M. Kantarcıoğlu, and J. Vaidya

Purdue University
{clifton, kanmurat, jsvaidya}@cs.purdue.edu

The growth of data mining has raised concerns among privacy advocates. Some of this is based on a misunderstanding of what data mining does. The previous chapters have shown how data mining concentrates on extraction of rules, patterns and other such summary knowledge from large data sets. This would not seem to inherently violate privacy, which is generally concerned with the release of individual data values rather than summaries.

To some extent, this has been recognized by privacy advocates. For example, the Data-Mining Moratorium Act proposed in the U.S. Senate in January 2003 would have stopped all data-mining activity by the Department of Defense[15]. A later version is more specific, defining "data-mining" as searches for individual information based on profiles[25]. While data mining may help in the *development* of such profiles, with the possible exception of outlier detection data mining would not be a forbidden activity under the later bill.

Although data mining *results* may have survived the scrutiny of privacy advocates, the data mining *process* still faces challenges. For example, the Terrorism Information Awareness (TIA) program, formerly known as Total Information Awareness, proposed government use of privately held databases (e.g., credit records) to aid in the discovery and prevention of terrorist activity[9]. This raised justifiable concerns, leading to a shutdown of the program[22] and proposals for restriction on government use of privately held databases[25]. The real problem is the potential for misuse of the information. The TIA program did not propose collection of new data, only access to existing collections. However, providing a single point of access to many collections, and linking individuals across these collections, provides a much more complete view of individuals than can be gleaned from any individual collection. While this could significantly improve capabilities to identify and track terrorists, it also makes it easier to misuse this information for unethical or illegal harassment of political dissidents, unpopular officials, or even neighbors of a rogue agent or hacker who manages to break into the system.

Information gives knowledge gives power, and many feel that the potential for misuse of this power exceeds the benefit.

Privacy-preserving data mining has emerged as an answer to this problem. The goal of privacy-preserving data mining is to develop data mining models without increasing the risk of misuse of the data used to generate those models. This is accomplished by ensuring that nobody but the original possessor of data sees individual data values. Since no real or virtual "data warehouse" providing integrated access to data is used, the potential for misuse of data is not increased by the data mining process. While the potential for misuse of the produced data mining models is not eliminated, these are considered to be sufficiently removed from individual data values (or perhaps so important) that the threat to individual privacy is not an issue.

Privacy-preserving data mining work is divided into two broad classes. One, first proposed in [3], is based on adding noise to the data before providing it to the data miner. Since real data values are not revealed (the noise can be added at the data source), individual privacy is preserved. The challenge in this class is developing algorithms that achieve good results in spite of the noise in the data. While these techniques have been shown effective, there is growing concern about the potential for noise reduction and thus compromise of individual privacy[1, 10, 21]. As a result, we will touch only briefly on this approach, giving a summary of how the method of [3] works in Sect. 2.

The second class of privacy-preserving data mining comes out of the cryptography community[24]. The idea is that the data sources collaborate to obtain data mining results without revealing *anything* except those results. This approach is based on the definitions and standards that have guided the cryptography community, in particular Secure Multiparty Computation[16, 38]. The disadvantage to this approach is that the algorithms are generally distributed, requiring active participation of the data sources. However, as this model limits any privacy breach to that inherent in the results, this chapter will emphasize this class of privacy-preserving data mining techniques.

1 Privacy-Preserving Distributed Data Mining

Privacy-preserving distributed data mining uses algorithms that require parties to collaborate to get results, while provably preventing disclosure of data except the data mining results. As a simple example, assume several supermarkets wish to collaborate to obtain global "market basket" association rules, without revealing individual purchases or even the rules that hold at individual stores. To simplify, assume we only wish to compute the global support count for a single itemset, e.g., "beer" and "diapers". Each market first computes the number of market baskets it has that contain both items. A designated starting market also generates a random number R. The starting party adds its support count S_1 to R, and sends $R + S_1$ to the second market. The second market adds its support count, sending $R + S_1 + S_2$ to the third market. This continues, with the nth market sending $R + \sum_{i=1}^{n} S_i$ to the first market. The first market subtracts R to obtain the desired result.

A crucial assumption for security is that markets $i+1$ and $i-1$ do not collude to learn S_i. However, if we can assume no collusion, all operations take place over a closed field, and the choice of R is uniformly distributed over the field, we can see that no market learns anything about the other market's support counts except what can be inferred from the result and one's own data. While this does not guarantee that individual values are not revealed (for example, if the global support count is 0, we know the count is 0 for every market), it does preserve as much privacy as possible assuming we must obtain the results (in this case, global support count.)

The concept of privacy in this approach is based on a solid body of theoretical work. We briefly discuss some of this work now, then describe several techniques for privacy-preserving distributed data mining to demonstrate how this theory can be applied in practice.

1.1 Privacy Definitions and Proof Techniques

Secure Multiparty Computation (SMC) originated with Yao's Millionaires' problem[38]. The basic problem is that two millionaires would like to know who is richer, but neither wants to reveal their net worth. Abstractly, the problem is simply comparing two numbers, each held by one party, without either party revealing its number to the other. Yao presented a solution for any efficiently computable function restricted to two parties and semi-honest adversaries.

What do we mean by *semi-honest*? The secure multiparty computation literature makes use of two models of what an adversary may do to try to obtain information. These are:

Semi-Honest: Semi-honest (or Honest but Curious) adversaries will follow the protocol faithfully, but are allowed to try to infer the secret information of the other parties from the data they see during the execution of the protocol.

Malicious: Malicious adversaries may do anything they like to try to infer secret information (within the bounds of polynomial computational power). They can abort the protocol at any time, send spurious messages, spoof messages, collude with other (malicious) parties, etc.

Reference [16] extended Yao's result to an arbitrary number of parties as well as malicious adversaries. The basic idea is based on circuit evaluation: The function is represented as a boolean circuit. Each party gives the other(s) a randomly determined share of their input, such that the exclusive or of the shares gives the actual value. The parties collaborate to compute a share of the output of each gate. Although the exclusive or of the shares gives the output of the gate, the individual shares are random in isolation. Since each party learns nothing but its share, nothing is revealed, and these shares can be used in the next gate in the circuit. At the end, the shares are combined to produce the final result. Since the intermediate shares were randomly determined, nothing is revealed except the final result.

Informally, the definition of privacy is based on equivalence to having a trusted third party perform the computation. Imagine that each of the data sources gives their input to a (hypothetical) trusted third party. This party, acting in complete isolation, computes the results and reveals them. After revealing the results, the trusted party forgets everything it has seen. A secure multiparty computation approximates this standard: no party learns more than with the (hypothetical) trusted third party approach.

One fact is immediately obvious: no matter how secure the computation, some information about the inputs may be revealed. If one's net worth is $100,000, and the other party is richer, one has a lower bound on their net worth. This is captured in the formal SMC definitions: any information that can be inferred from one's own data and the result can be revealed by the protocol. For example, assume one party attributes A and B for all individuals, and another party has attribute C. If mining for association rules, gives that $AB \Rightarrow C$ with 100% confidence, then if one knows that A and B hold for some individual it is okay to learn C for that individual during the data mining process. Since this could be inferred from the result anyway, privacy is not compromised by revealing it during the *process* of data mining. Thus, there are two kinds of information leaks; the information leak from the function computed irrespective of the process used to compute the function and the information leak from the specific process of computing the function. Whatever is leaked from the function itself is unavoidable as long as the function has to be computed. In secure computation the second kind of leak is provably prevented. There is *no* information leak whatsoever due to the process. Some algorithms improve efficiency by trading off some security (leak a small amount of information). Even if this is allowed, the SMC style of proof provides a tight bound on the information leaked; allowing one to determine if the algorithm satisfies a privacy policy.

This leads to the primary proof technique used to demonstrate the security of privacy-preserving distributed data mining: a simulation argument. Given only its own input and the result, a party must be able to simulate what it sees during execution of the protocol. Note that "what it sees" is not constant: for example, what each party sees during the secure summation described above is dependent on the first party's choice of R. So the view to be simulated is actually a distribution. The formal definition for secure multiparty computation captures this: the *distribution* of values produced by the simulator for a given input and result must be equivalent to the distribution of values seen during real executions on the same input. The key challenge is to simulate this view *without knowledge* of the other party's input (and based only on the given party's input and output). The ability to simulate shows that the view of the party in a real protocol execution could actually have been generated by itself (without any interaction and just been given the output). Therefore, anything that the party can learn, it can learn from its input and output (because just running the simulation locally is equivalent to participating in a real protocol).

This is captured in the following definition (based on that of [17], however for readability we present it from the point of view of one party.)

Definition 1. *Privacy with respect to semi-honest behavior.*

Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ *be a probabilistic, polynomial-time functionality. Let* Π *be a two-party protocol for computing* f*.*

The view *of a party during an execution of* Π *on* (x, y)*, denoted* $VIEW^{\Pi}(x, y)$ *is* $(x, r, m_1, \ldots, m_t)$*, where* r *represents the outcome of the party's internal coin tosses, and* m_i *represents the* ith *message it has received. The final* output*s of the parties during an execution are denoted* $OUTPUT_1^{\Pi}(x, y)$ *and* $OUTPUT_2^{\Pi}(x, y)$*.*

Π *privately computes* f *if there exists a probabilistic polynomial time algorithm* S *such that*

$$\begin{aligned} &\{(S(x, f(x, y)),)\}_{x,y\in\{0,1\}^*} \\ &\quad \equiv^C \left\{\left(VIEW^{\Pi}(x, y), OUTPUT_2^{\Pi}(x, y)\right)\right\}_{x,y\in\{0,1\}^*} \end{aligned}$$

where $\equiv^C$ *denotes computational indistinguishability. Note that a party's own output is implicit in its view.*

The definition given above is restricted to two parties. The basic idea holds for extension to more than two parties. Reference [17] proved that this definition is essentially equivalent to the "trusted third party" definition, showing that any computation meeting this simulation argument in fact meets our intuitive expectations of security. A similar, but considerably more complex, definition exists for malicious adversaries. Because of the complexity, we will stick to the semi-honest definition. However, many applications require something stronger than semi-honest protocols. Intermediate definitions are possible (e.g., the secure association rules discussed at the beginning of this section is secure against malicious parties that do not collude), but formal frameworks for such definitions remain to be developed.

One key point is the restriction of the simulator to polynomial time algorithms, and that the views only need to be *computationally* indistinguishable. Algorithms meeting this definition need not be proof against an adversary capable of trying an exponential number of possibilities in a reasonable time frame. While some protocols (e.g., the secure sum described above) do not require this restriction, most make use of cryptographic techniques that are only secure against polynomial time adversaries. This is adequate in practice (as with cryptography); security parameters can be set to ensure that the computing resources to break the protocol in any reasonable time do not exist.

A second key contribution is the composition theorem of [17], stated informally here:

Theorem 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.

Informally, the theorem states that if a protocol is shown to be secure except for several invocations of sub-protocols, and if the sub-protocols themselves are proven to be secure, then the entire protocol is secure. The immediate consequence is that, with care, we can combine secure protocols to produce new secure protocols.

While the general circuit evaluation method has been proven secure by the above definition, it poses significant computational problems. Given the size and computational cost of data mining problems, representing algorithms as a boolean circuit results in unrealistically large circuits. The challenge of privacy-preserving distributed data mining is to develop algorithms that have reasonable computation and communication costs on real-world problems, and prove their security with respect to the above definition. While the secure circuit evaluation technique may be used within these algorithms, use must be limited to constrained sub-problems. For example, by adding secure comparison to the protocol at the beginning of this Section, the protocol can simply reveal if support and confidence exceed a threshold without revealing actual values. This mitigates the 100% confidence privacy compromise described above.

We now describe several techniques whose security properties have been evaluated using the standards described above. These examples demonstrate some key concepts that can be used to develop privacy-preserving distributed data mining algorithms, as well as demonstrating how algorithms are proven to be secure.

1.2 Association Rules

Association Rule mining is one of the most important data mining tools used in many real life applications. It is used to reveal unexpected relationships in the data. In this section, we will discuss the problem of computing association rules within a horizontally partitioned database framework. We assume homogeneous databases: All sites have the same schema, but each site has information on different entities. The goal is to produce association rules that hold globally, while limiting the information shared about each site.

The association rules mining problem can formally be defined as follows[2]: Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let DB be a set of transactions, where each transaction T is an itemset such that $T \subseteq I$. Given an itemset $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$ where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has *support* s in the transaction database DB if $s\%$ of transactions in DB contain $X \cup Y$. The association rule holds in the transaction database DB

with *confidence* c if $c\%$ of transactions in DB that contain X also contains Y. An itemset X with k items is called a k-itemset. The problem of mining association rules is to find all rules whose support and confidence are higher than certain user specified minimum support and confidence.

Clearly, computing association rules without disclosing individual transactions is straightforward. We can compute the global support and confidence of an association rule $AB \Rightarrow C$ knowing only the local supports of AB and ABC, and the size of each database:

$$support_{AB\Rightarrow C} = \frac{\sum_{i=1}^{\#sites} support\_count_{ABC}(i)}{\sum_{i=1}^{sites} database\_size(i)}$$

$$support_{AB} = \frac{\sum_{i=1}^{\#sites} support\_count_{AB}(i)}{\sum_{i=1}^{sites} database\_size(i)}$$

$$confidence_{AB\Rightarrow C} = \frac{support_{AB\Rightarrow C}}{support_{AB}}$$

Note that this doesn't require sharing any individual transactions. and protects individual data privacy, but it does require that each site disclose what rules it supports, and how much it supports each potential global rule. What if this information is sensitive? Clearly, such an approach will not be secure under SMC definitions.

A trivial way to convert the above simple distributed method to a secure method in SMC model is to use secure summation and comparison methods to check whether threshold are satisfied for every potential itemset. For example, for every possible candidate 1-itemset, we can use the secure summation and comparison protocol to check whether the threshold is satisfied.

Figure 1 gives an example of testing if itemset ABC is globally supported. Each site first computes its local support for ABC, or specifically the number of itemsets by which its support exceeds the minimum support threshold (which may be negative). The parties then use the previously described secure summation algorithm (the first site adds a random to its local excess support, then passes it to the next site to add its excess support, etc.) The only change is the final step: the last site performs a secure comparison with the first site to see if the $sum \geq R$. In the example, $R + 0$ is passed to the second site, which adds its excess support (-4) and passes it to site 3. Site 3 adds its excess support; the resulting value (18) is tested using secure comparison to see if it exceeds the Random value (17). It is, so itemset ABC is supported globally.

Due to huge number of potential candidate itemsets, we need to have a more efficient method. This can be done by observing the following lemma: If a rule has $support > k\%$ globally, it must have $support > k\%$ on at least one of the individual sites. A distributed algorithm for this would work as follows: Request that each site send all rules with support at least k. For each

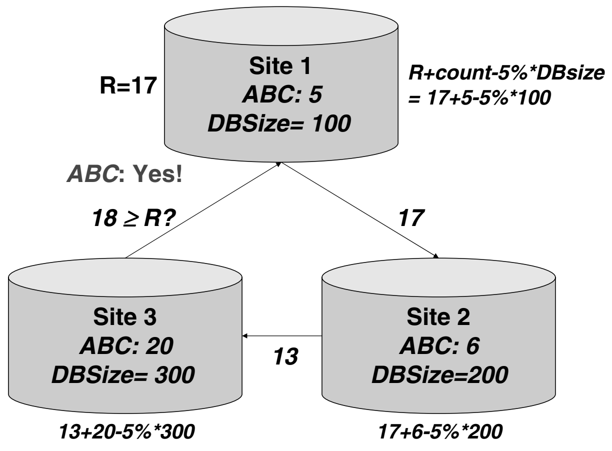


Fig. 1. Determining if itemset support exceeds 5% threshold

rule returned, request that all sites send the count of their transactions that support the rule, and the total count of all transactions at the site. From this, we can compute the global support of each rule, and (from the lemma) be certain that all rules with support at least k have been found. This has been shown to be an effective pruning technique[7]. In order to use the above lemma, we need to compute the union of locally large sets. We then use the secure summation and comparison only on the candidate itemsets contained in the union.

Revealing candidate itemsets means that the algorithm is no longer fully secure: itemsets that are large at one site, but not globally large, would not be disclosed by a fully secure algorithm. However, by computing the union *securely*, we prevent disclosure of which site, or even how many sites, support a particular itemset. This release of innocuous information (included in the final result) enables a completely secure algorithm that approaches the efficiency of insecure distributed association rule mining algorithms. The function now being computed reveals more information than the original association rule mining function. However, the key is that we have provable limits on what is disclosed. We now demonstrate how to securely compute a union.

Secure Union of Locally Large Itemsets

One way to securely compute a union is to directly apply secure circuit evaluation as follows: For each possible large k-itemset, each site can create a 0/1 vector such that if the ith itemset is locally supported at the site, it will set the ith bit of its vector to 1 otherwise it will set it to 0. Let's denote this vector as v_j for site j and let $v_j(i)$ be the ith bit of this vector. All the itemsets are arranged according to lexicographic order. Now for any given itemset, we can find its index i, and evaluate $\vee_{j=1}^{n} v_j(i)$ where $\vee$ is the or gate. Assuming that we use a secure generic circuit evaluation for or gate ($\vee$), the above protocol is secure and reveals nothing other than the set union result. However expensive circuit evaluation is needed for each potential large k-itemset. This secure method does not use the fact that local pruning eliminates some part of the large itemsets. We will now give a much more efficient method for this problem. Although the new method reveals a little more information than the above protocol, a precise description of what is revealed is given, and we prove that nothing else is revealed.

To obtain an efficient solution without revealing what each site supports, we instead exchange locally large itemsets in a way that obscures the source of each itemset. We assume a secure commutative encryption algorithm with negligible collision probability. Intuitively, under commutative encryption, the order of encryption does not matter. If a plaintext message is encrypted by two different keys in a different order, it will be mapped to the same ciphertext. Formally, commutativity ensures that $E_{k1}(E_{k2}(x)) = E_{k2}(E_{k1}(x))$. There are several examples of commutative encryption schemes, RSA is perhaps the best known.

The detailed algorithm is given in Algorithm 1. We now briefly explain the key phases of the algorithm. The main idea is that each site encrypts the locally supported itemsets, along with enough "fake" itemsets to hide the actual number supported. Each site then encrypts the itemsets from other sites. In Phases 2 and 3, the sets of encrypted itemsets are merged. Due to commutative encryption, duplicates in the locally supported itemsets will be duplicates in the encrypted itemsets, and can be deleted. The reason this occurs in two phases is that if a site knows which fully encrypted itemsets come from which sites, it can compute the size of the intersection between any set of sites. While generally innocuous, if it has this information for itself, it can guess at the itemsets supported by other sites. Permuting the order after encryption in Phase 1 prevents knowing exactly which itemsets match, however separately merging itemsets from odd and even sites in Phase 2 prevents any site from knowing the fully encrypted values of its own itemsets.

Phase 4 decrypts the merged frequent itemsets. Commutativity of encryption allows us to decrypt all itemsets in the same order regardless of the order they were encrypted in, preventing sites from tracking the source of each itemset.

The detailed algorithm assumes the following representations: F represents the data that can be used as fake itemsets. $|LLe_{i(k)}|$ represents the set of the encrypted k itemsets at site i. E_i is the encryption and D_i is the decryption by site i.

An illustration of the above protocol is given in Fig. 2. Using commutative encryption, each party encrypts its own frequent itemsets (e.g., Site 1 encrypts itemset C). The encrypted itemsets are then passed to other parties, until all parties have encrypted all itemsets. These are passed to a common party to eliminate duplicates, and to begin decryption. (In the figure, the full set of itemsets are shown to the left of Site 1, after Site 1 decrypts.) This set is then

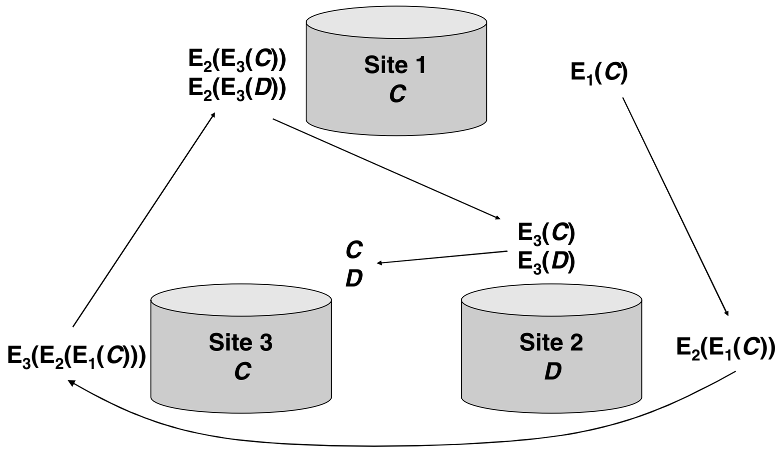


Fig. 2. Determining global candidate itemsets

passed to each party, and each party decrypts each itemset. The final result is the common itemsets (C and D in the figure).

Clearly, Algorithm 1 finds the union without revealing which itemset belongs to which site. It is not, however, secure under the definitions of secure multi-party computation. It reveals the number of itemsets having common support between sites, e.g., sites 3, 5, and 9 all support some itemset. It does not reveal *which* itemsets these are, but a truly secure computation (as good as giving all input to a "trusted party") could not reveal even this count. Allowing innocuous information leakage (the number of itemsets having common support) allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach.

If we deem leakage of the number of commonly supported itemsets as acceptable, we can prove that this method is secure under the definitions of secure multi-party computation. The idea behind the proof is to show that given the result, the leaked information, and a site's own input, a site can simulate everything else seen during the protocol. Since the simulation generates everything seen during execution of the protocol, the site clearly learns nothing new from the protocol beyond the input provided to the simulator. One key is that the simulator does not need to generate exactly what is seen in any particular run of the protocol. The exact content of messages passed during the protocol is dependent on the random choice of keys; the simulator must generate an equivalent distribution, based on random choices made by the simulator, to the distribution of messages seen in real executions of the protocol. A formal proof that this proof technique shows that a protocol preserves privacy can be found in [17]. We use this approach to prove that Algorithm 1 reveals only the union of locally large itemsets and a clearly bounded set of innocuous information.

Algorithm 1 [20] Finding secure union of large itemsets of size k

```
Require: $N \geq 3$ sites numbered $0..N-1$, set $F$ of non-itemsets.

  *Phase 0: Encryption of all the rules by all sites*
  for each site i do
    generate $LL_{i(k)}$ (Locally Large k-itemsets)
    $LLe_{i(k)} = \emptyset$
    for each $X \in LL_{i(k)}$ do
      $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(X)\}$
    end for
    for $j = |LLe_{i(k)}| + 1$ to $|CG_{(k)}|$ do
      $LLe_{i(k)} = LLe_{i(k)} \cup \{E_i(\text{random selection from } F)\}$
    end for
  end for

  *Phase 1: Encryption by all sites*
  for Round $j = 0$ to $N-1$ do
    if Round $j = 0$ then
      Each site $i$ sends permuted $LLe_{i(k)}$ to site $(i+1) \bmod N$
    else
      Each site $i$ encrypts all items in $LLe_{(i-j \bmod N)(k)}$ with $E_i$, permutes, and
      sends it to site $(i+1) \bmod N$
    end if
  end for{At the end of Phase 1, site $i$ has the itemsets of site $(i+1) \bmod N$
  encrypted by every site}

  *Phase 2: Merge odd/even itemsets*
  Each site $i$ sends $LLe_{i+1 \bmod N}$ to site $1 - ((i+1 \bmod N) \bmod 2)$
  Site 0 sets $RuleSet_1 = \cup_{j=1}^{\lceil (N-1)/2 \rceil} LLe_{(2j-1)(k)}$
  Site 1 sets $RuleSet_0 = \cup_{j=0}^{\lfloor (N-1)/2 \rfloor} LLe_{(2j)(k)}$

  *Phase 3: Merge all itemsets*
  Site 1 sends permuted $RuleSet_1$ to site 0
  Site 0 sets $RuleSet = RuleSet_0 \cup RuleSet_1$

  *Phase 4: Decryption*
  for $i = 0$ to $N-1$ do
    Site $i$ decrypts items in $RuleSet$ using $D_i$
    Site $i$ sends permuted $RuleSet$ to site $i+1 \bmod N$
  end for
  Site $N-1$ decrypts items in $RuleSet$ using $D_{N-1}$
  $RuleSet_{(k)} = RuleSet - F$
  Site $N-1$ broadcasts $RuleSet_{(k)}$ to sites $0..N-2$
```

Theorem 2. *[20] Algorithm 1 privately computes the union of the locally large itemsets assuming no collusion, revealing at most the result* $\cup_{i=1}^{N} LL_{i(k)}$ *and:*

1. *Size of intersection of locally supported itemsets between any subset of odd numbered sites,*
2. *Size of intersection of locally supported itemsets between any subset of even numbered sites, and*
3. *Number of itemsets supported by at least one odd and one even site.*

Proof. Phase 0: Since no communication occurs in Phase 0, each site can simulate its view by running the algorithm on its own input.

Phase 1: At the first step, each site sees $LLe_{i-1(k)}$. The size of this set is the size of the global candidate set $CG_{(k)}$, which is known to each site. Assuming the security of encryption, each item in this set is computationally indistinguishable from a number chosen from a uniform distribution. A site can therefore simulate the set using a uniform random number generator. This same argument holds for each subsequent round.

Phase 2: In Phase 2, site 0 gets the fully encrypted sets of itemsets from the other even sites. Assuming that each site knows the source of a received message, site 0 will know which fully encrypted set $LLe_{(k)}$ contains encrypted itemsets from which (odd) site. Equal itemsets will now be equal in encrypted form. Thus, site 0 learns if any odd sites had locally supported itemsets in common. We can still build a simulator for this view, using the information in point 2 above. If there are k itemsets known to be common among all $\lfloor N/2 \rfloor$ odd sites (from point 1), generate k random numbers and put them into the simulated $LLe_{i(k)}$. Repeat for each $\lfloor N/2 \rfloor - 1$ subset, etc., down to 2 subsets of the odd sites. Then fill each $LLe_{i(k)}$ with randomly chosen values until it reaches size $|CG_{i(k)}|$. The generated sets will have exactly the same combinations of common items as the real sets, and since the *values* of the items in the real sets are computationally indistinguishable from a uniform distribution, their simulation matches the real values.

The same argument holds for site 1, using information from point 2 to generate the simulator.

Phase 3: Site 1 eliminates duplicates from the $LLe_{i(k)}$ to generate $RuleSet_1$. We now demonstrate that Site 0 can simulate $RuleSet_1$. First, the size of $RuleSet_1$ can be simulated knowing point 2. There may be itemsets in common between $RuleSet_0$ and $RuleSet_1$. These can be simulated using point 3: If there are k items in common between even and odd sites, site 0 selects k random items from $RuleSet_0$ and inserts them into $RuleSet_1$. $RuleSet_1$ is then filled with randomly generated values. Since the encryption guarantees that the values are computationally indistinguishable from a uniform distribution, and the set sizes $|RuleSet_0|$, $|RuleSet_1|$, and $|RuleSet_0 \cap RuleSet_1|$ (and thus $|RuleSet|$) are identical in the simulation and real execution, this phase is secure.

Phase 4: Each site sees only the encrypted items after decryption by the preceding site. Some of these may be identical to items seen in Phase 2, but since all items must be in the union, this reveals nothing. The simulator for site i is built as follows: take the values generated in Phase 2 step $N-1-i$, and place them in the *RuleSet*. Then insert random values in *RuleSet* up to the proper size (calculated as in the simulator for Phase 3). The values we have not seen before are computationally indistinguishable from data from a uniform distribution, and the simulator includes the values we have seen (and knew would be there), so the simulated view is computationally indistinguishable from the real values.

The simulator for site $N-1$ is different, since it learns $RuleSet_{(k)}$. To simulate what it sees in Phase 4, site $N-1$ takes each item in $RuleSet_{(k)}$, the final result, and encrypts it with E_{N-1}. These are placed in *RuleSet*. *RuleSet* is then filled with items chosen from F, also encrypted with E_{N-1}. Since the choice of items from F is random in both the real and simulated execution, and the real items exactly match in the real and simulation, the *RuleSet* site $N-1$ receives in Phase 4 is computationally indistinguishable from the real execution.

Therefore, we can conclude that above protocol is privacy-preserving in the semi-honest model with the stated assumptions. ∎

The information disclosed by points 1–3 could be relaxed to the number of itemsets support by 1 site, 2 sites, ..., N sites if we assume anonymous message transmission. The number of jointly supported itemsets can also be masked by allowing sites to inject itemsets that are not really supported locally. These fake itemsets will simply fail to be globally supported, and will be filtered from the final result when global support is calculated as shown in the next section. The jointly supported itemsets "leak" then becomes an upper bound rather than exact, at an increased cost in the number of candidates that must be checked for global support. While not truly zero-knowledge, it reduces the confidence (and usefulness) of the leaked knowledge of the number of jointly supported itemsets. In practical terms, revealing the size (but not content) of intersections between sites is likely to be of little concern.

A complimentary problem of mining association rules over vertically partitioned data is addressed in [34, 37]. While we do not describe these techniques here, we would like to emphasize that the different model of distribution requires very different solution techniques.

1.3 Decision Trees

The first paper discussing the use of Secure Multiparty Computation for data mining gave a procedure for constructing decision trees[24], specifically running ID3 [31] between two parties, each containing a subset of the training entities. Of particular interest is the ability to maintain "perfect" security in the SMC sense, while trading off efficiency for the quality of the resulting decision tree.

Building an ID3 decision tree is a recursive process, operating on the decision attributes R, class attribute C, and training entities T. At each stage, one of three things can happen:

1. R may be empty; i.e., the algorithm has no attributes on which to make a choice. In this case a leaf node is created with the class of the leaf being the majority class of the transactions in T.
2. All the transactions in T may have the same class c, in which case a leaf is created with class c.
3. Otherwise, we recurse:
   (a) Find the attribute A that is the most effective classifier for transactions in T, specifically the attribute that gives the highest information gain.
   (b) Partition T based on the values a_i of A.
   (c) Return a tree with root labeled A and edges a_i, with the node at the end of edge a_i constructed from calling ID3 with $R - \{A\}, C, T(A_i)$.

In step 3a, *information gain* is defined as the change in the entropy relative to the class attribute. Specifically, the entropy

$$H_C(T) = \sum_{c \in C} -\frac{|T(c)|}{|T|} \log \frac{|T(c)|}{|T|}.$$

Analogously, the entropy after classifying with A is

$$H_C(T|A) = \sum_{a \in A} -\frac{|T(a)|}{|T|} H_C(T(a)).$$

Information gain is

$$Gain(A) \overset{def}{=} H_C(T) - H_C(T|A).$$

The goal, then, is to find A that maximizes $Gain(A)$, or minimizes $H_C(T|A)$. Expanding, we get:

$$\begin{aligned} H_C(T|A) &= \sum_{a \in A} \frac{|T(a)|}{|T|} H_C(T(A)) \\ &= \frac{1}{|T|} \sum_{a \in A} |T(a)| \sum_{c \in C} -\frac{|T(a,c)|}{|T(a)|} \log \left(\frac{|T(a,c)|}{|T(A)|} \right) \\ &= \frac{1}{|T|} \left(-\sum_{a \in A} \sum_{c \in C} |T(a,c)| \log(|T(a,c)|) + \sum_{a \in A} |T(a)| \log(|T(a)|) \right) \end{aligned} \tag{1}$$

Looking at this from the point of view of privacy preservation, we can assume that R and C are known to both parties. T is divided. In Step 1 we need only determine the class value of the majority of the transactions in

T. This can be done using circuit evaluation 1. Since each party is able to compute the count of local items in each class, the input size of the circuit is fixed by the number of classes, rather than growing with the (much larger) training data set size.

Step 2 requires only that we determine if all of the items are of the same class. This can again be done with circuit evaluation, here testing for equality. Each party gives as input either the single class c_i of all of its remaining items, or the special symbol $\perp$ if its items are of multiple classes. The circuit returns the input if the input values are equal, else it returns $\perp$.[1]

It is easy to prove that these two steps preserve privacy: Knowing the tree, we know the majority class for Step 1. As for Step 2, if we see a tree that has a "pruned" branch, we know that all items must be of the same class, or else the branch would have continued. Interestingly, if we test if all items are in the same class before testing if there are no more attributes (reversing steps 1 and 2, as the original ID3 algorithm was written), the algorithm would not be private. The problem is that Step 2 reveals if all of the items are of the same class. The decision tree doesn't contain this information. However, if a branch is "pruned" (the tree outputs the class without looking at all the attributes), we know that all the training data at that point are of the same class – otherwise the tree would have another split/level. Thus Step 2 doesn't reveal any knowledge that can't be inferred from the tree *when the tree is pruned* – the given order ensures that this step will only be taken if pruning is possible.

This leaves Step 3. Note that once A is known, steps 3b and 3c can be computed locally – no information exchange is required, so no privacy breach can occur. Since A can be determined by looking at the result tree, revealing A is not a problem, provided nothing but the proper choice for A is revealed. The hard part is Step 3a: computing the attribute that gives the highest information gain. This comes down to finding the A that minimizes (1).

Note that since the database is horizontally partitioned, $|T(a)|$ is really $|T_1(a)| + |T_2(a)|$, where T_1 and T_2 are the two databases. The idea is that the parties will compute (random) shares of each $(|T_1(a, c)| + |T_2(a, c)|) \log(|T_1(a, c)|+|T_2(a, c)|)$, and $(|T_1(a)|+|T_2(a)|) \log(|T_1(a)|+|T_2(a)|)$. The parties can then locally add their shares to give each a random share of $H_C(T|A)$. This is repeated for each attribute A, and a (small) circuit, of size linear in the number of attributes, is constructed to select the A that gives the largest value.

The problem, then is to efficiently compute $(x + y) \log(x + y)$. Lindell and Pinkas actually give a protocol for computing $(x+y) \ln(x+y)$, giving shares of $H_C(T|A) \cdot |T| \cdot \ln 2$. However, the constant factors are immaterial since the goal is simply to find the A that minimizes the equation. In [24] three protocols are

[1]The paper by Lindell and Pinkas gives other methods for computing this step, however circuit evaluation is sufficient – the readers are encouraged to read [24] for the details.

given: Computing shares of $\ln(x{+}y)$, computing shares of $x{\cdot}y$, and the protocol for computing the final result. The last is straightforward: Given shares u_1 and u_2 of $\ln(x+y)$, the parties call the multiplication protocol twice to give shares of $u_1 \cdot y$ and $u_2 \cdot x$. Each party then sums three multiplications: the two secure multiplications, and the result of multiplying its input (x or y) with its share of the logarithm. This gives each shares of $u_1 y u_2 x + u_1 x + u_2 y = (x+y)(u_1+u_2) = (x+y)\ln(x+y)$.

The logarithm and multiplication protocols are based on oblivious polynomial evaluation[27]. The idea of oblivious polynomial evaluation is that one party has a polynomial P, the other has a value for x, and the party holding x obtains $P(x)$ without learning P or revealing x. Given this, the multiplication protocol is simple: The first party chooses a random r and generates the polynomial $P(y) = xy - r$. The resulting of evaluating this on y is the second party's share: $xy - r$. The first party's share is simply r.

The challenge is computing shares of $\ln(x{+}y)$. The trick is to approximate $\ln(x+y)$ with a polynomial, specifically the Taylor series:

$$\ln(1+\epsilon) = \sum_{i=1}^{k} \frac{(-1)^{i-1}\epsilon^i}{i}$$

Let 2^n be the closest power of 2 to $(x+y)$. Then $(x+y) = 2^n(1+\epsilon)$ for some $-1/2 \leq \epsilon \leq 1/2$. Now

$$\ln(x) = \ln(2^n(1+\epsilon)) = n\ln 2 + \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \ldots$$

We determine shares of $2^N n \ln 2$ and $2^N \epsilon$ (where N is an upper bound on n) using circuit evaluation. This is a simple circuit. $\epsilon \cdot 2^n = (x+y) - 2^n$, and n is obtained by inspecting the two most significant bits of $(x+y)$. There are a small (logarithmic in the database size) number of possibilities for $2^N n \ln 2$, and $\epsilon \cdot 2^N$ is obtained by left shifting $\epsilon \cdot 2^n$.

Assume the parties share of $2^N n \ln 2$ are α_1 and α_2, and the shares of $2^N\epsilon$ are β_1 and β_2. The first party defines

$$P(x) = \sum_{i=1}^{k} \frac{(-1)^{i-1}}{2^{N(i-1)}} \frac{(\alpha_1 + x)^i}{i} - r$$

and defines it's share $u_1 = \beta_1 + r$. The second party defines its share as $\beta_2 + P(\alpha_2)$. Note that $P(\alpha_2)$ computes the Taylor series approximation times 2^N, minus the random r. Since 2_N is public, it is easily divided out later, so the parties do get random shares of an approximation of $\ln(x+y)$.

As discussed in Sect. 1.2, all arithmetic is really done over a sufficiently large field, so that the random values (e.g., shares) can be chosen from a uniform distribution. In addition, the values in the Taylor series are multiplied by the least common multiple of $2, \ldots, k$ to eliminate fractions.

The key points to remember are the use of oblivious polynomial evaluation, and the use of an efficiently computable (bounded) approximation when efficiently and privately computing the real value is difficult.

There has also been a solution for constructing ID3 on vertically partitioned data[12]. This work assumes the class of the training data is shared, but some the attributes are private. Thus most steps can be evaluated locally. The main problem is computing which site has the best attribute to split on – each can compute the gain of their own attributes without reference to the other site.

1.4 Third Party Based Solutions

The use of an outside party often enables more efficient solutions to secure computations. The key issues are what level of trust is placed in this third, outside party; and what level of effort is required of the third party. Generally the trust issue is rooted in collusion: What happens if parties collude to violate privacy? This gives us a hierarchy of types of protocols:

No trusted third party. The most general type of protocol meets the strong statements of Definition 1: No party learns anything beyond what it can infer from the results. If parties collude, they are treated as one from the point of view of the definition: What can they infer from their combined inputs and results?

Non-colluding untrusted third party protocol. These protocols allow all parties to utilize an untrusted third party to do part of the computation. The third party learns nothing by itself (it need not even see the results). Provided this third party does not collude with one or more of the other parties, this method preserves privacy as well as a fully secure protocol.
Typically data is sent to this party in some "encrypted" form such that it cannot make any sense of the data by itself. This party performs some computations and replies to the local parties, which then remove the effect of the encryption to get back the final result. The key is that the untrusted third party does not see any "cleartext" data and is assumed to not collude with any of the other parties.

Commodity server protocol. Commodity server protocols also requires a non-colluding third party. They differ from non-colluding untrusted third party protocols in that only one way communication is allowed from the commodity server to the other parties. Because of this, the commodity server clearly learns nothing (absent collusion). The general approach is to use the commodity server to generate complementary data (e.g., public-private encryption key pairs), each part of which is given to a different party.
The commodity server model has been proven to be powerful enough to do all secure computation[5]. Thus, in terms of scope and power commodity server protocols are equivalent to protocols with an untrusted third party.

They are simpler to prove secure though typically more complicated than untrusted third party protocols.

Trusted third party protocol. The gold standard for security is the assumption that we have a trusted third party to whom we can give all data. The third party performs the computation and delivers only the results – except for the third party, it is clear that nobody learns anything not inferable from its own input and the results. The goal of secure protocols is to reach this same level of privacy preservation, without the (potentially insoluble) problem of finding a third party that everyone trusts.

The complexity of fully secure protocols generally increases with an increasing number of parties. Simple (completely secure) solutions for two parties do not extend easily to more than two parties. In such scenarios, it is often worthwhile to reduce the single complex problem to a series of two-party sub-problems. One approach is to make use of untrusted non-colluding third party protocols, using some of the participating parties to serve as "untrusted" parties for other parties in the protocol. If the target function consists of additive sub-blocks, or the target function can be reduced to a combination of associative functions, such an approach is possible. The key is to find an untrusted third party solution to the two-party problem, then securely combine the two-party results in a way that gives the desired final result.

We now give a couple of examples typifying these cases, and show solutions that illuminate the basic concept. First consider the following geometric function: Consider an n-dimensional space split between r different parties, $P_1, \ldots, P_r$. P_i owns a variable number n_i of dimensions/axes such that $\sum_{i=1}^{r} n_i = n$. A point X in this n-dimensional space would have its n co-ordinates split between the r parties. Thus, party i would know n_i of the co-ordinates of the point X. We assume that there is some way of linking the co-ordinates of the same point together across all the parties (i.e., a join key). Now, assume there are k points $Y_1, \ldots, Y_k$ split between the r parties. The target function is to jointly compute the index i of the point Y_i that is the "closest" to point X according to some distance metric $\mathcal{D}$.

Why is this problem interesting? K-means clustering over vertically partitioned data can be easily reduced to this problem. K-means clustering is an iterative procedure that starts with K arbitrary cluster means. In each iteration, all of the points are assigned to the current closest cluster (based on distance from mean). Once all the points are assigned, the cluster means are recomputed based on the points assigned to each cluster. This procedure is repeated until the clusters converge. One easy convergence condition is to stop when the difference between the old means and the new means is sufficiently small. The key step, assigning a point to a cluster, is done by finding the closest cluster to the point. This is solved by our earlier "abstract" geometrical problem.[2]

[2] A solution for clustering in horizontally partitioned data has also been developed [23], this relies heavily on secure summation.

One possible distance metric is the Minkowski distance, d_M. The Minkowski distance d_M between two points X and Y is defined as

$$d_M = \left\{ \sum_{i=1}^{n} (x_i - y_i)^m \right\}^{\frac{1}{m}}$$

Note that $m = 2$ gives the Euclidean distance, while $m = 1$ gives the Manhattan distance. Instead of comparing two distances, we get the same result by comparing the mth power of the distances. Note that if we do not take the mth root, the target function is additive. We can exploit this additiveness to get an efficient protocol.

The problem is formally defined as follows. Consider r parties $P_1, \ldots, P_r$, each with their own k-element vector $\mathbf{X_i}$:

$$P_1 \text{ has } \mathbf{X_1} = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{k1} \end{bmatrix}, P_2 \text{ has } \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{k2} \end{bmatrix}, \ldots, P_r \text{ has } \begin{bmatrix} x_{1r} \\ x_{2r} \\ \vdots \\ x_{kr} \end{bmatrix}.$$

The goal is to compute the index l that represents the row with the minimum sum. Formally, find

$$\underset{i=1..k}{argmin} \left(\sum_{j=1..r} x_{ij} \right)$$

For use in k-means clustering, $x_{ij} = |\mu_{ij} - point_j|$, or site P_j's component of the distance between a point and the cluster i with mean μ_i.

The security of the algorithm is based on three key ideas.

1. Disguise the site components of the distance with random values that cancel out when combined.
2. Compare distances so only the comparison result is learned; no party knows the distances being compared.
3. Permute the order of clusters so the real meaning of the comparison results is unknown.

The algorithm also requires three non-colluding sites. These parties may be among the parties holding data, but could be external as well. They need only know the number of sites r and the number of clusters k. Assuming they do not collude with each other, they learn nothing from the algorithm. For simplicity of presentation, we will assume the non-colluding sites are P_1, P_2, and P_r among the data holders.

The algorithm proceeds as follows. Site P_1 generates a length k random vector $\mathbf{V_i}$ for each site i, such that $\sum_{i=1}^{r} \mathbf{V_i} = \mathbf{0}$. P_1 also chooses a permutation π of $1..k$. P_1 then engages each site P_i in the permutation algorithm (see Sect. 1.4) to generate the sum of the vector $\mathbf{V_i}$ and P_i's distances $\mathbf{X_i}$. The resulting vector is known only to P_i, and is permuted by π known only to P_1, i.e., P_i has $\pi(\mathbf{V_i} + \mathbf{X_i})$, but does not know π or $\mathbf{V_i}$. P_1 and $P_3 \ldots P_{r-1}$ send their vectors to P_r.

Sites P_2 and P_r now engage in a series of secure addition/comparisons to find the (permuted) index of the minimum distance. Specifically, they want to find if $\sum_{i=1}^{r} x_{li} + v_{li} < \sum_{i=1}^{r} x_{mi} + v_{mi}$. Since $\forall l, \sum_{i=1}^{r} v_{li} = 0$, the result is $\sum_{i=1}^{r} x_{li} < \sum_{i=1}^{r} x_{mi}$, showing which cluster (l or m) is closest to the point. P_r has all components of the sum except $\mathbf{X_2} + \mathbf{V_2}$. For each comparison, we use a secure circuit evaluation that calculates $a_2 + a_r < b_2 + b_r$, without disclosing anything but the comparison result. After $k-1$ such comparisons, keeping the minimum each time, the minimum cluster is known.

P_2 and P_r now know the minimum cluster in the permutation π. They do not know the real cluster it corresponds to (or the cluster that corresponds to any of the others items in the comparisons.) For this, they send the minimum i back to site P_1. P_1 broadcasts the result $\pi^{-1}(i)$, the proper cluster for the point.

Algorithm 2 reproduces the full algorithm from [36]. We now describe the two key building blocks borrowed from the Secure Multiparty Computation literature. The secure addition and comparison consists of a circuit that has two inputs from each party, sums the first input of both parties and the second input of both parties, and returns the result of comparing the two sums. This (simple) circuit is evaluated securely using the generic algorithm. Though the generic algorithm is impractical for large inputs and many parties, it is quite efficient for a limited number invocations of the *secure_add_and_compare* function. For two parties, the message cost is $O(circuit\_size)$, and the number of rounds is constant. We can add and compare numbers with $O(m = \log(number\_of\_entities))$ bits using an $O(m)$ size circuit. A graphical depiction of stages 1 and 2 is given in Figs. 3(a) and 3(b).

We now give the permutation algorithm of [11], which simultaneously computes a vector sum and permutes the order of the elements in the vector.

The permutation problem is an asymmetric two party algorithm, formally defined as follows. There exist 2 parties, A and B. B has an n-dimensional vector $\mathbf{X} = (x_1, \ldots, x_n)$, and A has an n-dimensional vector $\mathbf{V} = (v_1, \ldots, v_n)$. A also has a permutation π of the n numbers. The goal is to give B the result $\pi(\mathbf{X} + \mathbf{V})$, without disclosing anything else. In particular, neither A nor B can learn the other's vector, and B does not learn π. For our purposes, the $\mathbf{V}$ is a vector of random numbers from a uniform random distribution, used to hide the permutation of the other vector.

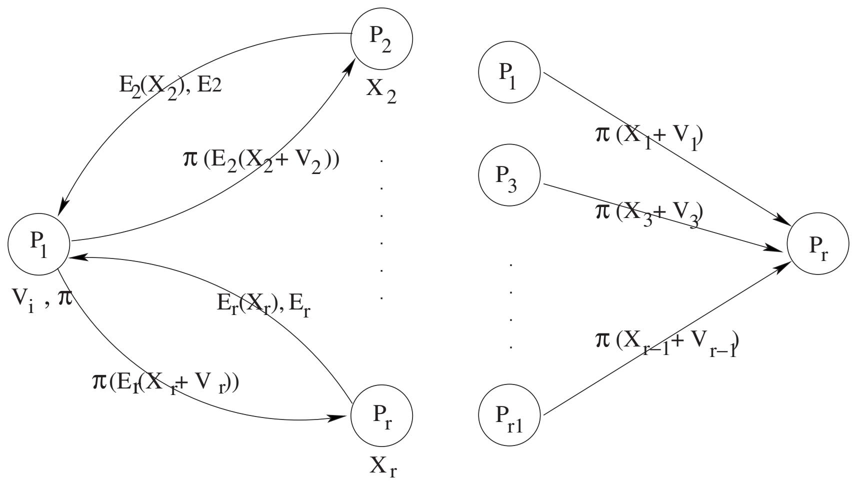
The solution makes use of a tool known as *Homomorphic Encryption*. An encryption function $\mathcal{H} : \mathcal{R} \to \mathcal{S}$ is called *additively homomorphic* if there is an efficient algorithm *Plus* to compute $H(x + y)$ from $H(x)$ and $H(y)$ that does not reveal x or y. Many such systems exist; examples include systems by [6, 26, 28], and [30]. This allows us to perform addition of encrypted data without decrypting it.

Algorithm 2 *closest_cluster*[36]: Find minimum distance cluster

```
Require: $r$ parties, each with a length $k$ vector $\mathbf{X}$ of distances. Three of these
    parties (trusted not to collude) are labeled $P_1$, $P_2$, and $P_r$.
 1: {Stage 1: Between $P_1$ and all other parties}
 2: $P_1$ generates $r$ random vectors $\mathbf{V_i}$ summing to $\mathbf{0}$ (see Algorithm 3).
 3: $P_1$ generates a random permutation $\pi$ over $k$ elements
 4: for all $i = 2 \ldots r$ do
 5:   $\mathbf{T_i}$ (at $P_i$) $= add\_and\_permute(\mathbf{V_i}, \pi(\text{at } P_1), \mathbf{X_i}(\text{at } P_i))$ {This is the permu-
      tation algorithm described in Sect. 1.4}
 6: end for
 7: $P_1$ computes $\mathbf{T_1} = \pi(\mathbf{X_1} + \mathbf{V_1})$
 8:
 9: {Stage 2: Between all but $P_2$ and $P_r$}
10: for all $i = 1, 3 \ldots r - 1$ do
11:   $P_i$ sends $\mathbf{T_i}$ to $P_r$
12: end for
13: $P_r$ computes $\mathbf{Y} = \mathbf{T_1} + \sum_{i=3}^{r} \mathbf{T_i}$
14:
15: {Stage 3: Involves only $P_2$ and $P_r$}
16: $minimal \leftarrow 1$
17: for j=2..k do
18:   if $secure\_add\_and\_compare(Y_j + T_{2j} < Y_{minimal} + T_{2minimal})$ then
19:     $minimal \leftarrow j$
20:   end if
21: end for
22:
23: {Stage 4: Between $P_r$ (or $P_2$) and $P_1$}
24: Party $P_r$ sends $minimal$ to $P_1$
25: $P_1$ broadcasts the result $\pi^{-1}(minimal)$
```

Algorithm 3 genRandom[36]: Generates a (somewhat) random matrix $V_{k \times r}$

```
Require: Random number generator $rand$ producing values uniformly distributed
    over $0..n-1$ spanning (at least) the domain of the distance function $-_D$.
Ensure: The sum of the resulting vectors is $\mathbf{0}$.
1: for all i = 1 ... k do
2:   $PartSum_i \leftarrow 0$
3:   for j = 2 ... r do
4:     $V_{ij} \leftarrow rand()$
5:     $PartSum_i \leftarrow PartSum_i + V_{ij} \pmod{n}$
6:   end for
7:   $V_{i1} \leftarrow -PartSum_i \pmod{n}$
8: end for
```



(a) Stage 1 (b) Stage 1

Fig. 3. Closest Cluster Computation

The permutation algorithm consists of the following steps:

1. B generates a public-private keypair (E_k, D_k) for a homomorphic encryption scheme.
2. B encrypts its vector $\mathbf{X}$ to generate the encrypted vector $\mathbf{X'} = (x'_1, \dots, x'_n)$, $x'_i = E_k(x_i)$.
3. B sends $\mathbf{X'}$ and the public key E_k to A.
4. A encrypts its vector $\mathbf{V}$ generating the encrypted vector $\mathbf{V'} = (v'_1, \dots, v'_n)$, $v'_i = E_k(v_i)$.
5. A now multiplies the components of the vectors $\mathbf{X'}$ and $\mathbf{V'}$ to get $\mathbf{T'} = (t'_1, \dots, t'_n)$, $t'_i = x'_i * v'_i$.
   Due to the homomorphic property of the encryption,
   $$x'_i * v'_i = E_k(x_i) * E_k(v_i) = E_k(x_i + v_i)$$
   so $\mathbf{T'} = (t'_1, \dots, t'_n), t'_i = E_k(x_i + v_i)$.
6. A applies the permutation π to the vector $\mathbf{T'}$ to get $\mathbf{T'_p} = \pi(\mathbf{T'})$, and sends $\mathbf{T'_p}$ to B.
7. B decrypts the components of $\mathbf{T'_p}$ giving the final result $\mathbf{T_p} = (t_{p_1}, \dots, t_{p_n})$, $t_{p_i} = x_{p_i} + v_{p_i}$.

Intersection

We now give an algorithm that demonstrates another way of using untrusted, non-colluding third parties. The specific algorithm is for computing the size of set intersection. This is useful for finding association rules in vertically partitioned data.

Assume the database is a boolean matrix where a 1 in cell (i, j) represents that the ith transaction contains feature j. The TID-list representation of the database has a transaction identifier list associated with every feature/attribute. This TID-list contains the identifiers of transactions that contain the attribute. Now, to count a frequency of an itemset $<AB>$ (where A and B are at different sites), it is necessary to count the number of transactions having both A and B. If we intersect the TID-lists for both A and B, we get the transactions containing both A and B. The size of this intersection set gives the (in)frequency of the itemset. Thus, the frequency of an itemset can be computed by securely computing the size of the intersection set of the TID-lists.

In addition to being useful for association rule mining, the algorithm illuminates a general technique that we can use to extend untrusted third party solutions for two parties to multiple parties. This works whenever the target function is associative. As in Sect. 1.2, the approach leaks some innocuous information, but can be proven to leak no more than this information. We start with the general technique, then discuss the specific application to set intersection.

Given k parties, the goal is to compute a function $y \in F_g$, where $y = f(x_1, \ldots, x_k)$, where $x_1, \ldots, x_k$ are the local inputs of the k parties. If the function can be decomposed into smaller invocations of an associative function, we can rewrite $y = x_1 \otimes x_2 \otimes \cdots \otimes x_k$. If we have a protocol f_s to securely compute the two-input function $\otimes$, we can construct a protocol to compute y as follows.

The key idea is to create two partitions P_0 and P_1. Split the k parties equally into the two partitions. We can now use the parties in partition P_i as untrusted third parties to evaluate partition P_{1-i}. To visualize this, construct a binary tree on the partition P_i with the leaves being the parties in P_i (Fig. 4).[3] There can be at most $|P_i| - 1$ interior nodes in the binary tree. Due to the (almost) equi-partitioning, the following invariant always holds: $|P_{1-i}| \geq |P_i| - 1$, for both values of i. Thus, there are sufficient parties in the other partition to act as interior nodes. The role of the parties in partition P_{1-i} is to act as the commodity server or untrusted third party for the parties in partition P_i.

In the first round, the $k/4$ of the parties from the other partition act as third parties for the $k/2$ parties in the first partition. For the remaining $\log k/2 - 1$ rounds the other $k/4$ parties of the 2nd partition act as third parties upwards along the tree. Each third party receives some form of the intermediate result, and utilizes it in the next round. It is important to analyze the amount of data revealed to the third party at this point and modify the protocol if necessary to limit the information disclosure. The entire process is illustrated in Fig. 4, where we show the process for partition P_0 consisting

[3]While the example assumes k is a power of 2, a proper assignment of parties to partitions is also possible if the tree is not complete. This is described in [37].

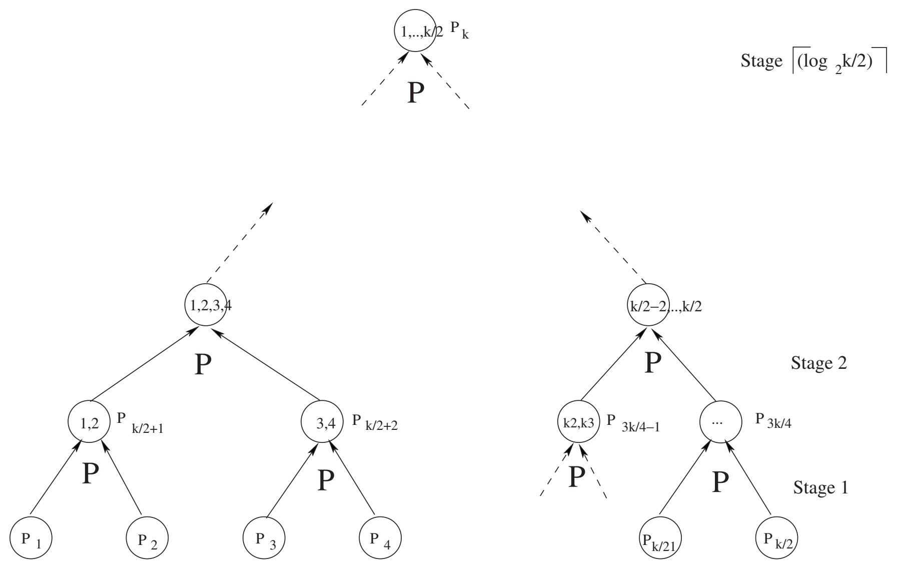


Fig. 4. The general protocol process applied on partition P_0

of the first $k/2$ parties. Thus, all of the parties $P_{k/2+1}, \ldots, P_k$ act as third parties/commodity servers in a single call to the protocol f_s when applied to the parties at the leaf nodes. There are a total of $\log k/2$ rounds in which several calls to the protocol f_s are made in parallel.

Once a similar process is done for the other partition P_1, the two topmost representatives of the two parties use a secure two party protocol f' to compute the final result. Every party possibly acquires some information about a few of the other parties, which goes against the precept of secure multi-party computation. But as long as the information revealed is held within strict (and provable) bounds, it is often worthwhile to trade this limited information disclosure for efficiency and practicality.

We summarize the use of this approach to solve secure set intersection [35, 37]. The problem is defined as follows. There are k parties, $P_1, \ldots, P_k$, each with a local set S_k drawn from a common (global) universe U. They wish to compute $| \cap_{j=1}^{k} S_j|$, i.e., the cardinality of the common intersection set. This is useful for several applications for example data mining association rules (see [37] for details.)

We now outline a two party protocol $f_\cap$ using a third untrusted party to compute $|S_a \cap S_b|$ for two parties A and B. The key idea behind protocol $f_\cap$ is to use commutative encryption (as described in Sect. 1.2) to allow comparing items without revealing them. Parties A and B generate encryption keys E_a and E_b respectively. A encrypts the items in its set S_a with E_a and sends them to B. Similarly, B encrypts the items in S_b with E_b and sends them to A. Each site now encrypts the received items with its own key, and sends the doubly-encrypted sets S'_a and S'_b to U. U now finds the intersection of these two sets. Because of commutativity of the encryption, an item $x \in S_a \cap S_b$ will correspond to an item $E_a(E_b(x)) = E_b(E_a(x))$ that appears in both S'_a and S'_b. Therefore, the size of the intersection $|S'_a \cap S'_b| = |S_a \cap S_b|$. Thus U learns the size of the intersection, but learns nothing about the items *in* the intersection.

Extending this to more than two parties is simple. We use the tree based evaluation for each partition. The lowest layer (consisting of leaves) proceeds as above. At the higher layers, the parties encrypt with the keys of their sibling's children. Since a party never sees any of the values from the sibling's children (even after encryption), knowing the keys gives no information. More details are given in [37].

2 Privacy Preservation through Noise Addition

The other approach to privacy-preserving data mining is based on adding random noise to the data, then providing the noisy dataset as input to the data mining algorithm. The privacy-preserving properties are a result of the noise: Data values for individual entities are distorted, and thus individually identifiable (private) values are not revealed. An example would be a survey: A company wishes to mine data from a survey of private data values. While the respondents may be unwilling to provide those data values directly, they would be willing to provide randomized/distorted results.

What makes this work interesting is how the mining of the noisy data set is done. Naïvely running a data mining algorithm on the data may work – for example, adding noise from a gaussian distribution centered at 0 will preserve averages – but does not always give good results. However, using knowledge of how the noise was generated enables us to do better. In particular, what is used is knowledge of the distribution that the noise came from (e.g., uniform or gaussian and the appropriate parameters). Knowing the distribution the random values came from does not reveal the specific values used to mask each entity, so privacy is still preserved. However, as we shall see the knowledge of the distribution of the noise does enable us to improve data mining results.

The problem addressed in [3] was building decision trees. If we return to the description of ID3 in Sect. 1.3, we see that Steps 1 and 3c do not reference the (noisy) data. Step 2 references only the class data, which is assumed to be known (for example, the survey may be demographics of existing customers – the company already knows which are high-value customers, and wants to know what demographics correspond to high-value customers.)

This leaves Steps 3a and 3b: Finding the attribute with the maximum information gain and partitioning the tree based on that attribute. Looking

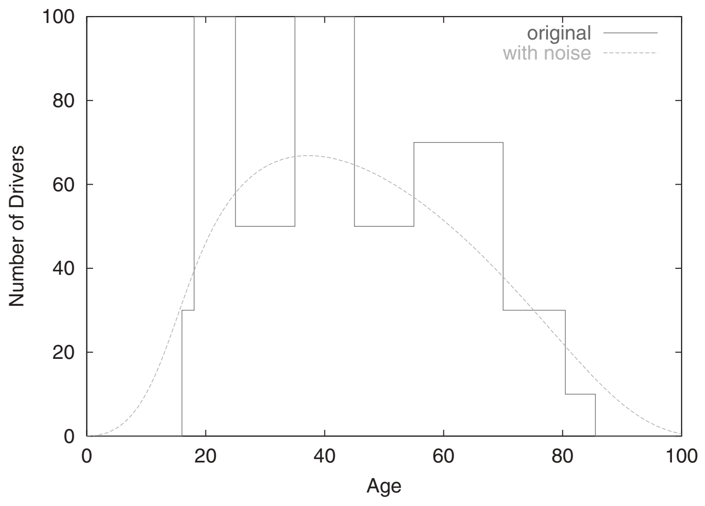


Fig. 5. Original distribution vs. distribution after random noise addition

at (1), the only thing needed is $|T(a, c)|$ and $|T(a)|$.[4] $|T(a)|$ requires partitioning the entities based on the attribute value, exactly what is needed for Step 3b. The problem is that the attribute values are modified, so we don't know which entity really belongs in which partition.

Figure 5 demonstrates this problem graphically. There are clearly peaks in the number of drivers under 25 and in the 25–35 age range, but this doesn't hold in the noisy data. The ID3 partitioning should reflect the peaks in the data.

A second problem comes from the fact that the data is assumed to be ordered (otherwise "adding" noise makes no sense.) As a result, where to divide partitions is not obvious (as opposed to categorical data). Again, reconstructing the distribution can help. We can see that in Fig. 5 partitioning the data at ages 30 and 50 would make sense – there is a natural "break" in the data at those points anyway. However, we can only see this from the actual distribution. The split points are not obvious in the noisy data.

Both these problems can be solved if we know the distribution of the original data, even if we do not know the original values. The problem remains that we may not get the *right* entities in each partition, but we are likely to get enough that the statistics on the class of each partition will still hold (In [3] experimental results are given demonstrating this fact.)

[4]Reference [3] actually uses the gini coefficient rather than information gain. While this may affect the quality of the decision tree, it has no impact on the discussion here. We stay with information gain for simplicity.

What remains is the problem of estimating the distribution of the real data (X) given the noisy data (w) and the distribution of the noise (Y). This is accomplished through Bayes' rule:

$$\begin{aligned}
F'_X(a) &\equiv \int_{-\infty}^{a} f_X(z|X+Y=w)dz \\
&= \int_{-\infty}^{a} \frac{f_{X+Y}(w|X=z)f_X(z)}{f_{X+Y}(w)}dz \\
&= \int_{-\infty}^{a} \frac{f_{X+Y}(w|X=z)f_X(z)}{\int_{-\infty}^{\infty} f_{X+Y}(w|X=z')f_X(z')dz'}dz \\
&= \frac{\int_{-\infty}^{a} f_{X+Y}(w|X=z)f_X(z)dz}{\int_{-\infty}^{\infty} f_{X+Y}(w|X=z)f_X(z)dz} \\
&= \frac{\int_{-\infty}^{a} f_Y(w-z)f_X(z)dz}{\int_{-\infty}^{\infty} f_Y(w-z)f_X(z)dz}
\end{aligned}$$

Given the actual data values $w_i = x_i + y_i$, we use this to estimate the distribution function as follows:

$$F'_X(a) = \frac{1}{n}\sum_{i=1}^{n} F'_{X_i} = \frac{1}{n}\sum_{i=1}^{n} \frac{\int_{-\infty}^{a} f_Y(w_i-z)f_X(z)dz}{\int_{-\infty}^{\infty} f_Y(w_i-z)f_X(z)dz}$$

Differentiating gives us the posterior density function:

$$f'_X(a) = \frac{1}{n}\sum_{i=1}^{n} \frac{f_Y(w_i-a)f_X(a)}{\int_{-\infty}^{\infty} f_Y(w_i-z)f_X(z)dz} \tag{2}$$

The only problem is, we don't know the real density function f_X. However, starting with an assumption of a uniform distribution, we can use (2) to iteratively refine the density function estimate, converging on an estimate of the real distribution for X.

In [3] several optimizations are given, for example partitioning the data to convert the integration into sums. They also discuss tradeoffs in *when* to compute distributions: Once for each attribute? Separately for each class? For only the data that makes it to each split point? They found that reconstructing each attribute separately for each class gave the best performance/accuracy tradeoff, with classification accuracy substantially better than naïvely running on the noisy data, and approaching that of building a classifier directly on the real data.

One question with this approach is how much privacy is given? With the secure multiparty computation based approaches, the definition of privacy is clear. However, given a value that is based on the real value, how do we know how much noise is enough? Agrawal and Srikant proposed a metric based the confidence in estimating a value within a specified width: If it can be estimated with $c\%$ confidence that a value x lies in the interval $[x_l, x_h]$, then the privacy at the $c\%$ confidence level is $|x_h - x_l|$. The quantify this in terms of a percentage: The privacy metric for noise from a uniform distribution is the confidence times twice the interval width of the noise: 100% privacy corresponds to a 50% confidence that the values is within two distribution widths of the real value, or nearly 100% confidence that it is within one width. They have an equivalent definition for noise from a gaussian distribution.

Agrawal and Aggarwal (not the same Agrawal) pointed out problems with this definition of privacy[1]. The very ability to reconstruct distributions may give us less privacy than expected. Figure 5 demonstrates this. Assume the noise is known to come from a uniform distribution over $[-15, 15]$, and the actual/reconstructed distribution is as shown by the bars. Since there are no drivers under age 16 (as determined from the reconstructed distribution), a driver whose age is given as 1 in the "privacy-preserving" dataset is known to be 16 years old – all privacy for this individual is lost. They instead give a definition based on entropy (discussed in Sect. 1.3). Specifically, if a random variable Y has entropy $H(Y)$, the privacy is $2^{H(Y)}$. This has the nice property that for a uniform distribution, the privacy is equivalent to the width of the interval from which the random value is chosen. This gives a meaningful way to compare different sources of noise.

They also provide a solution to the loss of privacy obtained through reconstructing the original data distribution. The idea is based on conditional entropy. Given the reconstructed distribution X, the privacy is now $2^H(Y|X)$. This naturally captures the expected privacy in terms of the interval width description: a reconstruction distribution that eliminates part of an interval (or makes it highly unlikely) gives a corresponding decrease in privacy.

There has been additional work in this area, such as techniques for association rules[14, 32]. Techniques from signal processing have also been applied to distribution reconstruction [21], generalizing much of this work. One problem is the gap between known abilities to reconstruct distributions and lower bounds on ability to reconstruct actual data values: the jury is still out on how effective these techniques really are at preserving privacy.

3 Conclusions and Recommendations

While privacy-preserving data mining does have the potential to reconcile the concerns of data mining proponents and privacy advocates, it has not reached the level of an effective panacea. Two issues remain.

First, the rigor required of the cryptography and security protocols communities must be brought to this field. While some of the work in this field does approach this level of rigor, much of the work does not. For some work, particularly with the noise addition approach, it is not clear if a determined adversary could compromise privacy (and in some cases, it is clear that they can [10, 21].) The distributed approach has a clear set of standards borrowed from the cryptography community, it is important that work be judged against these standards. In particular, work must move beyond the semi-honest model. This could mean developing efficient solutions secure against malicious adversaries, or possibly new definitions such as "proof against collusion" that meet practical needs and are defined with the rigor of the semi-honest and malicious definitions.

The second problem may be more challenging. Privacy-preserving data mining has operated under the assumption that data mining results do not of themselves compromise privacy. This is not necessarily true. While there has been some work on restricting data mining results [4, 18,19, 29, 33], this has emphasized protection against revealing specific results. This work does not address connections between the results and compromise of source data items. While work on limiting classification strength may address this issue[8], the proposed method also prevents the data from being useful for data mining in any form. Achieving a reasonable connection between individual privacy and data mining results is still an open problem. Until this is solved, the full concerns of privacy advocates will not have been addressed.

That said, privacy-preserving data mining in its current form does have practical applications. In some circumstances, the value of the data mining results may exceed the potential cost to individual privacy. This is most likely true where the individual data items reflect commercial data (e.g., intellectual property) rather than personal information. For example, U.S. antitrust law frowns on the general sharing of information between competitors, however if the shared information is limited to that absolutely necessary to achieve some consumer benefit the sharing is likely to pass legal muster. The concept that use of information is allowed when necessary also shows up in the European Community privacy recommendations[13], reconciling these laws with the potential privacy breach of data mining results will be easier if privacy-preserving data mining techniques are used to ensure that *only* the results are disclosed.

In summary, while privacy-preserving data mining has achieved significant success, many challenges remain.

References

1. D. Agrawal and C. 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*. Santa Barbara, California, USA: ACM, May 21–23 2001, pp. 247–255. [Online]. Available: http://doi.acm.org/10.1145/375551.375602 310, 336
2. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*. Santiago, Chile: VLDB, Sept. 12–15 1994, pp. 487–499. [Online]. Available: http://www.vldb.org/dblp/db/conf/vldb/vldb94-487.html 314

3. ——, “Privacy-preserving data mining,” in *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*. Dallas, TX: ACM, May 14–19 2000, pp. 439–450. [Online]. Available: http://doi.acm.org/10.1145/342009. 335438 310, 333, 334, 335
4. M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios, “Disclosure limitation of sensitive rules,” in *Knowledge and Data Engineering Exchange Workshop (KDEX’99)*, Chicago, Illinois, Nov. 8 1999, pp. 25–32. [Online]. Available: http://ieeexplore.ieee.org/iel5/6764/18077/00836532.pdf?isNumber=18077&prod=CNF&arnumber=00836532 337
5. D. Beaver, “Commodity-based cryptography (extended abstract),” in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. El Paso, Texas, United States: ACM Press, 1997, pp. 446–455. 325
6. J. Benaloh, “Dense probabilistic encryption,” in *Proceedings of the Workshop on Selected Areas of Cryptography*, Kingston, Ontario, May 1994, pp. 120–128. [Online]. Available: http://research.microsoft.com/crypto/papers/dpe.ps 328
7. D. W.-L. Cheung, V. Ng, A. W.-C. Fu, and Y. Fu, “Efficient mining of association rules in distributed databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 911–922, Dec. 1996. 316
8. C. Clifton, “Using sample size to limit exposure to data mining,” *Journal of Computer Security*, vol. 8, no. 4, pp. 281–307, Nov. 2000. [Online]. Available: http://iospress.metapress.com/openurl.asp?genre=article&issn=0926-227X&volume=8&issue=4&spage=281 337
9. “Total information awareness (TIA) system,” Defense Advanced Research Projects Agency. [Online]. Available: http://www.darpa.mil/iao/TIASystems.htm 309
10. I. Dinur and K. Nissim, “Revealing information while preserving privacy,” in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM Press, 2003, pp. 202–210. [Online]. Available: http://doi.acm.org/10.1145/773153.773173 310, 336
11. W. Du and M. J. Atallah, “Privacy-preserving statistical analysis,” in *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, December 10–14 2001. [Online]. Available: http://www.cerias.purdue.edu/homes/duw/research/paper/acsac2001.ps 328
12. W. Du and Z. Zhan, “Building decision tree classifier on private data,” in *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, C. Clifton and V. Estivill-Castro, Eds., vol. 14. Maebashi City, Japan: Australian Computer Society, Dec. 9 2002, pp. 1–8. [Online]. Available: http://crpit.com/Vol14.html 325
13. “Directive 95/46/EC of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data,” *Official Journal of the European Communities*, vol. No I., no. 281, pp. 31–50, Oct. 24 1995. [Online]. Available: http://europa.eu.int/comm/internal_market/privacy/ 337
14. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, “Privacy preserving mining of association rules,” in *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 23–26 2002, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/775047.775080 336
15. M. Feingold, M. Corzine, M. Wyden, and M. Nelson, “Data-mining moratorium act of 2003,” U.S. Senate Bill (proposed), Jan. 16 2003. [Online]. Available: http://thomas.loc.gov/cgi-bin/query/z?c108:S.188: 309

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*, 1987, pp. 218–229. [Online]. Available: http://doi.acm.org/10.1145/28395.28420 310, 311
17. O. Goldreich, "Secure multi-party computation," Sept. 1998, (working draft). [Online]. Available: http://www.wisdom.weizmann.ac.il/ oded/pp.html 313, 318
18. T. Johnsten and V. Raghavan, "Impact of decision-region based classification algorithms on database security," in *Proceedings of the Thirteenth Annual IFIP WG 11.3 Working Conference on Database Security*, Seattle, Washington, July 26–28 1999. [Online]. Available: http://www.cacs.usl.edu/Publications/Raghavan/JR99.ps.Z 337
19. ——, "A methodology for hiding knowledge in databases," in *Proceedings of the IEEE ICDM Workshop on Privacy, Security and Data Mining*, ser. Conferences in Research and Practice in Information Technology, vol. 14. Maebashi City, Japan: Australian Computer Society, Dec. 9 2002, pp. 9–17. [Online]. Available: http://crpit.com/confpapers/CRPITV14Johnsten.pdf 337
20. M. Kantarcıoğlu and C. Clifton, "Privacy-preserving distributed mining of association rules on horizontally partitioned data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 4, July 2004. 320
21. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the privacy preserving properties of random data perturbation techniques," in *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, Nov. 19–22 2003. 310, 336
22. M. Lewis, "Department of defense appropriations act, 2004," July 17 2003, title VIII section 8120. Enacted as Public Law 108–87. [Online]. Available: http://thomas.loc.gov/cgi-bin/bdquery/z?d108:h.r.02658: 309
23. X. Lin, C. Clifton, and M. Zhu, "Privacy preserving clustering with distributed EM mixture modeling," *Knowledge and Information Systems*, to appear 2004. 326
24. Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Advances in Cryptology – CRYPTO 2000*. Springer-Verlag, Aug. 20–24 2000, pp. 36–54. [Online]. Available: http://link.springer.de/link/service/series/0558/bibs/1880/18800036.htm 310, 321, 323
25. M. Murkowski and M. Wyden, "Protecting the rights of individuals act," U.S. Senate Bill (proposed), July 31 2003. [Online]. Available: http://thomas.loc.gov/cgi-bin/bdquery/z?d108:s.01552: 309
26. D. Naccache and J. Stern, "A new public key cryptosystem based on higher residues," in *Proceedings of the 5th ACM conference on Computer and communications security*. San Francisco, California, United States: ACM Press, 1998, pp. 59–66. 328
27. M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. Atlanta, Georgia, United States: ACM Press, 1999, pp. 245–254. 324
28. T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *Advances in Cryptology – Eurocrypt '98, LNCS 1403*. Springer-Verlag, 1998, pp. 308–318. 328
29. S. R. M. Oliveira and O. R. Zaïane, "Protecting sensitive knowledge by data sanitization," in *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, Nov. 19–22 2003. 337

30. P. Paillier, “Public key cryptosystems based on composite degree residuosity classes,” in *Advances in Cryptology – Eurocrypt ’99 Proceedings, LNCS 1592*. Springer-Verlag, 1999, pp. 223–238. 328
31. J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: http://ipsapp008.kluweronline.com/content/getfile/4984/119/4/abstract.htm 321
32. S. J. Rizvi and J. R. Haritsa, “Maintaining data privacy in association rule mining,” in *Proceedings of 28th International Conference on Very Large Data Bases*. Hong Kong: VLDB, Aug. 20–23 2002, pp. 682–693. [Online]. Available: http://www.vldb.org/conf/2002/S19P03.pdf 336
33. Y. Saygin, V. S. Verykios, and C. Clifton, “Using unknowns to prevent discovery of association rules,” *SIGMOD Record*, vol. 30, no. 4, pp. 45–54, Dec. 2001. [Online]. Available: http://www.acm.org/sigmod/record/issues/0112/SPECIAL/5.pdf 337
34. J. Vaidya and C. Clifton, “Privacy preserving association rule mining in vertically partitioned data,” in *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 23–26 2002, pp. 639–644. [Online]. Available: http://doi.acm.org/10.1145/775047.775142 321
35. ——, “Leveraging the “multi” in secure multi-party computation,” in *Workshop on Privacy in the Electronic Society held in association with the 10th ACM Conference on Computer and Communications Security*, Washington, DC, Oct. 30 2003. 332
36. ——, “Privacy-preserving k-means clustering over vertically partitioned data,” in *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, Aug. 24–27 2003. 328, 329
37. ——, “Secure set intersection cardinality with application to association rule mining,” *Journal of Computer Security*, submitted. 321, 331, 332, 333
38. A. C. Yao, “How to generate and exchange secrets,” in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE, 1986, pp. 162–167. 310, 311