**College Result Analysis with Horizontally**

 **Distributed Databases**

**ABSTRACT**

 We propose a protocol for secure mining of association rules in horizontally distributed databases. The current leading protocol is that of Kantarcioglu and Clifton. Our protocol, like theirs, is based on the Fast Distributed Mining (FDM) algorithm of Cheung et al. [8], which is an unsecured distributed version of the Apriori algorithm. The main ingredients in our protocol are two novel secure multi-party algorithms one that computes the union of private subsets that each of the interacting players hold, and another that tests the inclusion of an element held by one player in a subset held by another. Our protocol offers enhanced privacy with respect to the protocol in . In addition, it is simpler and is significantly more efficient in terms of communication rounds, communication cost and computational cost.

**Existing System**

That goal defines a problem of secure multi-party computation. In such problems, there are *M* players that hold private inputs, *x*1*, . . . , xM*, and they wish to securely compute *y* = *f*(*x*1*, . . . , xM*) for some public function *f*. If there existed a trusted third party, the players could surrender to him their inputs and he would perform the function evaluation and send to them the resulting output. In the absence of such a trusted third party, it is needed to devise a protocol that the players can run on their own in order to arrive at the required output *y*. Such a protocol is considered perfectly secure if no player can learn from his view of the protocol more than what he would have learnt in the idealized setting where the computation is carried out by a trusted third party. Yao [32] was the first to propose a generic solution for this problem in the case of two players. Other generic solutions, for the multi-party case, were later proposed in

**Proposed System**

Herein we propose an alternative protocol for the secure computation of the union of private subsets. The proposed protocol improves upon that in [18] in terms of simplicity and efficiency as well as privacy. In particular, our protocol does not depend on commutative encryption and oblivious transfer (what simplifies it significantly and contributes towards much reduced communication and computational costs). While our solution is still not perfectly secure, it leaks excess information only to a small number (three) of possible coalitions, unlike the protocol of that discloses information also to some single players. In addition, we claim that the excess information that our protocol may leak is less sensitive than the excess information leaked by the protocol of.

The protocol that we propose here computes a parameterized family of functions, which we call threshold functions, in which the two extreme cases correspond to the problems of computing the union and intersection of private subsets. Those are in fact general-purpose protocols that can be used in other contexts as well. Another problem of secure multiparty computation that we solve here as part of our discussion is the set inclusion problem; namely, the problem where Alice holds a private subset of some ground set, and Bob holds an element in the ground set, and they wish to determine whether Bob’s element is within Alice’s subset, without revealing to either of them information about the other party’s input beyond the above described inclusion.

### IMPLEMENTATION

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

 The implementation stage involves careful planning, investigation of the existing system and it’s constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

**Modules:**

**1. Privacy Preserving Data Mining:**

 Previous work in privacy preserving data mining has considered two related settings. One, in which the data owner and the data miner are two different entities, and another, in which the data is distributed among several parties who aim to jointly perform data mining on the unified corpus of data that they hold.

In the first setting, the goal is to protect the data records from the data miner. Hence, the data owner aims at anonym zing the data prior to its release. The main approach in this context is to apply data perturbation. The idea is that. Computation and communication costs versus the number of transactions *N* the perturbed data can be used to infer general trends in the data, without revealing original record information.

In the second setting, the goal is to perform data mining while protecting the data records of each of the data owners from the other data owners. This is a problem of secure multiparty computation. The usual approach here is cryptographic rather than probabilistic.

Lind ell and Pink as showed how to securely build an ID3 decision tree when the training set is distributed horizontally. Lin et al. discussed secure clustering using the EM algorithm over horizontally distributed data. The problem of distributed association rule mining was studied in the vertical setting, where each party holds a different set of attributes, and the horizontal setting. Also the work of considered this problem in the horizontal setting, but they considered large-scale systems in which, on top of the parties that hold the data records (resources) there are also managers which

**2. Distributed Computation:**

 We compared the performance of two secure implementations of the FDM algorithm Section In the first implementation (denoted FDM-KC), we executed the unification step using Protocol UNIFI-KC, where the commutative cipher was 1024-bit RSA in the second implementation (denoted FDM) we used our Protocol UNIFI, where the keyed-hash function was HMAC . In both implementations, we implemented Step 5 of the FDM algorithm in the secure manner that was described in later. We tested the two implementations with respect to three measures:

1) Total computation time of the complete protocols (FDMKC and FDM) over all players. That measure includes the Apriori computation time, and the time to identify the globally *s*-frequent item sets, as described in later.

2) Total computation time of the unification protocols only (UNIFI-KC and UNIFI) over all players.

3) Total message size. We ran three experiment sets, where each set tested the dependence of the different parameter: *• N* — the number of transactions in the unified database,

**Algorithm:**

 Secure computation of the *t*threshold function

Input: Each player *Pm* has an input binary vector **b***m ∈* Z*n*2 ,1 *≤ m ≤ M*.

Output: **b** := *Tt*(**b**1*, . . . ,* **b***M*).

1: Each *Pm* selects *M* random share vectors **b***m,ℓ ∈* Z*n M*+1,1 *≤ ℓ ≤ M*, such that Σ*M ℓ*=1 **b***m,ℓ* = **b***m* mod (*M* + 1).

2: Each *Pm* sends **b***m,ℓ* to *Pℓ* for all 1 *≤ ℓ ̸*= *m ≤ M*.

3: Each *P*Σ *ℓ* computes **s***ℓ* = (*sℓ*(1)*, . . . , sℓ*(*n*)) := *M m*=1 **b***m,ℓ* mod (*M* + 1).

4: Players *Pℓ*, 2 *≤ ℓ ≤ M −* 1, send **s***ℓ* to *P*1.

5: Σ*P*1 computes **s** = (*s*(1)*, . . . , s*(*n*)) := *M−*1 *ℓ*=1 **s***ℓ* mod (*M* + 1).

6: for *i* = 1*, . . . , n* do

7: If (*s*(*i*) + *sM*(*i*)) mod (*M* + 1) *< t* set *b*(*i*) = 0 otherwise set *b*(*i*) = 1.

8: end for

9: Output **b** = (*b*(1)*, . . . , b*(*n*)).

**3. Frequent Itemsets:**

 We describe here the solution that was proposed by Kantarcioglu and Clifton. They onsidered two possible settings. If the required output includes all globally *s*-frequent item sets, as well as the sizes of their supports, then the values of Δ(*x*) can be revealed for all *x ∈ Ck s* . In such a case, those values may be computed using a secure summation protocol , where the private addend of *Pm* is *suppm*(*x*) *− sNm*. The more interesting setting, however, is the one where the support sizes are not part of the required output. We proceed to discuss it.

**Algorithm:**

Unifying lists of locally Frequent Itemsets

Input: Each player *Pm* has an input subset *Ck,ms⊆*

*Ap*(*Fk−*1*s* ), 1 *≤ m ≤ M*.

Output: *Cks* =∪*Mm*=1 *Ck,ms* .

1: Each player *Pm* encodes his subset *Ck,ms* as a binary vector **b***m* of length *nk* = *|Ap*(*Fk−*1

*s* )*|*, in accord with the agreed ordering of *Ap*(*Fk−*1 *s* ).

2: The players invoke Protocol THRESHOLD-C to compute **b** = *T*1(**b**1*, . . . ,* **b***M*) =∨*M*

*m*=1 **b***m*.

3: *Cks* is the subset of *Ap*(*Fk−*1*s* ) that is described by **b**.

**4. Association Rules:**

 Once the set *Fs* of all *s*-frequent itemsets is found, we may proceed to look for all (*s, c*)-association rules (rules with support at least *sN* and confidence at least *c*), as described in . For *X, Y ∈ Fs*, where *X ∩ Y* = *∅*, the corresponding association rule *X ⇒ Y* has confidence at least *c* if and only if *supp*(*X ∪ Y* )*/supp*(*X*) *≥ c*, or, equivalently, *CX,Y* := *M*Σ *m*=1 (*suppm*(*X ∪ Y* ) *− c · suppm*(*X*)) *≥* 0 *.* (10) Since *|CX,Y | ≤ N*, then by taking *q* = 2*N*+1, the players can verify inequality (10), in parallel, for all candidate association rules, as described in Section 3. In order to derive from *Fs* all (*s, c*)-association rules in an efficient manner we rely upon the following straightforward lemma.

**Algorithm:**

Set Inclusion computation

Input: *P*1 has a vector **s** = (*s*(1)*, . . . , s*(*n*)) and *PM* has a vector **\_** = (Θ(1)*, ,* Θ(*n*)), where for all 1 *≤ i ≤ n*, *s*(*i*) *∈* Ω and Θ(*i*) *⊆* Ω for some ground set Ω.

Output: The vector **b** = (*b*(1)*, . . . , b*(*n*)) where *b*(*i*) = 0 if *s*(*i*) *∈* Θ(*i*) and *b*(*i*) = 1 otherwise, 1 *≤ i ≤ n*.

1: *P*1 and *PM* agree on a keyed-hash function *hK*(*·*) and on a secret key *K*.

2: *P*1 computes **s***′* = (*s′*(1)*, . . . , s′*(*n*)), where *s′*(*i*) = *hK*(*i, s*(*i*)), 1 *≤ i ≤ n*.

3: *PM* computes **\_***′* = (Θ*′*(1)*, . . . ,*Θ*′*(*n*)), where Θ*′*(*i*) =*{hK*(*i, θ*) : *θ ∈* Θ(*i*)*}*, 1 *≤ i ≤ n*.

4: *P*1 sends to *P*2 the vector **s***′*.

5: *PM* sends to *P*2 the vector **\_***′* in which each Θ(*i*) israndomly permuted.

6: For all 1 *≤ i ≤ n*, *P*2 sets *b*(*i*) = 0 if *s′*(*i*) *∈* Θ*′*(*i*) and *b*(*i*) = 1 otherwise.

7: *P*2 broadcasts the vector **b** = (*b*(1)*, . . . , b*(*n*)).

***System Architecture:***

******