Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 41 – 50

ISSN 1896-7094

Implementation and Tests of Clustering over Vertically Distributed Data without Sharing their Values

Marcin Gorawski[1] and Łukasz Słabiński[1]

[1] Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice, Poland
{M.Gorawski, L.Slabinski}@polsl.pl

Abstract. This paper describes an experimental implementation of a secure distributed *k*-Clustering algorithm which applies the homomorphic encryption scheme for preserving privacy of data. We show in details technical aspects of the homomorphic encryption implementation and using it in a distributed application. The implementation enables us to evaluate experimentally the efficiency of the clustering algorithm, indicate its limitations and potential bottlenecks. Finally, we provide some conclusions and propose further work in this area.

1 Introduction

The goal of this work is to evaluate the efficiency of privacy-preserving *k*-means clustering algorithm over vertically partitioned data previously presented by Jaideep Vaidy and Chris Clifton in [1]. We used their algorithm as a framework of our version of secure distributed *k*-means clustering. We provided a few modifications in implementation that increase its efficiency.

To show the idea of secure distributed data mining, in our case a distributed clustering, let us consider the following scenario:

The items are described by two attributes *X* and *Y*. Figure 1 shows that they concentrate around three points: (100; 100), (100; 400) and (450; 300). Let us assume that the attributes are split between two different companies. The company that knows only the attribute *X* is unable to see differences between two clusters on left side. It finds only two clusters with means in the points 100, 450. The second company that knows only the attribute *Y*, can not see any clusters or assign clusters with insufficient precision. The simplest solution in this situation is to join values of attributes *X* and *Y*, and to allow both companies to mine such a centralized dataset. However, if companies do not trust each other and cannot allow revealing private information, this solution is not acceptable. The only way is to use appropriate privacy-preserving clustering algorithm.

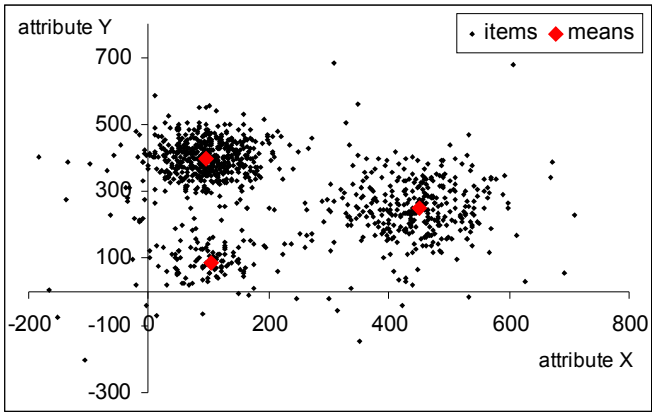

Fig. 1. The dataset distribution in two-attributes space.

The main idea of privacy-preserving algorithms is applying special techniques which hide real values of the analyzed data [2]. Most of them randomize the shared data, so that the data can be used in distributed computation, but do not reveal any true values [3]. Such an approach was presented in [4, 5, 6] in the case of clustering algorithms and in [7] in the case of secure mining of association rules. Unfortunately, randomization does not guarantee full privacy. When a malicious party receives randomized values it can try to approximate their distribution or variety. To preserve entire information about the data: row values, as well metadata, we have to use more sophisticated secure multi-party computation (SMC) techniques [8, 9, 10]. The SMC approach considers the problem of evaluating a function of at least two secret distributed inputs, so that each party obtains the result and nothing else, except what is implied by the party's own data and the result. Usually in order to obtain this property we use cryptographic algorithms. One of such cryptographic tools is the Commutative Encryption [11]. It ensures two main properties. First, we can encrypt the data a few times using different public-keys and decrypt them in any other order. Second, if two items are equal, with very high probability their cryptograms will also be equal. The commutative encryption is a suitable tool for analyzing data containing transactions, e.g. mining of association rules [12, 13]. Next very efficient encryption scheme is the Homomorphic Encryption [14, 18] that ensure that for any two encrypted messages E(A) and E(B) exits E(A +B), such that $E(A)*E(B)=E(A+B)$. The secure clustering algorithm that is presented in this paper is based on this encryption scheme.

In Section 2 we analyze the secure distributed *k*-means clustering and we present briefly its implementation. In Section 3 we describe more closely technical aspects of implementation the homomorphic encryption scheme. The application tests are showed in Section 4. Section 5 contains summary and further work.

2 Secure Distributed Clustering

2.1 Algorithm

The algorithm assumes that at least three non-colluded parties participate in the clustering. At the end of computation, each party knows global means of all clusters and assignment of their local items to these clusters. Simultaneously, each party can be sure that during computations none of their private data was revealed. The algorithm can be presented briefly as follows:

1. Each party arbitrarily chooses initial means of all *k* clusters: $\mu_1 \cdots \mu_k$.
2. Each party, for each item, generates vector *X* which stores the distances between the item and each cluster in *n*-dimensionally space.

3. One party (donated as P_1) generates *r* *k*-dimensionally vectors V_i and a random permutation π over *k* numbers.

$$\vec{X_i} = \begin{bmatrix} x_{1i} \\ x_{2i} \\ . \\ . \\ . \\ x_{ki} \end{bmatrix} \qquad \underset{z=1\ldots k}{x_{zi}} = \sqrt{\sum_{j=1}^{n} (x_{zj} - y_{zj})^2} \qquad \vec{V_i} = \begin{bmatrix} v_{1i} \\ v_{2i} \\ . \\ . \\ . \\ v_{ki} \end{bmatrix} \qquad \sum_{z=1\ldots k}^{r} v_{zj} = 0$$

4. Each party generates public-private key pair *(E_i, D_i)* and engages in communication with the party P_1 to calculate vector $\pi(X_i + V_i)$. During the communication parties use the homomorphic encryption schema which enables parties to compute vector $\pi(X_i + V_i)$ without sharing values of vectors X_i and V_i or π.

$$\vec{E_i(X_i)} = \begin{bmatrix} E_i(x_{1i}) \\ E_i(x_{2i}) \\ . \\ . \\ . \\ E_i(x_{ki}) \end{bmatrix} \qquad \vec{E_i(V_i)} = \begin{bmatrix} E_i(v_{1i}) \\ E_i(v_{2i}) \\ . \\ . \\ . \\ E_i(v_{ki}) \end{bmatrix} \qquad \vec{E_i(X_i)} * \vec{E_i(V_i)} = \begin{bmatrix} E_i(x_{1i} + v_{1i}) \\ E_i(x_{2i} + v_{2i}) \\ . \\ . \\ . \\ E_i(x_{ki} + v_{ki}) \end{bmatrix} = \vec{E(X_i + V_i)}$$

$$\vec{\pi(E_i(X_i + V_i))} = \begin{bmatrix} E_i(x_{\pi(1)i} + v_{\pi(1)i}) \\ E_i(x_{\pi(2)i} + v_{\pi(2)i}) \\ . \\ . \\ . \\ E_i(x_{\pi(k)i} + v_{\pi(k)i}) \end{bmatrix} \qquad \vec{D_i(\pi(E_i(X_i + V_i)))} = \begin{bmatrix} x_{\pi(1)i} + v_{\pi(1)i} \\ x_{\pi(2)i} + v_{\pi(2)i} \\ . \\ . \\ . \\ x_{\pi(k)i} + v_{\pi(k)i} \end{bmatrix} = \vec{\pi(X_i + V_i)}$$

5. Next, all parties, excluding one (denoted as P_2) send their vectors $\pi(X_i + V_i)$ to the party P_r.
6. Now, parties P_2 and P_r engage in communication in order to find the number *j* of row in vectors $\pi(X_i + V_i)$ which gives the minimal sum of elements (number of the closest cluster)

$$\underset{j=1\ldots k}{\arg\min}\left(\sum_{i=1\ldots r} x_{ji}\right)$$

7. To know the real number of the minimal row in vectors, the party P_2 sends the number *j* to the party P_1 which knows the permutation π.
8. The party P_1, after calculating the real number of the minimal row in vectors, broadcasts it to all parties.
9. Each party assigns the considered item to the globally closest cluster.
10. After assigning all items to the closest clusters, each party calculates new mean for each cluster: $\mu'_1 \ldots \mu'_k$.
11. All parties together, using secure algorithm, check if the sum of their total distance between new and old means is small enough to finish calculations.

2.2 Implementation

The framework of the algorithm was implemented in Java. Communication between parties uses the TCP/IP protocol implemented by the Java Sockets API.

The core of the algorithm implementation are three procedures which securely calculate the numbers of the closest clusters for points. At least three parties (denoted as P_1, P_2, P_r) have to participate in computation. Each consecutive party executes only basic operations (present in all procedures for P_1, P_2 and P_r).

```
/* The procedure for the party P1 */
private ArrayList closestCluster_P1(){
  ArrayList V,Pi,EX,PiEXV,PiXV,K;         /* variables */
  V = genVectorsV(parties_number);
  Pi = genPermutation(cluster_number);
  EX = encryptAllX();
  PiEXV = randOwnX(EX,V,Pi);              /* multiply and permute */
  PiXV = decrypt(PiEXV);
  receiveAndRandAllX(V,Pi);               /* randomize other X */
  sendXToPr(PiXV);
  K = receiveAndPermuteBack(Pi);
  broadcast(K);
  return K;
}
/* The procedure for the party P2 */
private ArrayList closestCluster_P2()throws IOException{
  ArrayList EX,PiEXV,K,PiXV,PiK;          /* variables */
  EX = encryptAllX();
  PiEXV = randomizeWithP1(EX);            /* communication with P1 */
  PiXV = decrypt(PiEXV);
  PiK = findMinRowWithPr(PiXV);           /* communication with Pr */
  sendKToP1(PiK);
  K = receiveKFromP1();                   /* wait for results */
  return K;
}

/* The procedure for the party Pr */
private ArrayList closestCluster_Pr()throws IOException{
  ArrayList EX,PiEXV,K,PiXV,PiK,AllXV;
  EX = encryptAllX();                     /* variables */
  PiEXV = randomizeWithP1(EX);            /* communication with P1*/
  PiXV  = decrypt(PiEXV);
  AllXV = receiveAllPiXV(PiXV);           /* wait for other PiXV */
  findRowWithP2(AllXV);                   /* communication with P2 */
  K = receiveKFromP1();                   /* wait for results */
  return K;
}
```

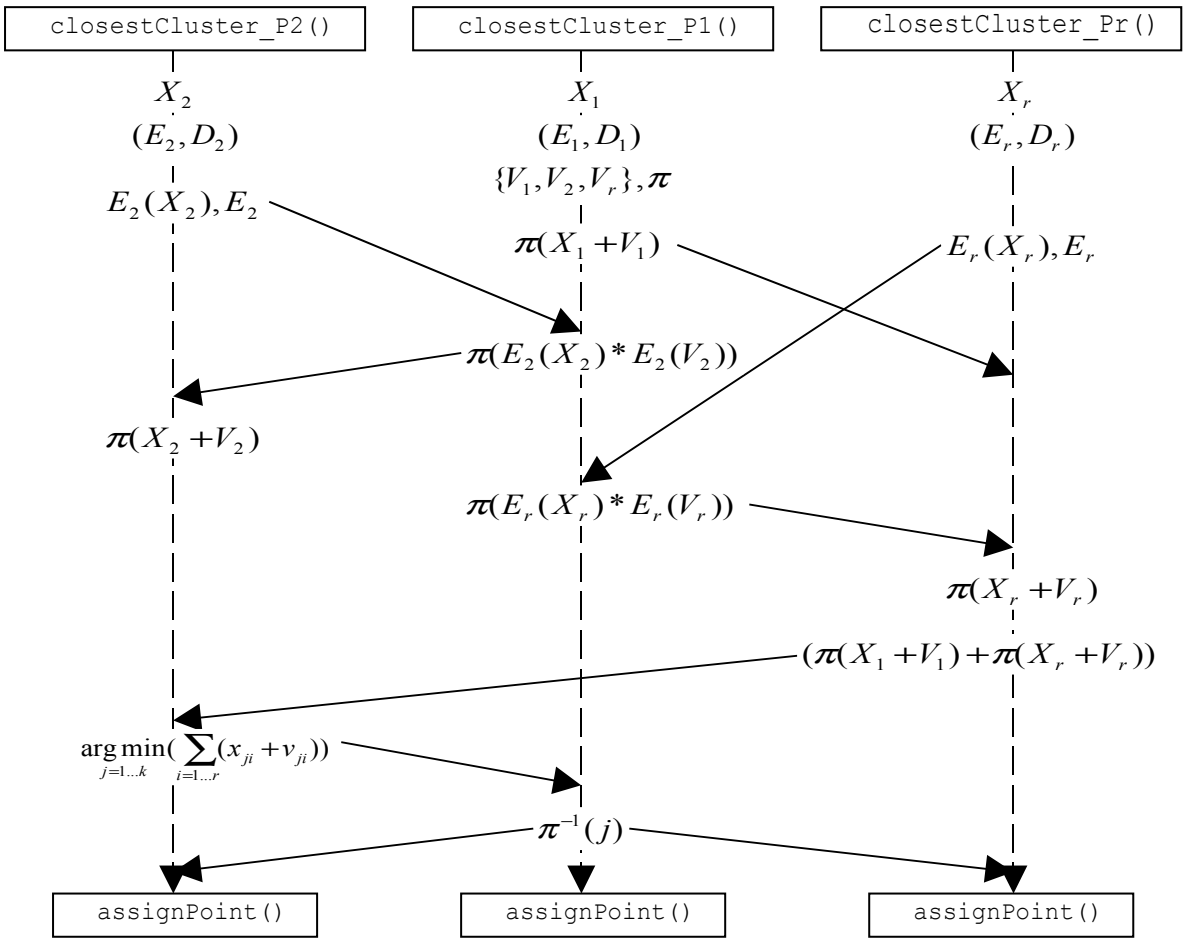

Fig. 2. Example of interactions between three parties during finding the closest cluster for the point.

Example of interactions between the above procedures is shown in figure 2. To simplify the graph we show interactions for a single point. To increase performance, our application conducts computations for all points in the same time.

```
/* Party P1 starts */
Long Sum = moveMeans();       /* local dislocation of cluster means */
String msg = Long.toString(Sum.longValue(),16);
String publicKey = crypto.key();          /* generate the key pair */
String Emsg = crypto.encrypt(msg);        /* encrypt message */
Packet pack = new Packet(Emsg,publicKey);
send(ip,port,pack);                       /* send packet to next party */
pack =(Packet)receive();                  /* wait for result */
```

```
/* Each other party in arbitrary chosen order computes */
Packet pack = (Packet)receive();          /* wait for packet */
Long locSum = moveMeans();    /* local dislocation of cluster means*/
String msq = Long.toString(locSum.longValue(),16);
/* multiply using homomorphic encryption */
String E_sum = addEncrypt(msg,pack.value(),pack.key());
pack.setValue(E_sum);
send(ip,port,pack);                       /* send to next party */
Boolean stop= (Boolean)receive();         /* wait for the result */
return stop;
```

```
/* when 'pack' comes back to the party P1 */
String msg = crypto.decrypt(pack.value());
Long R = new Long(Long.parseLong(msg),16);
Boolean stop = new Boolean(R.longValue()< threshold );
broadcast(stop);                          /* broadcast the result */
return stop.booleanValue();
```

After finding the closest cluster for all points and recalculating the clusters, parties check if the total (global) dislocation of cluster means is small enough to finish calculations. It is implemented by two procedures: one for party denoted as P_1 and one for all other parties. Example of interactions between parties during this stage is shown in figure 3.

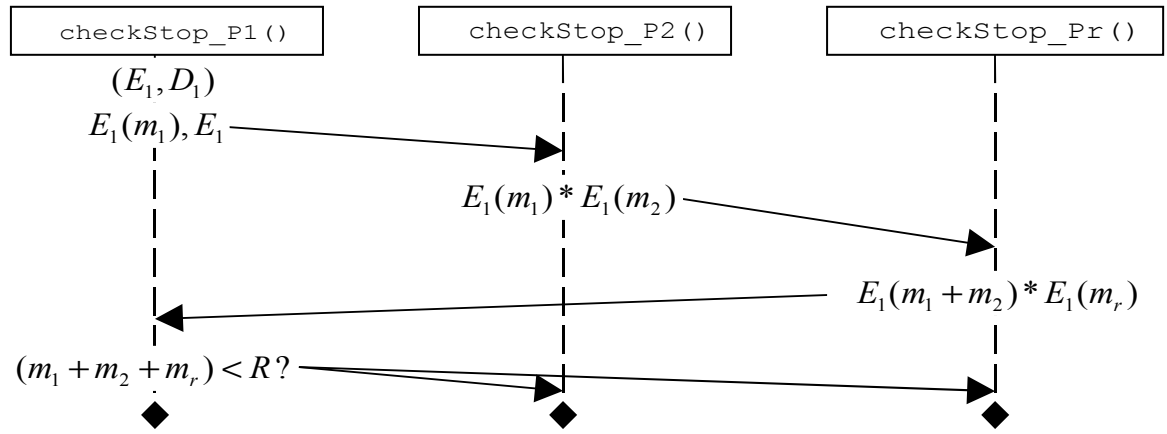

Fig. 3. Example of interactions between three parties during checking the stop criteria.

3 Cryptographic Elements

To preserve privacy of the clustered data, the algorithm uses the homomorphic encryption scheme. It is an asymmetric public-key cryptosystem which ensures that for any two encrypted messages E(A) and E(B) exists E(A + B), such that $E(A) * E(B) = E(A + B)$.

In our application, we implemented the Paillier's homomorphic cryptosystem [14]. In order to maximize its performance, all cryptographic elements were written in C++ and compiled into an external dynamic library. Main program uses them by means of the JINI protocol. In source code we used some techniques from the MIRACLE cryptographic library [15].

3.1 Key Pair Generation

In homomorphic encryption scheme key pair generation is very similar to key pair generation in the classic RSA algorithm. It bases on selecting two large primes *p, q* and *n=p*q:*

```
Big n,p,q;                      /* class Big represents a big integer */
int size = 128;                 /* size of big integer variable in bits */
long seed[4];

for (int i=0;i<4;i++)           /* generate random seeds */
  seed[i]=rand()*10000+rand();
```

```
/* function strongp() returns a random big integer */
p = strongp(size,seed[0],seed[1]);
q = strongp(size,seed[2],seed[3]);
n=p*q;
```

Finally:

- Public-key is the number *n*.
- Private-key is pair of numbers *p* and *q*.

3.2 Homomorphic Encryption

To encrypt a message m < n we have to select a random number *r* and calculate the equation: $c = g^m r^n \bmod n^2$.

```
Big m;                          /* message */
Big n;                          /* public key */
Big n2 = n*n;                   /* calculate modulo n^2 */
Big r = rand(n2);               /* select random r */
r = pow(r,n,n2);                /* calculate r^n mod n^2 */
Big g = 1+m*n;                  /* calculate g */
Big c = modmult(g,r,n2);        /* c = g^m r^n mod n^2 */
return c;                       /* return cryptogram */
```

3.3 Multiplication of Two Encrypted Messages

To enable the homomorphic property: $\mathrm{E(m1)} * \mathrm{E(m2)} = \mathrm{E(m1+m2)}$ we simply multiply two encrypted messages ensuring that all operations are modulo n^2

```
Big c1;                         /* encrypted message 1 */
Big c2;                         /* encrypted message 2 */
Bin n2 = n*n;                   /* public key */
Big c2 = modmult(c1,c2,n2);     /* E(m1)*E(m2)=E(m1+m2)  */
return c2;                      /* return added cryptograms */
```

3.4 Decryption

To decrypt the message *m* from the cryptogram *c*, we have to solve the equation: $m = \dfrac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$

```
Big p,q;                        /* private key */
Big n=p*q;
Big n2=n*n;
Big p2=p*p;
Big q2=q*q;
/* Carmichael function: λ(n) = lcm(p - 1,q - 1) */
Big lcm=((p-1)*(q-1))/gcd(p-1,q-1);
Big er[2],em[2];
em[0]=lcm;
em[1]=n;
er[0]=0;
er[1]=1;
Big d=CRT(em,er);               /* Chinese remainder theorem */
Big c;                          /* cryptogram */
em[0]=p2;
```

```
em[1]=q2;
er[0]=pow(c,d%(p2-p),p2);
er[1]=pow(c,d%(q2-q),q2);
m=CRT(em,er);                    /* Chinese remainder theorem */
m=(m-1)/n;
return m;                        /* return decrypted messages */
```

4 Application Tests

The tests proved that the algorithm is efficient. Having set the proper initial parameters (selecting optimal initial parameters is a separate problem, discussed e.g. in [16, 17]), each party, without revealing any information about their private data, obtained the global clusters which were visible only in the joined data. The clusters obtained during secure multi-part computations were identical to the ones obtained by *k*-means algorithm applied by a single party on the centralized data.

The computation time depends linearly on the size of clustering datasets (Fig. 4)

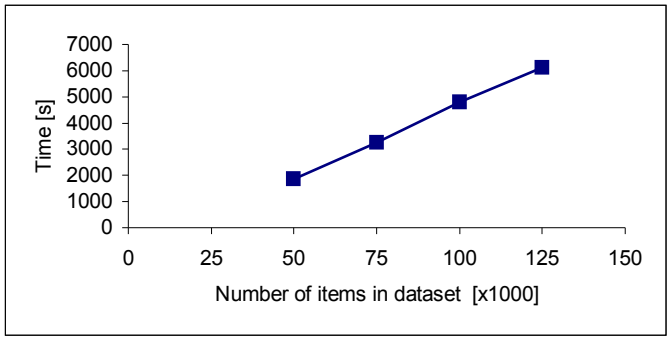

Fig. 4. Efficiency of the distributed data clustering over three parties. Parties divided items into 8 clusters and used 128 bits public-private key pairs.

The most time-consuming operations are encryption and decryption. Approximately more then 95% of the total time of algorithm execution takes cryptography. Additionally, the time of algorithm computations depends almost exponentially on the size of a public-private key pair (Fig. 5). This is the well-known problem for any SMC-based algorithm which always has to make the trade-off between performance and privacy level.

More the test results are provided in [18].

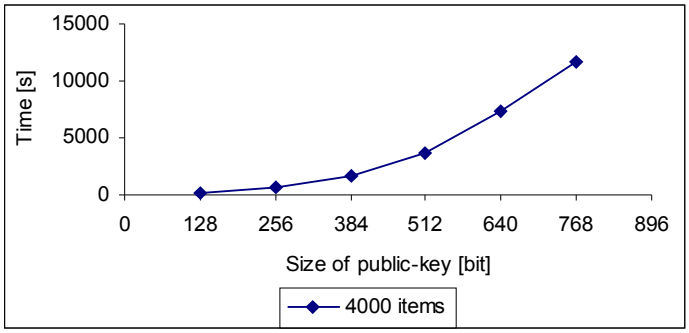

Fig. 5. Efficiency of the distributed data clustering over three parties. Chart shows time for one algorithm iteration. Tested were various sizes of the public-private key pairs for the dataset of 4000 items.

5 Conclusions

This paper presented the results of the experimental implementation of the secure distributed clustering algorithm. In particular, we described implementation of the homomorphic encryption scheme which was applied to preserve privacy clustered data. Tests proved that the algorithm is efficient. A few independent parties that know different attributes of items are able to obtain the global result without sharing any private data. Unfortunately, the homomorphic encryption scheme adds large additional computational cost relative to the previous non-secure algorithm of *k*-means clustering. This cost rapidly grows with the level of encryption security.

In upcoming research we propose to:

- limit and optimize the cryptographic elements to improve overall performance of the application,
- implement the collusions avoiding mechanism,
- develop a secure distributed algorithm for refining initial means of clusters.

References

1. Jaideep Vaidya and Chris Clifton: *Privacy-preserving k-means clustering over vertically partitioned data,* In Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 206-215. ACM Press, August 2003.
2. Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu: *Tools for Privacy Preserving Distributed Data Mining,* In SIGKDD Explorations, **4**(2): 28-34 December 2002.
3. Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant: *Limiting privacy breaches in privacy preserving data mining,* In Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003), June 2003.
4. S. Merugu and J. Ghosh: *Privacy-Preserving Distributed Clustering Using Generative Models,* In Proceedings of the Third IEEE International Conference on Data Mining ICDM'03, November 2003.
5. M. Klusch, S. Lodi, and G. Moro: *Distributed Clustering Based on Sampling Local Density Estimates,* In Proc. Int. Joint Conference on Artificial Intelligence (IJCAI), 2003.
6. Stanley R. M. Oliveira and Osmar R. Zaane: *Privacy Preserving Clustering By Data Transformation,* In Proceedings of the Eighteenth Brazilian Symposium on Databases, pages 304-318, October 2003.
7. Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and J. E. Gehrke: *Privacy Preserving Mining of Association Rules,* In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.
8. A. C. Yao: *Protocols for secure computations,* In Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982.
9. C. Yao: *How to generate and exchange secrets,* in Proceedings 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 162–167.
10. O. Goldreich: *Secure multi-party computation* Sept. 1998, (working draft).
11. S. C. Pohlig and M. E. Hellman: *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance,* In *IEEE Transactions on Information Theory*, **IT-24**:106-110, 1978.
12. Jaideep Vaidya and Chris Clinton: *Privacy-Preserving Association Rule Mining in Vertically Partitioned Data,* In The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.
13. Murat Kantarcioglu and Chris Clifton: *Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data,* In IEEE Transactions on Knowledge and Data Engineering, vol. **16**, no. 9, pp. 1026-1037, September 2004.
14. P. Paillier: Public key cryptosystems based on composite degree residuosity classes. In Advances in Cryptology—Eurocrypt '99 Proceedings, LNCS 1592, pages 223–238. Springer-Verlag, 1999.
15. The Cryptographic Library MIRACLE. Available at `http://indigo.ie/~mscott/`
16. P. S. Bradley and U. M. Fayyad: *Refining initial points for K-Means clustering,*. In Proc. 15th International Conf. on Machine Learning, pages 91–99, May 1998.

17. K. Jain, M. N. Murty, and P. J. Flynn: *Data clustering: a review,* ACM Comput. Surv., **31**(3):264–323, 1999.
18. M. Gorawski, Ł. Słabinski: *Implementation of Homomorphic Encryption in Privacy-Preserving Clustering Distributed Data*. 2nd ADBIS Workshop on Data Mining and Knowledge Discovery, Greece 2006.