Peer-to-Peer Secure Multi-Party Numerical Computation

Danny Bickson[1]

IBM Haifa Research Lab
Mount Carmel, Haifa 31905, Israel
Email: dannybi@il.ibm.com

11/9/08

[1] This research was done while D.B. was a Ph.D. candidate at the Hebrew University of Jerusalem.

Joint work

- Benny Pinkas - Haifa University, Israel
- Genia Bezman and Danny Dolev - Hebrew University, Israel

- B.P. was supported by The Israel Science Foundation (grant No. 860/06).
- D.D. was supported by The Israel Science Foundation (grant No. 0397373).

Outline

1. Introduction
   - Motivation
   - Problem model
2. Cryptographic primitives
   - Random perturbations
   - Shamir's Secret Sharing
   - Homomorphic encryption
3. Proposed constructions
   - Homomorphic encryption
   - Shamir's secret sharing
4. Experimental results
   - Distributing Koren's algorithm
   - Comparing the different protocols
5. Conclusion

Motivation

- We consider the problem of performing a joint numerical computation of some function over a Peer-to-Peer network.
- Example applications are trust computation, ranking of nodes and data items, clustering, collaborative filtering, factor analysis etc.
- The aim of *secure multi-party computation* is to enable parties to carry out such distributed computing tasks in a secure manner.

Previous approaches

- Yao generic secure protocol, FairPlay system.
  - Either centralized or require all-to-all communications.
  - Usually uses heavy asymmetric encryption.
  - Previous work scales to tens of nodes
  - Can compute any function
- This work
  - Lightweight protocol
  - Does not use asymmetric encryption
  - Scales to millions of nodes
  - Fully distributed
  - Compute a small subset of functions

Our model

- We focus in algorithms which compute weighted sums: addition, substraction and multiplication.
- Numerous algorithms: belief propagation, EM (expectation minimization), power method, gradient descent etc.
- Using the semi-honest model: parties are curios but honest. (Malicious parties may still collaborate!)
- Our goal is to provide a privacy preserving mechanism: the nodes do not learn anything at the course of computation beside the output.
- Unlike previous work, which assume a threshold on the *global* number of adversaries, we assume a threshold on the *local* number of adversaries on each vicinity.

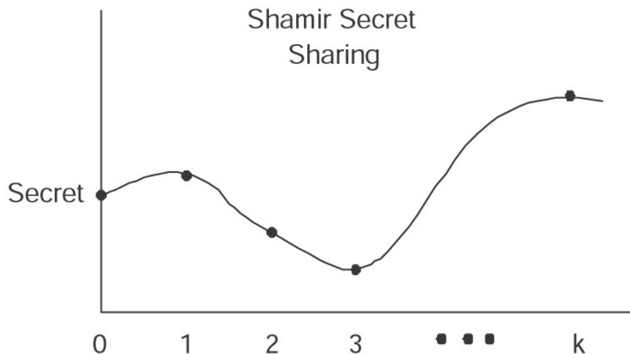
Cryptographic primitives

- Random perturbations
- Shamir secret sharing
- Homomorphic encryption

Random perturbations

- the node sends a value $u_i + v$, where u_i is the original scalar message, and v is a random value drawn from a certain distribution V.
- Perturbation adds 'noise' to the computation.
- For example: V is a distribution with zero mean, the perturbated sum of a large number of a values is close to the original sum.
- A weak technique, used mainly for performance comparison.

Shamir's Secret Sharing



- Homomorphic property

Homomorphic encryption

- There exists an efficient algorithm $+_{pk}$ whose input is the public key of the encryption scheme and two ciphertexts, and whose output is $E_{pk}(m_1) +_{pk} E_{pk}(m_2) = E_{pk}(m_1 + m_2)$.
- Namely, this algorithm computes, given the public key and two ciphertexts, the encryption of the sum of the plaintexts of two ciphertexts.
- Similarly, there exists a $\cdot_{pk}$ operation.
- We chose to use the Paillier encryption.

Main observation

- Numerous distributed algorithms compute in each node a weighted sum of scalars of the form

$$\Sigma_{j \in N_i} a_{ij} m_{ij}$$

- We propose three methods to compute this weighted sum
  - Random pertubations, SSS, homomorphic encryption.

Homomorphic encryption

Initialization

H0 The third party creates for node i a public and private key pair, $[pubk(i), prvk(i)]$. It sends the public key $pubk(i)$ to all of node i's neighbors, and splits the private key into shares, such that each node i neighbors gets a share s_{ji}.

One round of computation:

Node j would like to send a scalar value m_{ji} to node i:

H1 Encrypt the message to get $C_{ji} = E_{pubk(i)}(m_{ji})$.

H2 Send the result C_{ji} to node i.

H3 Node i aggregates all the incoming message C_{ji}, using the homomorphic property to get $E_{pubk(i)}(\sum a_{ij} m_{ji})$

Finally

Node i's neighbors assist it in decrypting the result x_i, without revealing the private key $prvk(i)$.

- H4 Node i sends all its neighbors the result computed in [H3]: $C_i = E_{pubk(i)}(\sum a_{ij}m_{ji})$.
- H5 Each neighbor, computes a part of the decryption $w_{ji} = C_i^{s_{ji}}$ where s_{ji} are node i private key shares computed in step [H0], and sends the result w_{ji} to node i.
- H6 Node i multiplies all the received values to get:

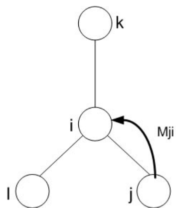
$$\Pi_{j \in N_i} w_{ji} = C_i^{\sum_{j \in N_i} s_i} = C_i^{\lambda_i} = \sum a_{ij} m_{ji} \mod N. \tag{1}$$

Homomorphic encryption protocol overhead

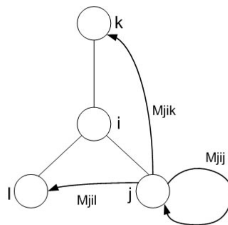
- Extra Message overhead
  - Public/private key generation - $2e$
  - Message size increased in a factor of about 1000
  - Decryption of output - e
- Computation overhead
  - One encryption to each algorithm message
  - $n(k-1)$ multiplications
  - evaluation of n random polynomials
- Protocol security derived from the Paillier encryption

SSS protocol

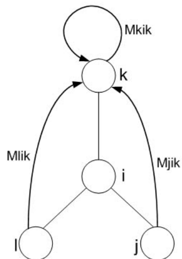
- S1 Generate a random polynomial P_{ji} of degree $d_i - 1$, of the type $P_{ji}(x) = m_{ji} + \sum_{i=1}^{d_i-1} a_i x^i$ (where $d_i \leq |N_i|$).
- S2 For each neighbor l of node i, create a share C_{jil} of the polynomial $P_{ji}(x)$ by evaluating it on a single point x_l.
- S3 Send C_{jil} to node l, which is i's neighbor.
- S4 Each neighbor l of node i aggregates the shares it received from all neighbors of node i and computes the value $S_{li} = \sum_{j \in N_i} a_{ij} P_{ji}(x_l)$. (Note that the result of this computation is equal to the value of a polynomial of degree $d_i - 1$, whose free coefficient is equal to the *weighted* sum of all messages sent to node i by its neighbors.)
- S5 Each neighbor l sends the sum S_{li} to node i.
- S6 Node i treats the value received from node l as a value of a polynomial of degree $d_i - 1$ evaluated at the point x_i.
- S7 Node i interpolates $P_i(x)$ for extracting the free coefficient, which in this case is the weighted sum of all messages $\sum_{j \in N_i} a_{ij} m_{ji}$.



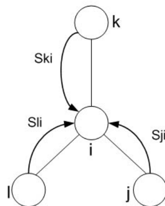
(a)



(b)



(c)



(d)

SSS protocol overhead

- Polynomial generation of k parts
- Increasing message overhead by d, message size remains the same
- One polynomial extrapolation

Example application: Netflix movie rating challenge

Koren's algorithm (ICDM 07')

- r_{ui} is the rating user u assigned movie i.
- Output rating is computed using a weighted average of the neighboring peers

$$r_{ui} = \sum_{j \in N_i} r_{uj} w_{uj}.$$

- Our goal is to find the weights matrix $\mathbf{W}$ where w_{ij} signifies the weight node i assigns node j.

Example application: Netflix movie rating challenge

Koren's algorithm (ICDM 07')

- This is a least square minimization problem for user i :

$$\min_{\mathbf{w}} \sum_{v \neq u} (r_{vi} - \sum_{j \in N_i} w_{ij} r_{vj})^2 .$$

- The optimal solution is formed by differentiation and solution of a linear systems of equations $\mathbf{Rw} = \mathbf{b}$.
- The optimal weights (for each user) are given by:

$$\mathbf{w} = (\mathbf{R}^T \mathbf{R})^{-1} \mathbf{R}^T \mathbf{b}$$

The challenge

- Computing Koren's algorithm in a Peer-to-Peer network.
- Adding a privacy preserving layer.

First challenge: distributing Koren's algorithm

- The 'trick' $\mathbf{R} \in \mathbb{R}^{m \times n}$

$$\tilde{\mathbf{R}} \triangleq \begin{pmatrix} \mathbf{I}_m & \mathbf{R}^T \\ \mathbf{R} & 0 \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}.$$

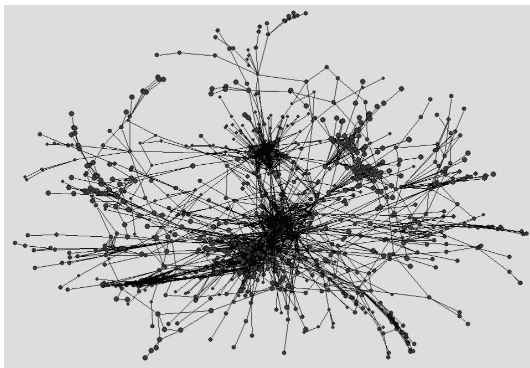
- define a new vector of variables $\tilde{\mathbf{w}} \triangleq \{\hat{\mathbf{w}}^T, \mathbf{z}^T\}^T \in \mathbb{R}^{(m+n)\times 1}$, where $\hat{\mathbf{x}} \in \mathbb{R}^{m \times 1}$ is the solution vector
- Define a a new observation vector $\tilde{\mathbf{b}} \triangleq \{\mathbf{0}^T, \mathbf{b}^T\}^T \in \mathbb{R}^{(m+n)\times 1}$.
- Solve the new linear system $\tilde{\mathbf{R}}\tilde{\mathbf{w}} = \tilde{\mathbf{b}}$, taking the first m entries
- Apply the Jacobi iterative algorithm for solving systems of linear equations

Experimental settings

- Code written in C using MPI.
- Cluster of Linux Pentium IV computers with 4GB memory.
- Implemented the Jacobi algorithm for solving systems of linear equations.

Topologies used for experimentation

Topology	Nodes	Edges	Data Source
Blogs Web Crawl	1.5M	8M	IBM
DIMES	337,326	2,249,832	DIMES
Netflix	497,759	100M	Netflix



Scheme	Operation	Time (micro second)	Msg size (bytes)
Random pert.	Adding noise	0.0783745	8
	Receiver operation	–	
SSS	Polynomial generation and evaluation	11.18382125	8
	Polynomial extrapolation	6.13709025	
Paillier	Key generation	5016199.4	2048
	Encryption	203478.62	
	Decryption	193537.97	
	Multiplication	99.063958	

Table: Running time of local operations. As expected, the Paillier cryptosystem basic operations are time consuming relative to the SSS scheme.

Topology	Scheme	Time (HH:MM:SS)	computing nodes
DIMES	None	0:33.36	1
	Random Perturbations	0:35.27	1
	SSS	10:53.44	1
	Paillier	28:44:24.00	1
Blogs	None	1:28.16	1
	Random Perturbations	1:34.85	1
	SSS	38:00.24	1
	Paillier	101:52:00.00	1
Netflix	None	5:31.14	8
	Random Perturbations	5:54.69	8
	SSS	21:40.00	8
	Paillier	-	-

Table: Running time of eight iterations of the Jacobi algorithm. The baseline timing is compared to running without any privacy preserving mechanisms added. Empirical results show that computation time of the homomorphic scheme is a factor of about 1,350 times slower then the SSS scheme.

Conclusion and future work

- We have compared three possible approaches for performing P2P SMPC.
- Out of the three examined schemes the Shamir Secret Sharing scheme is the most efficient scheme.
- The drawback of this scheme is the involvement of 2 hops neighborhood in the computation.
- Future work: handling malicious participants.

THANK YOU!