Privacy Preserving Fuzzy Reputation Management in Decentralized P2P Networks

Rishab Nithyanand[1] and Karthik Raman[2]

[1] University of California - Irvine
rishabn@uci.edu
[2] McAfee Avert Labs
karthik_raman@avertlabs.com

Abstract. The P2PRep algorithm [1, 2] is a reputation-management mechanism in which a peer uses fuzzy techniques to compute local reputations and aggregates these results to compute a global reputation for another peer which has made an offer of service. While this mechanism is known to be extremely effective in the presence of malicious peers, it has one drawback: it does not preserve the anonymity of peers in the network during the voting phase of protocol. This makes it unsuitable for use in networks which associate peers with a routing identifier such as an IP address. We propose in this paper, a solution to this problem - the 3PRep (Privacy Preserving P2PRep) algorithm which implements two protocols to maintain vote privacy in P2PRep without significant additional computation and communications overhead. In doing so, we also provide a method to compute the Ordered Weighted Average (OWA) over distributed datasets while maintaining privacy of these data.

1 Introduction

Distributed collections of peers that share resources with each other form peer-to-peer (P2P) networks. These networks are growing in popularity as a result of their excellent availability, reliability, and scalability. Often in such networks, there are no trusted central authorities to help with maintainence of the network, including with trust and reputation management. Reputation-management systems provide peers with a measure of trustworthiness of other peers in the network, thus preventing wastage of resources (that would result from interaction with malicious peers) and helping improve quality of service. As noted in previous work, the challenges of designing a decentralized reputation-management system include the following:

1. Deciding where to place reputation information securely in the absence of a trusted central authority
2. How to collect and aggregate, for a global reputation, the reputation assessments of a peer by other peers in the network
3. How to maintain the privacy of these reputation assessments.

Malicious peers might retaliate against other peers who have contributed towards the propagation of a low reputation. For example, as noted in [3] reputation assessments are often reciprocal. In addition, malicious peers might react more harshly by launching attacks such as DoS against an identified peer [4].

1.1 Contributions

In this paper we propose the 3PRep (Privacy Preserving P2PRep) algorithm which augments the P2PRep algorithm with two new protocols to preserve vote privacy without significant additional computation and communications overhead. In 3PRep, local reputation assessments (votes) are transmitted to querying peers as ciphertexts and are aggregated without the disclosure of their values. In doing so, we also provide a method to compute the Ordered Weighted Average (OWA) over distributed datasets while maintaining privacy of these data. This method is also applicable to mining statistical information from private distributed datasets, and other voting systems, where preserving the privacy of votes is paramount. The major result of 3PRep is its application to non-anonymous peer-to-peer networks where user identifiers are often routing identifiers.

1.2 Related Work

In this section, we describe related research and work in the two categories that encompass our paper - trust management and privacy preserving mining.

There has been a considerable amount of research in the area of trust management and reputation-management systems for decentralized networks. The EigenTrust algorithm [5] has been the benchmark for reputation management systems in decentralized P2P networks. It implements the pagerank [6] algorithm for a peer to peer architecture. Yao and Tamassia [7] present a private distributed scalar product protocol that can be used to derive trust from private recommendations using homomorphic encryption. Although this system preserves the privacy of recommendations made by peers, the reputation management system proposed is not highly resistant to attacks by malicious peers. The SuperTrust [8] framework manages the trust ratings of peers in a privacy-preserving manner for the super peer P2P architecture. There are other techniques to preserve privacy in reputation systems. TrustMe [9] provides anonymity to peers and their recommendations. However, its architecture requires a single central authority so it is not applicable to decentralized P2P networks. The authors of [10] propose several privacy preserving protocols for reputation systems that make use of additive aggregation. One of these is is fallible to collusion. Another is resistant to collusion but has significant computation and communications overhead of $O(n^3)$. [11] describes the architecture for an anonymous reputation management layer plug-in for decentralized P2P networks that prevents sybil attacks. [12] provides a privacy preserving method for computing trust in an expert (using the Knots model). However, it assumes that all peers in the network behave honestly. Other research work has been concentrated on reputation management for centralized networks in internet communities. [13] presents some design options for privacy preservation in reputation systems in centralized architectures. The proposals provided unlinkability and anonymity of user reputation profiles and were backed up with common information theoretic models. Other work in [14]

allows users to build and use reputations aggregated from a set of different communities which are interoperable with the reputation system, the work is illustrated with the phpBB forum architecture. Finally, [3] provides statistical evidence that reciprocal feedback distorts the production of reputation in peer to peer networks.
The concept of secure multi-party computation was first introduced by Andrew Yao [15] for two parties. Goldreich *et al.* [16] extended this for more than two parties, proving security on the assumption that private communication channels existed between each communicating party. Since then, there has been considerable work done in the area of mining statistics and information from private data held by two or more parties. Atallah and Du [17] proposed efficient and secure solutions to solve geometric problems using data from multiple parties. Several methods [18, 19] have also been proposed for privacy preserving classification of data from multiple datasets. Work in [20, 21] addresses secure mining of association rules over horizontally partitioned data. The methods incorporate cryptographic techniques and homomorphic encryption to minimize the information shared between parties. Finally, work in [22, 23] present methods for privacy preserving k-means clustering when different sites contain different attributes for a common set of entities.

1.3 Organization

In section 2, we introduce the P2PRep algorithm which forms the basis of our paper. In section 3, we present a definition of privacy and then describe two protocols for preserving vote privacy in P2PRep along with an analysis of correctness, communications and computation complexity. In section 4, we describe 3PRep - a privacy preserving alternative to P2PRep. Finally, in section 5, we make our conclusions.

2 The P2PRep Algorithm

The P2PRep algorithm consists of 5 phases - Resource Searching, Polling, Vote Cleaning, Vote Aggregation, and Resource Downloading. In the first phase, a peer (the requester) sends a broadcast message to others in the network requesting for some resource, other peers (offerers) respond with offers of service if they have the requested resource. In the Polling phase, the requester broadcasts a request for the opinions on the quality of service provided by the offerers. The peers that have interacted with the offerers in the past respond to the request by sending their opinions (essentially a vote). The requester then cleans votes based on its own experiences with the voters *i.e.* some votes may be completely discarded if the requester doesn't trust the source. Finally, the set of clean votes for each offerer is aggregated, resulting in the computation of a global reputation value of the offerer. The requested resource is then downloaded from the offerer with the highest global reputation.
In this section of the paper, we describe the reputation model of the algorithm - *i.e.* the Polling phase (where peers compute the local reputations of offerers) and the Vote Aggregation Phase (where the received votes

are aggregated to compute an offerers global reputation). The remaining sections of the paper will be dedicated to modifying these two phases to maintain privacy of the votes.

2.1 The P2PRep Reputation Model

The P2PRep Reputation model considers two levels of reputations - local and global. Local reputations denote the amount of trust one peer has in another based on its own interactions with it. Global reputations are an aggregation of many local reputations. Global reputations are used to allow new peers, or peers with insufficient interaction with the offerers to decide which offerers provide best quality of service.

Computation of Local Reputation Each local reputation (r_{ij}) is initialized after the first interaction between i and j by taking the value of $t_{ij}^{(1)}$. Where $t_{ij}^{(n)} = 1$ for a satisfactory transaction and $t_{ij}^{(n)} = 0$ for an unsatisfactory transaction. After every subsequent interaction, the local reputation is updated on the basis of the peer i's satisfaction with the interaction with peer j.

$$r_{ij}^{(n)} = \alpha(n) r_{ij}^{(n-1)} + (1 - \alpha(n)) t_{ij}^{(n)}$$

Here, $\alpha(n)$ is some "freshness" value (between 0 and 1) that is used to denote the importance of i's past interactions with j. If $\alpha(n) \approx 1$, then the reputation computed as a result of past interactions is more significant than the result of the most recent one. If $\alpha(n) \approx 0$, the past reputations carry almost no importance at all. $\alpha(n)$ must be high for the first few transactions and reduce as n increases. We will not describe the method for computation of this freshness parameter, but more information may be obtained from [2].

Computation of Global Reputation If a peer has not interacted with an offerer sufficiently enough, it enters the polling phase of P2PRep and requests all other peers in the network to send their opinions of the offerer. Once the vote (r_{kj}) expressed by each peer k for the offerer j is received by the requesting peer i, these votes need to be put together in some manner to produce a global reputation value for j. This is done using the Ordered Weighted Average (OWA) operator.

$$\lambda_{OWA} = \frac{\sum_{k=1}^{n} W_k r_{t_k,j}}{\sum_{k=1}^{n} W_k}$$

Here, n is the number of reputations that need to be aggregated, W is the weighing vector and $r_{t_1,j} \geq r_{t_2,j} \geq .... \geq r_{t_n,j}$. The weights are set asymmetrically since the aggregation operator needs to be biased towards the lower end of the interval in order to increase the impact of low-local reputations on the overall result since peers are usually assumed to be trustworthy and malicious behaviour is exceptional.
A bonus is given to multiple occurances of the same vote value. The unit

interval of all possible local reputation values is partitioned into $d+1$ intervals and their extreme values are used as the set of weights in the weighing vector W for aggregating the d distinct local reputation values. Peer i's local reputation of j may also be added to the computation by partitioning the unit interval into $d+2$ intervals and associating it with the highest possible weight ($\frac{d+1}{d+2}$). Then, j's global reputation R_j can be given by:

$$R_j = \frac{\sum_{x=1}^{d+1} (\frac{x}{d+2}) D_x |D_x|}{\sum_{x=1}^{d+1} (\frac{x}{d+2}) |D_x|}$$

where D_x denotes the x^{th} distinct vote value and $|D_x|$ denotes the number of identical votes with value D_x that were received.

2.2 Loss of Vote Privacy in the P2PRep Algorithm

During the polling phase, the requester (i) polls peers about the reputation of the offerer (j). Peers reply with messages in the form (ID, $LocalRep_j$). The ID is required to be sent along with the vote since it is required in the Vote Cleaning phase. There is no possibility for a peer to maintain privacy of its vote ($LocalRep_j$). The most fundamental requirement of any reputation or voting mechanism - anonymity, is compromised. In the following sections, we will descibe how to avoid revealing $LocalRep_j$ values while still allowing computation of R_j, thereby, making P2PRep completely privacy preserving and suitable for all decentralized peer-to-peer networks.

2.3 P2PRep vs. EigenTrust

The P2PRep algorithm is known to have significantly better ability than the EigenTrust [5] to choke malicious peers from the peer-to-peer network. Although the algorithm gets of to a slow start, it catches up and eventually surpasses the benchmark set by EigenTrust when the number of reputation queries in the network increases. This is clearly illustrated in [1]. Our modifications not effect the performance of the P2PRep algorithm, instead it just adds the important element of anonymity, which allows the P2PRep to be extended even to non-anonymous decentralized peer to peer networks.

3 Preserving Vote Privacy in P2PRep

We require the use of some semantically secure homomorphic encryption scheme (we describe the rest of the paper using the Paillier cryptosystem [24]) to allow privacy preserving computation of global reputations in the Vote Aggregation phase. We also introduce into P2PRep a concept that was introduced originally in EigenTrust - an apriori notion of trust. We assume that there exist some small percentage ($\leq 5\%$) of peers that will undertake more responsibility for the good of the network and are modeled as honest but curious peers, as are all the others. These peers

are called pre-trusted peers. All such peers have a public-private key pair (PuK, PrK), of which PuK is known to all peers in the network (the public keys of all pre-trusted peers may be distributed when a new peer joins the network). Although pre-trusted peers will be used in the process of reputation computation, we will keep their involvement to a minimum. These pre-trusted peers are usually the peers which were involved during the setup and initialization of the P2P network.
Our definition of privacy is similar to the definition of privacy used in some distributed data-mining operations [25]. It can be defined by:

$$|Pr(B \text{ learns } T|\mathcal{P}_{AB}) - Pr(B \text{ learns } T|\mathcal{P}_{BC})| \leq \epsilon$$

where, $\mathcal{P}_{XY}$ denotes some privacy preserving protocol executed between the parties X and Y. The client, X, is the party requesting some service (statistics, mining, etc.) on its encrypted data from the Server, Y. Here, Y is the party in possession of the decryption key. $Pr(Y \text{ learns } T|\mathcal{P}_{XY})$ denotes the probability of the server, learning some information T from the client's data during the execution of $\mathcal{P}_{XY}$. ϵ is some value that we aim to minimize.
Essentially, our definition of privacy states that a protocol $\mathcal{P}$ between two parties, a client and a server, preserves $\epsilon-$privacy iff the probability of the server Y learning anything from the client X, that he wouldn't have learnt by playing the role of the client himself with some other server Z is less than or equal to ϵ.

3.1 The Paillier Cryptosystem and Its Homomorphic Properties

The Paillier cryptosystem [24] is an $IND\text{-}CPA$ secure public key encryption scheme based on the composite residuosity class problem[1] that is believed to be intractable.

Homomorphic Properties: The paillier scheme is an additively homomorphic scheme which has the following very useful homomorphic properties:

1. $e(m_1) \times e(m_2) = e(m_1 + m_2)$, where m_1 and m_2 are two messages (plaintext) which belong to the message space M and $e(m)$ denotes the encryption (ciphertext) of a message m under the paillier encryption algorithm.
2. $e(m_1)^c = e(m_1 \times c)$, where m_1 is some message (plaintext) which belongs to the message space M, $e(m)$ denotes the encryption (ciphertext) of the message m under the paillier encryption algorithm, and c is any constant.

[1] Given a composite n and an integer z, it is hard to compute y such that $z=y \bmod n^2$.

Cost of Computation: The paillier cryptosystem executes with costs that are comparable with other public-key cryptosystems such as RSA and El-Gamal. The key generation process requires constant time and needs to be executed just once in our protocols. The encryption algorithm requires the product of two exponentiations in $\mathbb{Z}_{N^2}$, with exponent N, where N is the product of two large primes. The encryption of a single message requires 6 primary constant time operations. The decryption algorithm requires essentially just the cost of one exponentiation in $\mathbb{Z}_{N^2}$. The process of decryption of a single ciphertext requires 14 primary constant time operations. A more detailed analysis of the computational costs, effective speed-ups, along with comparisons to other public key cryptosystems such as RSA and El-Gamal may be found in [24, 26].

3.2 Vote Preparation in the Polling Phase

When a requester i, enters the polling phase, it sends a *Poll* message to all the other peers in the network requesting for their reputation of some offerer j. The message is of the form $\{i, j, ptpeer, PuK_i\}$ where *ptpeer* is any pre-trusted peer chosen by i such that $ptpeer \neq i$. Recipients of the message check if they have some local information regarding the reputation of j. If they do, they encrypt their vote (LocalRep) using the public key PuK of the pre-trusted peer mentioned in the message and reply to i with a *PollReply* message of the form $e\left\{ID, e\left\{LocalRep\right\}_{PuK_{ptpeer}}\right\}_{PuK_i}$.
At the end of this phase, i decrypts all the *PollReply* messages, from which receives a collection of IDs and encrypted votes. The votes can then be cleaned, and a small subset can even be verified to detect spoofing attacks as suggested in Vote Cleaning phase described in [2]. Once the cleaning and verification process is complete, we enter the Vote Aggregation phase. In order to compute the global reputation value of a peer j in the vote aggregation phase, we need to perform the following steps:

1. Compute the number 'd' of distinct votes and the frequency of each distinct vote received by the requester. The set of distinct votes will be represented by D and their frequencies by $|D_y|$ for each $y \in \{1, ..., d\}$.
2. Compute the weight value W_y given by $(\frac{y}{d+2})|D_y|$ and then find $\sum_{y=1}^{d+1} W_y$ and the global reputation value $\frac{\sum_{y=1}^{d+1} W_y D_y}{\sum_{y=1}^{d+1} W_y}$.

3.3 Protocol 1: Counting the Number of Distinct Votes and Frequency of each Vote

After the vote cleaning phase, and at the start of the vote aggregation phase, the first operation the requester peer i will need to perform is - counting the number of distinct votes (or reputations) received and the frequency of each distinct vote *i.e.* compute d, D_x, and $|D_x|$ for all $x \in \{0, ..., d\}$ from the set of (say) n encrypted votes $(e(v_1), ..., e(v_n))$ that it received. To do this, we perform the following steps described below:

1. **Step I: Computing the Encrypted Vote Difference List**
   i computes $e(v_x) \times e(v_y)^{-1}$ ($= e(v_x - v_y)$) for all $x, y \in \{1, ..., n\}$ where $x < y$. We call this sequence α. Elements of the sequence are labeled by the pair (x, y), where, as before $x < y$. i then permutes this sequence randomly to obtain a new sequence α'. Elements of this sequence are labeled by the pair (p, q) such that $PERMUTE(x, y) = (p, q)$.
2. **Step II: Computing the Permuted +1/0/-1 Sequence**
   i sends α' to the pre-trusted peer *ptpeer* which decrypts it and generates another sequence β' such that the p, q^{th} element of β' is 0 if the decrypted value of the p, q^{th} element of α' is 0. If the p, q^{th} value of α' is +ve, then the p, q^{th} element of β' is assigned the value +1, otherwise a value of -1 is assigned. i.e.

$$\beta'_q = \begin{cases} 0 & \text{if } decrypt(\alpha'_{p,q}) = 0 \\ +1 & \text{if } decrypt(\alpha'_{p,q}) > 0 \\ -1 & \text{if } decrypt(\alpha'_{p,q}) < 0 \end{cases}$$

3. **Step III: Computing the Original +1/0/-1 Sequence Matrix**
   i receives the permuted sequence of +1/0/-1's ($\beta'_{p,q}$) and removes the earlier applied permutation to obtain $\beta_{x,y}$. This sequence is then used to construct an $n \times n$ matrix (M). The element $M_{x,y}$ is given by $\beta_{x,y}$ and $M_{y,x} = -M_{x,y}$.
4. **Step IV: Sorting and Counting Using the Matrix**
   Now, i needs to compute the frequency of occurance of each distinct vote. To do this, it computes the sum of each row of the matrix M, *i.e.* i computes $Sum_x = \sum_{y=1}^{n} M_{x,y}$ for all $x \in \{1, ..., n\}$. i first sorts this sequence to obtain a $SortedSum$. It then counts the number of distinct values that occur in $\{SortedSum_1, ..., SortedSum_n\}$ and the number of occurances of each value, thereby obtaining d, $e(D_x)$, and $|D_x|$ where $x \in \{1, ..., d\}$ and $D_1 > D_2 > ... > D_d$.

Highlight of Protocol 1: Protocol 1 solves the following problem - Assume a party A has a set of n encrypted numbers $E = \{e(x_1), ..., e(x_n)\}$ and party B has the private key PrK to perform decryption of this set. How does A first sort and then compute the frequency of occurance of each number in the set $X = \{x_1, ..., x_n\}$ without revealing the numbers in X to B (*i.e.* without simply sending the set E to B for decryption) while ensuring that B obtains no vital information, even in spite of possessing PrK?

Correctness Analysis of Protocol 1: To illustrate the correctness of Protocol 1 clearly, we make use of the following example: In our scenario, we let A and B be the requester and pre-trusted peer respectively. A has the encrypted set of four ratings (or votes) $e(V) = \{e(75), e(50), e(90), e(50)\}$.
A computes $\alpha = \{e(+25), e(-15), e(+25), e(-40), e(0), e(+40)\}$. Using some permuting function $PERMUTE(x, y) \longmapsto (p, q)$, A now computes $\alpha' = \{e(-40), e(0), e(+40), e(+25), e(-15), e(+25)\}$ which is sent to B.

B decrypts this sequence to obtain the permuted set of differences β' $\{-4, 0, +4, +2, -2, +2\}$ and gains no vital information regarding the original set of votes that it couldn't have obtained by playing the role of the requester itself (ofcourse in that case, he wouldn't know PrK!). Now B computes $\beta' = \{-1, 0, +1, +1, -1, +1\}$. A receives β' from B and using $PERMUTE^{-1}$ $(p, q) \longmapsto (x, y)$, obtains $\beta = \{+1, -1, +1, -1, 0, +1\}$. The matrix M is given by

$$M = \begin{pmatrix} 0 & +1 & -1 & +1 \\ -1 & 0 & -1 & 0 \\ +1 & +1 & 0 & +1 \\ -1 & 0 & -1 & 0 \end{pmatrix} \begin{matrix} RowSum : +1 \\ RowSum : -2 \\ RowSum : +3 \\ RowSum : -2 \end{matrix}$$

A obtains $SortedSum = \{+3, +1, -2, -2\}$ from which it can sort the votes it received to obtain $Sorted - e(V) = \{e(90), e(75), e(50), e(50)\}$. By counting the occurance of elements in $SortedSum$, A obtains $d = 3$, and also finds $|D| = \{1, 1, 2\}$, $e(D) = \{e(90), e(75), e(50)\}$.

Computation and Communication Cost Analysis of Protocol 1: The cost of computation may be split into the following: (1) Cost of computing the set of vote differences at peer i: $O(n^2)$, (2) Cost of permuting the sequence of vote differences at peer i: $O(n)$, (3) Cost of decryption of the permuted sequence at peer *ptpeer*: $O(n^2)$, (4) Cost of computing the original sequence of values at peer i: $O(n)$, (5) Cost of computing the +1/0/-1 matrix at peer i: $O(n^2)$, (6) Cost of computing the sum of the rows of the matrix at peer i: $O(n^2)$, (7) Cost of computing the sorted sum of the rows at peer i: $O(n \log n)$, (8) Cost of counting the number of distinct elements in the sorted sum at peer i: $O(n)$, (9) Cost of computing the frequency of occurrence of each distinct element at peer i: $O(n)$. Total cost of computation at peer i (from 1, 2, and 4 - 9) is $O(n^2)$, and the cost of computation at peer *ptpeer* (from 3) is also $O(n^2)$.
The communications cost may be split into the following: (1) i sends *ptpeer* the permuted sequence of vote differences (in ciphertext): $O(n^2)$, (2) *ptpeer* sends i the +1/0/-1 sequences for the n^2 differences: $O(n^2)$. Total communications overhead that results from protocol 1 for the network: $O(n^2)$.

3.4 Protocol 2: Computing the Global Reputation Value

At the end of the first protocol, peer i has the following data available to it: d, a sorted list of $e(D_x)$ with the corresponding $|D_x|$ for all $x \in \{1, ..., d\}$. Using this information, we need for peer i to (1) compute the weight vector W, include its own opinion of the offerer in that vector, then compute the value of its aggregate, (2) compute the product of the encrypted votes and the weight vector and the aggregate of the resulting vector - the WD vector, and finally (3) compute the global reputation of the offerer.

1. **Step I: Computing the Weight Vector and Its Aggregate:** Peer i computes $W_x = \frac{x}{d+2} \times |D_x|$ for all $x \in \{1, ..., d\}$. Recall that we do this to ensure that higher importance is given to lower votes since peers are more likely to be trustworthy than not. However, if peer i has some local reputation value of its own, it should assign $W_{d+1} = \frac{d+1}{d+2}$ and $e(D_{d+1}) = e(v_{ij})$ where v_{ij} denotes peer i's own locally computed reputation of the offerer j. We do this to ensure that i's vote receives highest priority - regardless of its value. Computing the aggregate now requires only a simple $\sum_{x=1}^{d+1} W_x$.
2. **Step II: Computing the WD Vector and Its Aggregate:** At this point, peer i has the plaintext values of W_x and $\sum W$, the encrypted values of $e(D_x)$ for all $x \in \{1, ..., d+1\}$. Peer i now computes $e(D_x)^{W_x}$ to obtain $e(W_x \times D_x)$ for all $x \in \{1, ..., d+1\}$. To compute $e(\sum_{x=1}^{d+1} W_x D_x)$, we multiply all the ciphertexts obtained from previous operations - *i.e.* $e(W_1 D_1) \times e(W_2 D_2) \times ... \times e(W_{d+1} D_{d+1}) = e(W_1 D_1 + W_2 D_2 + ... + W_{d+1} D_{d+1})$.
3. **Step III: Compute the Global Reputation of Offerer j** Peer i uses the value $e(\sum_{x=1}^{d+1} W_x D_x)$ obtained in the previous step to compute R_j. This is done by computing

$$e(\sum_{x=1}^{d+1} W_x D_x)^{(\frac{1}{\sum_{x=1}^{d+1} W_x})}$$

from which i obtains:

$$e(\frac{\sum_{x=1}^{d+1} W_x D_x}{\sum_{x=1}^{d+1} W_x}) = e(R_j)$$

This value is then sent to the pre-trusted peer *ptpeer* for decryption, to obtain R_j. *ptpeer* may then send a message to peer i informing it of the latest global reputation value computed for the offerer j.

Highlight of Protocol 2: Protocol 2 allows a party to compute the aggregate and average of encrypted values for which it does not have the ability to decrypt. In addition, when combined with protocol 1, it solves the following problem - Assume a party A has a set of n encrypted numbers $E = \{e(x_1), ..., e(x_n)\}$ and party B has the private key PrK to perform decryption of this set. How does A compute the Ordered Weighted Average (OWA) of the set $X = \{x_1, ..., x_n\}$ while allowing lower values of X to receive higher weights and vice versa, and giving a bonus to the values in X which occur repetitively, without allowing the decryptor B obtain more information than itself? Our protocols solve this problem while keeping the communication between A and B at a minimum.

Correctness Analysis of Protocol 2: To illustrate the correctness of Protocol 2 clearly, we continue the illustration of our example from Protocol 1. Recall, at the end of Protocol 1 we had - $d = 3$, $|D| = \{1, 1, 2\}$, and $e(D) = \{e(90), e(75), e(50)\}$. In step 1, A computes $W = \{\frac{1}{5} \times 1, \frac{2}{5} \times 1, \frac{3}{5} \times 2\}$ and then sets $W_4 = \frac{4}{5}$, D_4 to his own

computed local reputation for the offerer (say 60), to do this, he sets $e(D_4) = e(60)$. Finally, A computes $\sum W = \frac{1+2+6+4}{5} = 2.6$. In step 2, A computes $e(90)^{\frac{1}{5}}, e(75)^{\frac{2}{5}}, e(50)^{\frac{6}{5}}, e(60)^{\frac{4}{5}}$ to obtain the set of values in the vector $e(WD) = \{e(18), e(30), e(60), e(48)\}$. On performing $e(18) \times e(30) \times e(60) \times e(48)$, A obtains $e(156)$. Finally, in step 3, A computes $e(156)^{\frac{1}{2.6}}$ to obtain $e(\frac{156}{2.6})$, on decryption by B, the global reputation of the offerer is computed to be 60.

Computation and Communication Cost Analysis of Protocol 2: The cost of computation may be split into the following: (1) Cost of computing the elements of the weight vector and its aggregate at peer i: $O(n)$, (2) Cost of computing the elements of the WD vector at peer i: $O(n)$, and all other operations in the protocol occur with constant computation cost. Total cost of computation at peer i is $O(n)$, and the cost of computation at peer *ptpeer* is $O(1)$. The communications costs for protocol 2 are also of the order of $O(1)$.

4 3PRep: Privacy Preserving P2PRep

In this section, we re-write the P2PRep protocol to allow for our privacy preserving protocols to be incorporated. We will do this by describing the functioning of each of the 5 phases in 3PRep.

4.1 Phase I: Resource Searching in 3PRep

The first phase of 3PRep (and P2PRep) is the resource location phase. Here, a querying peer i attempts to find a set of offerers of some resource that it requires. The peer i may reduce the size of this set by providing in it search query, a set of minimum requirements that it requires from the offerers (such as a minimum bandwidth requirement). i sends a broadcast message of the form *Query*(min_req, resource_req) to all peers in the network.
Once the peers receive this message, if they have the resource required by i and they meet the minimum requirements, they reply with a message of the form *QueryHit*(ID, bandwidth, hits). i uses the message to compile a list of offerers.

4.2 Phase II: Polling in 3PRep

During the polling phase of 3PRep, the querying peer i selects using some metric, an offerer o from the list obtained in the Resource Searching phase. At this point before downloading the resource, it needs to find out if o is trustworthy or not. To do this, it first selects some bootstrapping peer (as the pre-trusted peer - *ptpeer*). Once this is done, i sends out a broadcast message of the form *poll*(i,o,ptpeer, PuK_i) to all peers in the network. Every peer that receives this message checks their reputation log to check if they have a locally computed reputation for peer o as a

result of past interactions.
If they have not interacted with o, they ignore this request. If they have a history with the peer o, they look into their repository to find the public-key (PuK_{ptpeer}) of the peer *ptpeer* listed in the message and then reply to peer i with a message of the form:

$$\textit{poll-reply}\left\{ e\left\{ ID, e\left\{ local\_rep_o \right\}_{PuK_{ptpeer}} \right\}_{PuK_i} \right\}.$$

4.3 Phase III: Vote Cleaning in 3PRep

After the querying peer i receives the encrypted values of other peer's local reputations for the peer o, it enters the vote cleaning phase in which it (1) filters out replies obtained from peers that it does not trust, and (2) selects some subset of votes which it will verify, to ensure that vote spoofing or modification is not being carried out. While performing (1) is trivial even with encrypted votes (since IDs in the poll-reply message are in plaintext), (2) is a little more complicated.
First, the peer i needs to find a set of votes for which it will run a verification procedure. The number of votes to verify may be calculated based on a fast-start, fast-recovery algorithm such that, when a peer enters the vote cleaning phase for the first time, he verifies the maximum number of votes possible (there is a network limit on the number of votes any peer may verify - Threshold_max). If all the votes are correctly verified and there is no vote spoofing, the peer verifies just half the number of votes during the next cleaning phase (until it reaches a lower limit - Threshold_min). However, if there is some spoofing detected, it doubles the number of votes to be verified during the next cleaning phase.
When a peer j is asked to verify a vote that was sent by him (to i for o and using *ptpeer*'s PuK), he is sent a message of the form - $Verify$ (i, o, *ptpeer*, $e\left\{ local\_rep_o \right\}_{PuK}$). Since the paillier encryption scheme is randomized, we use either of the following methods for vote verification

- **Method One - Using a Vote Cache:** Every peer j records the $e\left\{ local\_rep_o \right\}_{PuK}$ value that it sent in previous poll-reply messages. If the $Verify$ message has the same encrypted value that is stored in its logs, peer j replies positively, otherwise, he reports that a modification or spoofing attack occured on his vote. This method is the simplest solution to the problem, and does not require any major communication or computation overhead. However, it requires peers to store their previous encrypted votes - thus increasing storage costs.
- **Method Two - Invoking a Pre-Trusted Peer:** When a peer j receives the $Verify$ message, it first extracts the $e\left\{ local\_rep_o \right\}_{PuK}$ value from it. Next, it re-ecrypts its stored local reputation using the same PuK to obtain $e\left\{ verify\_rep_o \right\}_{PuK}$. It then computes - $e\left\{ local\_rep_o \right\}_{PuK} \times e\left\{ -verify\_rep_o \right\}_{PuK}$ and sends it to *ptpeer* where it is decrypted. If the decrypted value is non-zero, *ptpeer* informs i and j of the attack on the vote, otherwise, *ptpeer* sends a positive report to i. This method requires no extra storage overhead, but it does require extra communication costs since there are multiple additional messages sent between i, j, and *ptpeer*.

4.4 Phase IV: Vote Aggregation in 3PRep

Once peer i has a clean, verified collection of encrypted about peer o, it needs to compute the Ordered Weighted Average of these votes, giving more importance to low reputation votes, and recurring vote values. This is a challenge since peer i is unaware of the decryption key PrK. At this point, i invokes protocol 1 to learn the number of distinct votes and their frequency of occurance. i needs to communicate with *ptpeer* just once during this protocol. i then invokes protocol 2 to compute the final global reputation value of the offerer o. This is again done with just one interaction between i and *ptpeer*, minimal communication and computation costs, and reasonable storage costs.

4.5 Phase V: Resource Downloading in 3PRep

If peer i is satisfied with the computed global reputation for the offerer o, he starts downloading his resource from o. After the transaction is complete, he updates his locally computed reputation for o depending on his satisfaction with the interaction. This locally computed reputation is updated using the fuzzy technique described earlier in section 2.1.

5 Conclusions

In this paper, we presented several useful contributions towards the computation of reputation in decentralized peer-to-peer networks and privacy preserving statistics computation in a distributed environment. First, we presented the 3PRep algorithm, a reputation-management system that is applicable to even non-anonymous decentralized peer-to-peer networks where user identifiers may be rounting identifiers. The performance of the system in the presence of malicious peers remains exactly the same as the original (known to be very effective) P2PRep algorithm that it is based upon, since the reputation computation metrics are unchanged. We also presented a method for computing the Ordered Weighted Average of data in a distributed environment while maintaining their privacy. The method for privacy preserving computation of the Ordered Weighted Average may also be applied to other voting, recommendation, and statistical computation applications, where data from certain parties are given more weight than others depending on their values (i.e. weights are required to be directly/indirectly proportional to the value of the data held) and the privacy of the individual datasets must be respected.

References

1. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: Managing and sharing servents' reputations in p2p systems. IEEE Transactions on Knowledge and Data Engineering **15**(4) (2003) 840–854

2. Aringhieri, R., Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems. JASIST **57**(4) (2006) 528–537
3. Bolton, G.E., Greiner, B., Ockenfels, A.: Engineering Trust - Reciprocity in the Production of Reputation Information. SSRN eLibrary (2009)
4. El Defrawy, K., Gjoka, M., Markopoulou, A.: Bottorrent: misusing bittorrent to launch ddos attacks. In: SRUTI'07: Proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the internet, Berkeley, CA, USA, USENIX Association (2007) 1–6
5. Kamvar, S.D., Schlosser, M.T., Garcia-molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: In Proceedings of the Twelfth International World Wide Web Conference, ACM Press (2003) 640–651
6. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Netw. ISDN Syst. **30**(1-7) (1998) 107–117
7. Yao, D., Tamassia, R., Proctor, S.: Private distributed scalar product protocol with application to privacy-preserving computation of trust. In: IFIP International Federation for Information Processing. Volume 238., IFIP, Springer (2007) 1–16
8. Dimitriou, T., Karame, G., Christou, I.: Supertrust: a secure and efficient framework for handling trust in super-peer networks. In: PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM (2007) 374–375
9. Singh, A., Liu, L.: Trustme: anonymous management of trust relationships in decentralized p2p systems. In: Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference on. (2003) 142–149
10. Pavlov, E., Rosenschein, J.S., Topol, Z.: Supporting privacy in decentralized additive reputation systems. In: iTrust. (2004) 108–119
11. Müller, W., Plötz, H., Redlich, J.P., Shiraki, T.: Sybil proof anonymous reputation management. In: SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication netowrks, New York, NY, USA, ACM (2008) 1–10
12. Gudes, E., Gal-Oz, N., Grubshtein, A.: Methods for computing trust and reputation while preserving privacy. In: DBSec. (2009) 291–298
13. Steinbrecher, S.: Design options for privacy-respecting reputation systems within centralised internet communities. In: SEC. (2006) 123–134
14. Pingel, F., Steinbrecher, S.: Multilateral secure cross-community reputation systems for internet communities. In: TrustBus. (2008) 69–78
15. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. (1982) 160–164
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. (1987) 218–229
17. Atallah, M.J., Du, W.: Secure multi-party computational geometry. In: WADS2001: Seventh International Workshop on Algorithms and Data Structures, Providence, Rhode Island (August 2001) 165–179

18. Agrawal, R., Srikant, R.: Privacy preserving data mining. In: Proceedings of the ACM SIGMOD Conference on Management of Data, Dallas, TX (May 2000) 439–450
19. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: Proceedings of the IEEE International Conference on Privacy, Security and Data Mining, Maebashi City, Japan, Australian Computer Society, Inc. (December 2002) 1–8
20. Kantarcioglu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. In: ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02). (June 2002)
21. M. Atzori, F. Bonchi, F.G., Pedreschi, D.: Blocking anonymity threats raised by frequent itemset mining. In: Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05). (November 2005)
22. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C. (August 2003) 206–215
23. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, New York, NY, USA, ACM Press (2005) 593–599
24. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT. (1999) 223–238
25. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, London, UK, Springer-Verlag (2000) 36–54
26. Catalano, D., Gennaro, R., Howgrave-Graham, N., Nguyen, P.Q.: Paillier's cryptosystem revisited. In: ACM Conference on Computer and Communications Security 2001, ACM Press (2001) 206–214