Privacy Preserving Reputation Systems

Salam HAJJAR

Encadré par: Prof. Lionel BRUNIE
En correspondance avec M. Omar HASAN doctorant 3ème année INSA de Lyon
LIRIS – INSA de Lyon Bâtiment 501 69621 Villeurbanne, France
{Salam.hajjar, omar.hassan, lionel.brunie}@insa-lyon.fr

Résumé. Nos travaux s’intéressent à la problématique de l’évaluation de la réputation d’un membre d’une communauté pair à pair (P2P). Le concept de réputation est mis en œuvre dans de très nombreuses applications, qu’il s’agisse de sites de e-commerce, de protocoles de gestion de réseaux mobiles ou d’heuristiques de gestion de plates-formes P2P. Dans ce rapport, nous proposons deux algorithmes de calcul de réputation. Le premier algorithme, permet de réaliser un calcul fiable et sécurisé de la réputation en dépit de l’état dynamique du réseau (déconnexions fréquentes de pairs). Le second offre un mécanisme de calcul préservant la confidentialité des avis donnés par les pairs participant au protocole. Ces deux algorithmes s’appuient sur une structure totalement décentralisée et ont été démontrés robustes à la plupart des types d’attaques. Ils offrent un coût optimisé en terme de bande passante en raison de la petite taille des messages échangés et de la complexité linéaire en nombre de messages, plus faible que celle des algorithmes concurrents.

Abstract. Reputation systems provide mechanisms for multiple parties, in a community, to quantify the trust between one another and to encourage adherence to contract in e-commerce. In this thesis we provide two protocols to calculate the reputation value of participant agents in peer to peer community, using a private, additive and fully decentralized manner. The first protocol considers the churn rate in the network, where peers enter and leave the network very often and the second protocol solves the problem of the leak of trustworthiness among peers. We also provide an analyzing framework to compare our algorithms to former ones in the domain of reputation systems. We refer that our proposals are resilient against the semi-trusted and curious peers attacks and they both have linear complexity.

Key words: Trust, Reputation, Privacy, Feedback, Peer to Peer System, Adversaries and colluding agents.
Table of contents

1. Introduction
   1.1. What is a Reputation System  3
   1.2. Objective  4
   1.3. Definitions  4

2. Related Work
   2.1. Anonymity in Reputation System  5
   2.2. A Reputation System with Privacy and Incentive  5
   2.3. Calculating Reputation values using Secure Sum Algorithm  6
   2.4. Secure Sum Algorithm proposed by Elan Pavlov  6
   2.5. Calculating Reputation using 2 Rounds Algorithm  7
   2.6. Splitting key Protocol  8

3. Problems and Challenges

4. Solutions
   4.1. The Fault Tolerant Reputation Protocol  9
   4.2. Blind Calculation Protocol  18

5. Analysis Framework
   5.1. Calculation Structure  24
   5.2. Privacy Preserving and Attack Defense  24
   5.3. Connection Status  26
   5.4. Analysis  26
   5.5. Analysis Observations  28

6. Conclusion and Future Work  28

7. References  28
1 Introduction

Every day when people go shopping, they choose almost the same shops where they trust the quality of goods. In e-commerce, the context is totally different as customers often do not have any previous experience of the reliability of the sellers. Some people regret the transaction they did because of misleading descriptions of goods or even because of the time of shipping, which were far from what they expected.

No one can deny that Internet brings new opportunities for business and communication, and facilitates daily actions, but, in the same time we find ourselves interacting with strangers and unknown people without the warranty of trustworthiness or authorities to mediate the interactions.

Reputation systems attempt to help people in choosing which service provider is the most trusted depending on his past behavior. They rely on the feedback given by people who had experience with this service provider.

1.1. What is a Reputation System?

Reputation systems are mechanisms that emerged to provide prediction of the future behavior of agents participating in a communication society. The auctioning site eBay employs one of the earliest and best known Internet reputation systems. It gathers comments from 212 millions of buyers and sellers about each other after each transaction [19].

Peer to peer File-sharing networks like Gnutella [8] use the reputation concept to avoid the free-rider problem, i.e. the selfish consumption of common resources from some users, who do not contribute to the system. The EigenTrust Algorithm [12] is also proposed to provide reputation in P2P file-sharing networks. Basically, it depends on the concept of trust transitivity, i.e. if an agent x trusts y it would consequently trust all agents trusted by y.

Another reputation system is used in peer to peer ad-hoc mobile networks [4]. It detects and isolates misbehaving nodes, depending on the observations of the node’s neighbors, i.e. if peer A asked for a service from peer B and peer B refused to respond, then peer A will consider this as a misbehaving action from B’s part and inform all its neighbors about that; as a result, all peers who trust the judgment of peer A will consider B as distrusted node.

Statistics applied against eBay [19] show the high convergence between feedbacks provided by agents contacting each other; this can be referred to the fear of reciprocation. The fear of revenge leads agents not to give true feedback values. For this reason, Privacy Preserving Reputation Systems aim to compute reputation scores of peers while keeping the identity of feedback providers hidden from other agents.

Protocols like the secure sum and two rounds protocol [9] were suggested to calculate the reputation of agents in decentralized additive manner. However, these algorithms vary in complexity and in privacy degree.
1.2. Objective

Our goal in this research is to provide reputation algorithms that return a global reputation of a peer, by accumulating feedback values of other peers in a decentralized manner, with a reasonable cost and by keeping the individual feedback of each peer hidden from all other peers.

1.3. Definitions

1.3.1. Trust: In the domain of e-commerce and online transactions, trust is defined as the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends [10].

1.3.2. Centralized vs. Decentralized structures: In a centralized structure transactions are settled through a mediator or central authority (CA). Peers involved must subscribe to the central authority, thus, all feedback values would be sent to the CA; this latter must be trusted by all agents so that it can answer the reputation queries. Contrariwise, decentralized structures are characterized by the absence of the CA concept, i.e. all peers are similar and have the same privileges, each agent stores locally the feedback it has about other agents, and gives it to an inquiring agent when it is demanded [10].

1.3.3. Privacy in Reputation Systems: Preserving privacy in reputation system means ensuring that the identities of feedback providers are publicly hidden, neither the inquirer nor any other peers in the system can learn if agent x has negative or positive opinion about agent y. Therefore, peers can give honest feedback about each other without the fear of retaliation.

1.3.4. Malicious agent: Participants, who attempt to subvert the protocol or reveal the local feedback values of other agents, are called malicious agents or adversaries. They achieve their goals either by doing some malicious actions or by colluding with other agents. In [7] authors categorize adversaries regarding their behavior during the calculating protocol:

Semi-disruptive agents: this type of agents follow the steps of calculating protocol correctly; however in the same time, they are curious to learn the local feedback values provided by others, thus, they may lonely perform some actions or collude with others to satisfy their curiosity.

Disruptive agents: these are more dangerous peers, since they have beyond desires against the reputation system itself, i.e. they may execute any malicious action during their participation in the protocol, not only to break the privacy of others, but, also to fail the protocol.

In this paper we only consider the presence of curious, semi-disruptive, agents among the participant peers.
1.3.5. Examples of semi-disruptive adversaries
In centralized structure, a malicious agent X can very simply ask the central authority about the feedback given by another agent Y (if such request is allowed); furthermore, the CA can be considered as curious agent since it can observe all values sent to it. Oppositely, in decentralized (P2P) systems, agents need to collude with each other and also with the inquirer to discover the reputation value given by a peer.

2 Related Work

In this section we summarize the protocols proposed in the literature of reputation and we seek into the details of protocols which are close to our contribution

2.1. Anonymity in Reputation Systems
Elli Androulaki et al [1], propose a reputation schema for anonymous environment. Their contribution combines the advantages of anonymity and reputation, basing on the concept of e-coins. Each peer has a number of initial points (e-coins). He can create pseudonyms of himself under the condition of giving each one some points. This mechanism bounds the number of pseudonyms an agent can produce. Peers also provide points to each other using their pseudonyms, thus, honest agents will not need to create pseudonyms, rather, they would prefer to keep one identity with high reputation, whereas, dishonest agents won’t receive points from others so that they won’t be able to create large number of pseudonyms.

2.2. A Reputation System with Privacy and Incentive[2]
The main goal of this system is to encourage both buyers and sellers to provide rating after their transactions take place.
To achieve this goal, authors propose to grant a discount token to participants when they provide feedback about others. To avoid the action of giving dishonest feedback values by peers, these latter are willing to use their pseudonyms instead of their real IDs. Each peer registered in the system receives a smart card prepared by a card issuer and then he can generate an anonymous identity. Peers provide the feedback values to a merchant. This latter aggregates and collects values and also gives each feedback provider a discount token to use in a future transaction.

2.3. Calculating Reputation values using Secure Sum Algorithm
Secure sum is considered as the simplest protocol to calculate the reputation value of peers located in P2P community. Summation process is done in decentralized manner while preserving the privacy of local values [7, 5].
The reputation value of a target agent is calculated as follows: bunch of agents collaborate and calculate the sum of their local feedback values about a target agent as shown in figure 1.

A positive point in secure sum protocol is its linear complexity. It can be always handled by the networks. Its simple formula makes it easy to be understood and followed correctly by participants. However, it still suffers from the collusion challenge, where curious agents can exchange some messages, out of the protocol, to reveal the local value of an agent, who will be considered as victim. This problem was solved in [7] by using secure sum protocols with two phases. In the following section we summarize both protocols of E. Pavlov et al and O. Hasan, and highlight the differences between them.

2.4. Secure Sum Algorithm proposed by Élan Pavlov [17]

This protocol preserves the privacy of individual values in the presence of curious, but un-colluding agents, since \( q \), the querying agent, initializes an ordered list of source agents and sends each agent the id of its successor. \( q \) starts the protocol by choosing randomly an agent from the list and sends it \( r = y \); where \( r \) represents the reputation variable and \( y \) is an initiative perturbing value.

Each receiving agent \( a_i \) adds its local feedback to \( r \) and forward it to \( a_{i+1} \), the last agent in the list will send the value to the querying agent, this latter subtracts \( y \) to obtain the pure reputation value.

An important assumption that we need to underline is the using of SSL (Secure Socket Layers) which attain secured exchange of information between two agents. Since the list of resources is ordered, the querying peer can organize it in a specific order to attack some peers. Figure 2 shows how \( q \) can collude with some agents to reveal the local feedback value of \( s_4 \).
2.5. Calculating Reputation using 2 Rounds Algorithm [7]

In this protocol authors deal with the collusion issue in more sophisticated manner, by proposing a perturbation value in each calculating step. The protocol starts when the querying agent q asks the target agent t about the list of resources' agents, i.e. those who have previous experience with the target peer (t). t will forward this list to q as $S_t = \{s_1, s_2, \ldots, s_n\}$ where $s_1, s_2, \ldots, s_n$ represent the source agents as shown in figure 3 step 1.

The first agent in $S_t$ receives a value $r = 0$ from q, updates it by adding its local feedback $lat_1$, and a local noise $ln_1 = [-y, +y]$. It then forwards the final value to the most trusted agent in the list $S$ after removing itself from the list $S$ which consequently contains only the remaining peers. The benefit from using the local noise is to perturb the true value.

$$r_i = r_{i-1} + lat_i + ln_i$$

Each agent that receives the message will do the same till the message arrives to the last agent in the list. This latter updates the value as others, then forwards it to a seed agent. Seeds are globally known and trusted peers in the system, their trustworthiness reaches $0.99 \in [-1, +1]$. Such type of seeds is already used in EigenTrust algorithm [12] and in Adovgato system[15]. The seed prepares the backward round, by selecting $x \in [-y, +y]$ and divides it into $n = |S_t|$ parts where $\sum_{i=1}^n x_i = x$ and distributing it to agents in $S_t$ as backward noise. Notice steps 2 and 3 in figure 3

In the backward round, each agent removes the local noise value he added in the forward round, and replaces it by the backward noise received from the seed. The last agent in the list forwards the final value to q as shown in step 4 of Figure 3. Total reputation value $tr = \sum_{i=1}^n lat_i + \sum_{i=1}^n x_i = \sum_{i=1}^n lat_i + x_i$

---

1Each agent maintains a set of agents that have interacted with it and have reported and assigned a feedback to it [2].
2.6. Splitting key Protocol [17]

In this protocol authors provide a decentralized privacy preserving reputation system, they are basing on splitting the local feedback value of each peer into n part, n is the number of participant agents, and distributing these parts to n-1 peer. As shown in figure.4 every single peer accumulates the values received from other peers and forwards the result to the querying agent “q”. q aggregates these values to reach the final reputation. We refer that all protocols mentioned above have linear complexity $O(n)$ whereas the splitting key protocol requires $O(n^2)$ messages which is considered costly. For example, given n=100, then we need 10,000 messages to calculate the total reputation value. This will abuse the network. Hence, the protocol is resilient against the collusion of up to n-1 agents, in other words. It is enough to have one honest agent (un-colluder) to keep the local feedback value of a peer vague to all other agents.
3 Problems and Challenges

We discuss two main problems which are not addressed in the protocols mentioned above neither in other privacy preserving reputation protocols.

The first one is the effect of Churn rate on the performance of PPRS\(^2\). Churn rate is defined by the number of agents entering and leaving the system in a time unit [20]. Indeed P2P systems have dynamic nature and peers enter and leave the network very often. Applying this scenario to protocol 2 in [7] (cf. section 2.5) leads the protocol to a critical position. Simply we can ask: who will remove, in the backward round, the local noise added by agent \(s_i\) if \(s_i\) is now absent? Restarting the protocol is not an efficient solution in such cases. Indeed it needs resending the same number of messages and we can’t guarantee that no another agent will fail during the next round time.

The second problem is The Trust Risk. Participants in protocols explained above are obliged to send their reputation values, differently perturbed, to agents from the list \(S_t\). We raise the following problem: what happens if an agent can’t find any trusted peers in the list \(?\). It will probably not provide an honest feedback value because it will be under attack with high probability. And it is not enough to count only on the trustworthiness of seeds.

4 Solutions

Here, we propose solutions for the problems mentioned in section 3 under two different types of attack.

**Attack type one:** each participant in the protocol exchanges intermediate messages with some other peers, those latter can collude in order to disclose the feedback value of that agent, using the values they have as shown in figure 2 above

**Attack type two:** observing the reputation value before and after an agent modifies it directly reveal the modification of the feedback given by that agent.

4.1. The Fault Tolerant Reputation Protocol

As discussed above, P2P environments have a dynamic status. High churn rate results from peer failures, e.g., because of moving out of coverage, discharged batteries, or from short session times based on the user behavior, e.g., due to high online costs.

We propose an algorithm which is an extension of the protocol 2 explained in section 2.5 designed to make it applicable in high churn rate networks.

\(^2\) PPRS = Privacy Preserving Reputation System
4.1.1. Protocol Overview
To facilitate understanding the protocol we provide in table 1 definitions for the symbols used in the algorithm.

Table 1. Symbols used to explain the algorithms in section 4.1.1 and 4.2.1

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>Querying agent</td>
</tr>
<tr>
<td>t</td>
<td>Target agent</td>
</tr>
<tr>
<td>p</td>
<td>Time of sending the message</td>
</tr>
<tr>
<td>St</td>
<td>List of resource agents</td>
</tr>
<tr>
<td>S</td>
<td>List of remaining agents</td>
</tr>
<tr>
<td>D</td>
<td>List of seeds</td>
</tr>
<tr>
<td>r</td>
<td>Reputation value</td>
</tr>
<tr>
<td>y</td>
<td>Value ∈ [-2, +2]</td>
</tr>
<tr>
<td>f_{ai}</td>
<td>Number of backups and trustees of a peer a_i</td>
</tr>
<tr>
<td>f'_{ai}</td>
<td>Number of backups and trustees who repaired the missing of a_i</td>
</tr>
<tr>
<td>AV</td>
<td>Availability flag: Identifies the status of a peer. AV = YES means peer is available. AV = NO peer is missed</td>
</tr>
</tbody>
</table>

The message circulating among agents contains information about (q, t, p, r, d, St, S, sheet); we explain the sheet few lines latter.

The querying peer will demand the list of resources and randomly chooses a seed d ∈ D and sends him the list St with r=0. d selects an agent from St to start the forward round and sends r = random[-y, +y].

Now each receiving agent S_i updates the reputation and passes the message to the most trusted peer exactly as explained in section 2.5.

Backup Noise processing: each peer keeps backup of all the noise he has at 2 trusted peers in the list St. When agent x sends y some noise we call x is trustee of y and y is backup of x.

The sheet is where each agent cites the number of his trustees and backups. It is represented as tuple (agent, f_{ai}, AV, f'_{ai}).

The first agent adds his local noise to the perturbing value it received from d and splits the totality into unequal two parts and sends these values to 2 trusted agents in the list, then modifies the sheet by adding (S_i, 3, YES, 0).

The backup noise of each peer is the accumulation of all backup noise he received from other peer and his own local noise:

\[ b_{n_{i}} = l_{n_{i}} + \sum_{j=1}^{b} b_{n_{j}}^{i} \]  \hspace{1cm} (2)

Last agent in the list sends both the reputation value and his backup noise to d.

Figure 5 illustrates exchanging messages during the forward round.

---

3 b is the number of trustees of a_i, b_n_i backup noise of peer a_i, j is a counter, b_{n_j}^i backup noise sent from a_j to a_i.
The backward round: as explained in 2.5 seed distributes perturbation noise values called “backward noise”. Each agent replaces his local forward noise by the backward noise. If any agent discovers the missing of a peer m, he sets the availability flag “AV” of m to “NO”.

When the message is caught by a backup peer of m, he subtracts the backup noise value he had received from m, and increases the value $f_m$ by one.

When the message is caught by a trustee of m, he doesn’t subtract the backup noise value that he gave to m, and also he increases $f_m$ by one. At the end of the round each missed agent must have $f_m = f_m'$ that means all his backup peers and trustees did their missions correctly. Notice the flow of backward messages in figure 6.

If $f_m \neq f_m'$ it means some backup peers or trustees were not informed by this missing (we consider that the missing was discovered after the message passed them). In this case d starts a second backward round to solve the problem.

Once all missed agents are fixed then the receiving agent will forward the total reputation value to q

If the second round is finished and $f_m \neq f_m'$ then the reputation value will be sent to q perturbed by a value equals $[-y,+y]*m$; $m$ = the number of non-fixed missed agents. Here, q decides either to accept it or to restart the protocol again

4.1.2. Formulation
We formalize the messages being exchanged among agents. Table 2 defies the symbols used in the formulas. To avoid repeating figures we refer some explanations to figure 3 shown above.
Table 2. symbols used in the protocols formulation.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Current agent</td>
</tr>
<tr>
<td>S(in)</td>
<td>List of remaind peers received by a</td>
</tr>
<tr>
<td>a(f, in)</td>
<td>Former agent in the forward round (agent before a)</td>
</tr>
<tr>
<td>a(f, out)</td>
<td>Successor agent in the forward round</td>
</tr>
<tr>
<td>Time()</td>
<td>Function returns current time</td>
</tr>
<tr>
<td>Random(x,y)</td>
<td>Function returns a random value between x and y</td>
</tr>
<tr>
<td>Random(L)</td>
<td>Function returns a random element from list L</td>
</tr>
<tr>
<td>y(a,p)</td>
<td>Local noise added by peer at time p</td>
</tr>
<tr>
<td>r(in)</td>
<td>Reputation value received from the former peer</td>
</tr>
<tr>
<td>r(out)</td>
<td>Updated reputation value to be sent to the next peer</td>
</tr>
<tr>
<td>a(b, in)</td>
<td>Former agent in the backward round (agent before a)</td>
</tr>
<tr>
<td>a(b, out)</td>
<td>Successor agent in the backward round</td>
</tr>
</tbody>
</table>

Initialization of the query by the querying peer:

1. q sends message(get_source_list) to t //as shown above in figure.3
2. t sends message(sources, St)
3. if |St|>2 //q checks the number of resource peers
4. then d = random (D) //q selects d
5. a(f, out) = d
6. p = Time() //take current time
7. S = St //initialize S to be equal to St at the start
8. r = 0
9. for all peers update their sheet as f = f’ = 0 and set default value for AV = YES
10. send message (seed, q,t,p,r,St,S,sheet) to d;

Message (seed, q,t,p,r,St,S,sheet) sent from q to d. d follows the steps 1 to 4

1. if a ∈ D and S = St // checking that the current peer is a seed
2. r = y //initialize r with a perturbation value
3. a(f, out) = random(St) //choosing the destination of the message
4. send message (forward, q,t,p,r,d,St,S,sheet) to a(f, out)

Message (forward, q,t,p,r,d,St,S,sheet)
This message carries the reputation value among all peers as shown in figure.7. Every peer “a: executes the steps 1 to 8

1. r(in) = r //read received value
2. S(in) = S
3. r(f, out) = r(in) + lat + y(a,p) //update the reputation value
4. S(out) = S(in) – a //removing current peer from the list S
5. if |S| ≠ 0 then
6. send message(forward, q,t,p,r(out),d,S,St,sheet) to a(f, out)
7. else a(f, out) = d //S is empty so send message to d
8. send (forward_result, q,t,p,r,S,St,sheet) to a(f, out) //sending the forward result to d
Fig. 7. The flow of the Forward Message.

Message (backup, q,t,p,r,d,S,St,sheet)
Each peer “a” executes the steps 1 to 9 to calculate his backup noise and send it to backup peers. The message backup carries the backup noise among peers as shown in figure.8
1. total_noise = \( l_n + \) backup noise received from former peers
2. \( r(out_1) = \) part of (total_noise) //calculate first part of backup noise
3. \( r(out_2) = \) total_noise – \( r(out_1) \) // calculate second part of backup noise
4. \( S(out) = S(in) – a \) //remove current peer from the list S
5. For i = 1 to 2 //choose 2 backup peers
6. \( a(f, out_i) = \) most-trusted(S(out))
7. \( S = S – a(out) \) //remove backup from S
8. \( f_a + 1 \) //increase the number of peer contacts
9. send message(backup, q,t,p,r(out),d,S,St,sheet) to a(f, out_i)

Fig. 8. Sending backup noise to backup peers.

Message (forward_result, q,t,p,r,d,S,St,sheet)
This message is sent from the last peer in the list S to the seed carrying the reputation value calculated in the forward round. Peer d follows the steps below
1. \( n = |St| \) //get the cardinality of the list St
2. \( S = St \) // prepare S for the backward round
3. for I = 1 to n //generate n backward noise values
4. Generate \( x_i \) where \( \sum_{i=1}^{n} x_i = [-y, +y] \)
5. for I = 1 to n //send backward noise to peers
6. send \( x_i \) to a_i
7. \( a(b, out) = \) random (S) //choose peer to start the backward round
8. \( r(out) = r \) //r is the reputation value calculated in the forward round
9. send (backward, q,t,p,r(out),d,S,St,sheet) to a(b, out)

Message (backward, q,t,p,r,d,S,St,sheet) received from a(b, in)
This message carries the reputation value among peers during the backward round as shown above in figure.3. step 4. Each receiving peer executes the following steps.
1. \( r(in) = r \) //read received value
2. \( S(in) = S \)
3. \( r(b, out) = r(in) - y(a,t,p) + x_a \)  
   //subtract the forward noise value and replace it by backward noise value

4. \( S(out) = S(in) - a \)  
   //remove current peer from list S

5. if \(|S| \neq 0\) then

6. \( a(b, out) = \text{most\_trusted}(S) \)  
   //choose the most trusted peer in S

7. if \((b, out)\) is missed

8. set availability flag of \((b, out)\) is missed = NO

9. \( S = S - a(b, out) \)  
   //remove the missed peer from the list

10. return to step 6.

11. else send message(backward, q,t,p,r(out),d,S,St,sheet) to \((b, out)\)

12. else \(a(b, out) = d\)  
   //round is finished so pass the message to the seed again

13. send (result, q,t,p,r,d,S,St,sheet) to d

Message (result, q,t,p,r,d,S,St,sheet) received from \((b, in)\)
This message carries the final reputation value to q. Each peer can send the result to q when S is empty by executing the following steps.
1. for all missed agents if \(f_m = f_m'\)  
   //check if all missed peers are fixed.
2. then rt = r  
   //reputation protocol is done
3. send the value to q

4. else if \(f_m \neq f_m'\) and that was the first backward round then:
5. \( S = St - \text{missed agents} \)  
   //rebuild S from St without missed peers
6. \( a(b, out) = d \)
7. send message repair_missing(q,t,p,r,d,S,St,sheet) to \((b, out)\)
8. else send result to q with referring to the number of missed agents

Message (repair_missing, q,t,p,r,d,S,St,sheet)
This message passes among peers during the second backward round. Figure.11 in the section 4.1.4 shows the flow of it. Each peer involved in the second backward round follows the steps 1 to 12.

1. \( m \): is the missed agent
2. for all the missed agents do the following:
3. if \(a\) is backup of \(m\)  
   //if current peer is backup of \(m\)
4. \( r(out) = r - bn_{ma}^4 \)  
   //subtract the backup noise of \(m\)
5. \( f_m'++ \)  
   //increase the number of peers who fixed the missing of peer \(m\)

6. else if \(a\) is trustee of a missed agent  
   //if current peer is a trustee of \(m\)
7. \( r(out) = r + bn_{ma} \)  
   //add the backup noise of the missed peer
8. \( f_m'++ \)  
   //same as step 5
9. if \(|S| \neq 0\) and \(f_m \neq f_m'\)  
   //there are more missed peers to be fixed
   and list S is not empty
10. \( a(b, out) = \text{random}(S) \)  
   //choose a peer from S
11. send (repair_missing, q,t,p,r,d,S,St,sheet) to \((b, out)\)
12. else send message(result, q,t,p,r,d,S,St,) to d

---

\(^4\) \(bn_{ma}\): backup noise sent from peer \(m\) to peer \(a\) during the forward round.
4.1.3. Correctness

Theorem 1. If all peers follow the steps of the algorithm correctly, then, even if some peers were missed, the total reputation value will be

\[ rt = \sum_{i=1}^{n} lat_i + \sum_{i=1}^{n} x_i \]

Proof: during the forward round peers add their local feedback values and noise, at the end of this round the seed will receive an intermediate reputation value

\[ r = \sum_{i=1}^{n} lat_i + \sum_{i=1}^{n} ln_i \]

During the backward round each agent receives \( r'_{i-1} \) and sends \( r'_i = r'_{i-1} - ln_i + x_i \)

At the completion of the first, or the second, backward round the total reputation value will be as follows:

\[ rt = \sum_{i=1}^{n} lat_i + \sum_{i=1}^{n} x_i \]

The following example explains the correctness in case of missing 2 peers among 7.

4.1.4. Example

Given an agent \( q \) inquiring about the reputation of peer \( t \), the set of resources \( St = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\} \) is the chosen seed to participate in the protocol, \( y \in [-2, +2] \) is the perturbation value sent from \( d \) to \( s_1 \) we notice in figure.9 the flow of messages in the forward round

\[ r_i = y + lat_i + ln_i \]  

(3)

\[ bn_i = bn_{13} + bn_{14} = y + ln_i \]  

(4)

\begin{figure}
\centering
\includegraphics[width=\textwidth]{forward_round.png}
\caption{Exchanging intermediate feedback and noise values in the forward round}
\end{figure}

\( S_1 \) sets the variable \( f_i = 4 \) to determine the total number of his backups and trustees. 

Table.3 shows the number of backups and trustees of each peer.

We notice that \( S_3 \) calculates its backup noise in considering the part he received from \( S_1; bn_3 = ln_3 + bn_{13} \) at the end of the forward round \( d_2 \) receives the value \( r = \sum_{i=1}^{7} r_i \)

\begin{table}
\centering
\caption{Each peer and the total number of its backups and trustees.}
\begin{tabular}{|l|c|}
\hline
Agent & \( f \) \\
\hline
S_1 & 2 \\
S_2 & 2 \\
\hline
\end{tabular}
\end{table}
In the backward round: it is time for each agent to remove his local noise and replace it by the value he receives from the seed.

Missing agents in the backward round: as illustrated in figure.10 $S_3$ notices the missing of $S_4$ and modifies AV of $S_4$ to NO. Agents who are involved: $S_1$, $S_6$ and $S_5$ will notice the flag AV. $S_1$ and $S_6$ increase the value of $f'_i$ in the first backward round, whereas $S_5$ needs another round to do his mission. $S_1$ discovers the missing of $S_2$ and raise a flag too. The involved agents are $S_3$ and $S_7$, both already read the message before, so, they need the second round to do their missions. $S_2$ receives the message, since he is the last agent in the round he notices that the message is still incomplete. So, $S_6$ starts another round by choosing randomly an available agent and circulate the message again. In the second backward round $S_4$ and $S_7$ correct the reputation value and increase $f'_2$.

![Fig. 10. Replacing forward noise by the backward noise, detecting missed agents](image)

**Table 4.** in the first backward round all backups and trustees of $S_4$ increase $f'_i$, backups and trustees of $S_2$ have read the message before discovering the missing of $S_2$

<table>
<thead>
<tr>
<th>Missed agent</th>
<th>$f$</th>
<th>$f'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_4$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$S_2$</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Second backward round: as mentioned above, this round is to allow backups and trustees to do their missions if they didn’t do that during the former round. We suppose that $S_3$ is down during the second backward round. We underline that missing won’t cause any problem since $S_3$ is not involved in the correction process now. When the second round is finished $f'_i = f'_i$ for all missed agents and the reputation value is correct, as shown in figure.11.
Fig. 11. Removing the backup noise by S\textsubscript{5} and S\textsubscript{7}

Table 5. Number of backups and trustees who signed the message in the 2\textsuperscript{nd} backward round

<table>
<thead>
<tr>
<th>Missed agent</th>
<th>( f )</th>
<th>( f' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>S\textsubscript{4}</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>S\textsubscript{2}</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S\textsubscript{1}</td>
<td>Don’t care</td>
<td></td>
</tr>
</tbody>
</table>

4.1.5. Privacy Preserving

Resilience against attack one: In normal cases the agent communicates with more than 8 agents, 6 of them need to collude to reveal the local reputation value. In worst trust case, when no agents trust the peer a, this latter communicates with 7 agents 4 of them need to collude to reveal the local reputation value.

Since each single value is distributed to many peers, we discuss what each agent can learn to derive the number of colluders needed to reveal the value:

\( a(f, \text{in}) \) only knows the value he sends to a, we call it \( v_1 \).

\( a(f, \text{out}) \) receives:

\[
r_i = r_{i+1} + \text{lat}_i + \text{ln}_i = v_2
\]

(5)

The backup noise value of each agent is calculated as:

\[
\text{bn}_i = \text{ln}_i + \sum_{j=1}^{b} \text{bn}_{ji}
\]

(6)

The first backup peer knows the value \( v_j \):

\[
v_j = \text{random part of } \text{bn}_i
\]

(7)

The second backup peer knows the value \( v_d \):

\[
v_d = \text{bn}_i - v_3
\]

(8)

\[
\text{lat}_i = v_2 - v_1 - v_3 - v_d - \sum_{j=1}^{b} \text{bn}_{ji}
\]

(9)

As noticed, to derive the local value of an agent, all his backups and trustees have to collude, and in the case of the first agent of the list, then the seed “d” should collude since there are no trustees.
Resilience against attack two: the local reputation value is never known by any single agent, in the forward round it is perturbed by the local noise of the agent itself

\[ r_i = r_{i-1} + \text{lat}_i + \text{ln}_i \Rightarrow r_i - r_{i-1} = \text{lat}_i + \text{ln}_i \] (10)

In the backward round the feedback value is perturbed by the noise received from the seed

\[ r'_i = r'_{i-1} - \text{ln}_i + x_i \Rightarrow r'_i - r'_{i-1} = \text{ln}_i + x_i \] (11)

Thus, observing the reputation value before and after its modification will never give the true value of last feedback added.

4.1.6. Complexity

We calculate the number of messages being exchanged among peers during the protocol. In best case, where the missing of peers can be solved during the first backward round and in the worst case, where a second complete backward round is needed:

<table>
<thead>
<tr>
<th>Table 6. source, destination and number of different messages exchanged between peers</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
</tr>
<tr>
<td>q</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>q</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>Si</td>
</tr>
<tr>
<td>Each Si except S_n</td>
</tr>
<tr>
<td>Backward round</td>
</tr>
<tr>
<td>Total number of messages in best case</td>
</tr>
<tr>
<td>Total number of messages in worst case</td>
</tr>
</tbody>
</table>

4.2. Blind Calculation Protocol

The goal of this protocol is to minimize the time of calculation and to protect the peers against the probability of the seed’s collusion by using one single round.

To achieve this goal, we suppose dividing the list of recourses and let the calculations happen in parallel and we allow integrating agents out of the resource list to participate in the calculation process. This will raise the opportunity of a peer to find trusted agents to share his value with. We assume that each peer in the resource list has a high-trusted peer, called friend, who doesn’t belong to the list St and who is 100% trustworthy and doesn’t collude against his friend. This assumption is realistic in most contexts.

Employing friend peers in the protocol will rise the privacy degree of the protocol and encourage agents to give honest reputation values.

\[ ^5 m \text{ represents the number of missed agents} \]
4.2.1. Protocol Overview

In this algorithm, q chooses 2 random seeds $d_1$ and $d_2$. He sends the list $St$ and the address of $d_1$ and an ID number to $d_1$. ID is a unique number chosen to identify all the messages related to the same query in order to be distinguished from other messages that can be exchanged among peers.

$d_1$ encrypts a global noise, $x_i$, for each agent in $St$, using their public keys, and integrates a list $GN$ of these encrypted values, into the message, where $|GN| = |St|$ and $\Sigma_{i=1}^{n} x_i = X \in [-y, +y]$.

$d_1$ also divides the recourses list into two sub-lists $St_1$ and $St_2$ and randomly chooses an agent from each sub-list. $d_1$ sends to these agents the initial message where $r = 0$.

Each receiving agent $a_i$ will decrypt his proper global noise $x_i$ using its private key and add it to its local reputation value $lat_i$. It then divides this value into two parts, then selects the most trusted agent from the list $S$ and a high-trusted peer out of it, to send each of them a part of the reputation value $r_i$.

$$r_i = r_{i-1} + lat_i + x_i$$

(12)

The last agent in each sub-list and every friend agent will forward the value he has to $d_2$; this latter gathers all values and calculates the total reputation $rt$. Before $d_2$ sends the reputation to $q$, $d_1$ will send him $GN$ value to subtract it from $rt$, this way we guarantee that $rt$ is pure reputation value and doesn’t contain noise as shown in fig.12.

An important assumption in this protocol is that we use an anonymous routing protocol between each agent and its friend, and also between friends and the seed $d_2$.

---

Fig. 12. Calculating Reputation value of a peer $t$ using Blind Calculation protocol.
4.2.2. **Anonymous Routing**

In the absence of anonymising technology, every computer connected to the Internet communicates with a unique address. As a result, an eavesdropper can derive who are the peers contacting each other. Anonymous communication technologies are designed to anonymise communication partners in a computer network even if the underlying network infrastructure does not support anonymous participation. One option is to avoid direct message transfers from senders to recipients.

The core of Anonymous routing techniques is to repeatedly decrypt and forward a message through some network nodes, where each node only knows where from it gets a message and where to forward it. As shown in figure 13. This guarantees the fact of hiding the identity of message sender from all peers in the system, even the destination can’t tell who is the message source, unless, this latter mentions anything about its identity inside the message.

Employing such routing for the calculating protocol prevents malicious agents from eavesdropping and traffic analyzing; that means no one can learn who is talking to whom. Some anonymous protocols are proposed like FreeNet [5], Onion Routing [21] and ARHR [3], they all have in common the main concept of transforming information between hosts without revealing the id of the sender or the receiver. This ensures that even eavesdroppers who observe all connections cannot infer any relation between data packets by evaluating content or timing information. Some anonymous routing techniques, like Onion routing, are bidirectional and have near real-time connections [21].

![Fig. 13. Passing message through anonymous routing.](image)

4.2.3. **Formulation**

To reach best understanding for the protocol we use the same formulation and terms used in section 4.1.2

1. q sends message(get_source_list) to t //request for resources
2. t sends message(sources, St) to q //request answer
3. if |St|>2
4. then d1 = random (D) //choose d1
5. a(out) = d1 //set d1 as the destination of the message
6. d2 = random (D) //choose d2
7. p = time() //get the current time
8. S = St //initiate S
9. ID = random_number //choose a unique number for all the messages belong to the same query
10. \( r = 0 \)  //initiate \( r \)
11. send message (seed, q,t,p,r,d,S,t,S,ID) to \( d_1 \)

Message (seed, q,t,p,r,d,S,t,S,ID) sent from q to \( d_1 \)
This message carries the initiative information from q to \( d_1 \) and \( d_1 \) executes the steps
1. \( n = |S_t| \)
2. \( tgn = 0 \)  //total global noise
3. \( gn\_list = \text{empty} \)  //initialize the global noise list
4. for \( I = 1 \) to \( n \) select \( gn_i = \text{random\_number} (-y, +y) \)  //generate global noise
5. \( tgn = tgn + gn_i \)  //calculate the totality of global noise
6. encrypt (\( gn_i \), \( pk \))  //encrypt \( gn \) using the private key of each peer
7. add \( gn_i \) to \( gn\_list \)  //fill the global noise list
8. \( i++ \)  //go to next peer
9. \( c = n \mod 2 \)  //counter to divide the list into 2 parts
10. for \( I = 1 \) to \( c \) move peers from \( S_t \) to \( S_{t1} \)  //build first sub-list
11. \( S_{t1} = S_t \)  //\( S_{t1} \) contains \( S_t/S_t \)
12. \( a(out) = \text{random}(S_{t1}) \)  //select peer from \( S_{t1} \)
13. send message(forward, q,t,p,r,d,S,\( gn\_list,ID \)) to \( a(out) \)
14. send message (global_noise, q,t,p,\( gn \)) to \( d_2 \)
15. \( a(out) = \text{random}(S_{t2}) \)  //select peer from \( S_{t2} \)
16. send message(forward, q,t,p,r,d,S,\( gn\_list,ID \)) to \( a(out) \)

Message (global-noise, q,t,p,\( gn \)) sent from \( d_1 \) to \( d_2 \)
1. \( gn = tgn \)  //\( d_2 \) reads the global noise value

Message (forward, q,t,p,r,d,S,\( gn\_list,ID \))
This message contains a part of the reputation value calculated at each peer as shown below in figure.14. Each receiving peer in \( S_t \) follows the steps.
1. \( r(in) = r \)  //read received reputation value
2. \( S(in) = S \)  //read list \( S \)
3. \( y(a,t,p) = \text{decrypt}(gn, \text{private\_key}) \)  //decrypt noise
4. \( gn\_list = gn\_list - y(a,t,p) \)  //remove the noise from \( gn\_list \)
5. \( r(out) = r(in) + lat + y(a,t,p) \)  //update the reputation value
6. \( r(out) = \text{part of} r(out) \)  //split the reputation value into 2 parts
7. \( r(out) = r(out) - r(out) \)
8. \( S(out) = S(in) - a \)  //remove current peer from \( S \)
9. \( a(out) = \text{most-trusted}(S_t) \)  //select the most trusted peer in the list
10. send message(forward, q,t,p,r,\( out(out)\),d,S,\( gn\_list,ID \)) to \( a(out) \)
11. \( a(out) = \text{friend}(\text{all\_agents}) \)  //select a friend from the whole system
12. send message(forward3, q,t,p,r,d,S,\( gn\_list,ID \))

Message (forward3, q,t,p,r,d,S,\( gn\_list,ID \))
This message is sent from each friend peer to \( d_2 \). It carries the second part of the reputation value as shown in figure.14. Each friend peer executes steps 1 to 4
1. \( r(in) = r \)  //read received value
2. \( r(out) = r(in) \)
3. \( a(out) = d_2 \)  //\( d_2 \) destination of the message
4. send message(forward3, q,t,p,r,d,S,\( gn\_list,ID \)) to \( a(out) \)
Message (result, q,t,p,r) sent from d₂ to q
The seed d₂ execute the following steps to calculate the total reputation value and send it to d
1. check if ID is correct //ensure that the message belongs to the same protocol
2. r(in) = r //read received value
3. r(out) = r(out) + r(in) //accumulate values to calculate rt
4. messages ++ //count the messages received by d₂
5. if messages = 2*|St| + 2 //all messages are received
6. r(out) = r(out) - gln //subtract the global noise value.
7. send message(result, r(out)) to q //send the pure reputation value to q
8. else reject the message //reject the message contains wrong ID

Fig. 14. Flowing messages in Blind calculation protocol.

4.2.4. Correctness
Theorem 2. If all agents follow properly the protocol steps, then at the completion of the query the total reputation value \( rt = \sum_{i=1}^{n} lat_i \)

Proof: each agent receives the message (forward, q,t,p,r,d₂,St,S,gn_list) and calculates \( r(out) \)

\[
r_i(out) = r_i(in) + y_i(a,p) \tag{13}
\]

The seed d₂ will receive messages from agents inside and outside St, the accumulated values received by d₂ is \( r \):

\[
r = \sum_{i=1}^{n} y_i(a,p) + \sum_{i=1}^{n} lat_i \tag{14}
\]

d₂ will receive also message(global_noise,gn) that contains (tgn) total global noise added by the agents

\[
GN = \sum_{i=1}^{n} y_i(a,p) \tag{15}
\]

by subtracting GN from the sum calculated in (14) the reputation value will be:

\[
rt = r - GN = \sum_{i=1}^{n} lat_i \tag{16}
\]
4.2.5. Privacy Preserving

Resilience against Attack type one: Each agent in St exchanges messages with 3 agents directly and receives indirect value from the seed d1. All the 4 agents must collude to reveal the local reputation value of an agent, which is highly improbable since 2 of these agents are chosen by the agent himself, one is a friend and the other is the most trusted agent of St and the third agent is the seed which is highly trusted by the global system.

Resilience against Attack type two: the total reputation value calculated by each agent can’t be learned by only one agent, since it is perturbed by the global noise received from d1 and it is separated into 2 parts. This way we ensure that observing the value from malicious peers in the list will never lead to derive the local feedback of a single peer.

Now, we calculate the probability to reveal the local feedback value of a trivial peer. We consider that W is set of agent who desire to discover the local feedbacks of other agents. We refer to the colluding probability of peer a by P(a ∈ W).

The probability to break the privacy of an agent = P(a(in) ∈ W) * P(a(out1) ∈ W)* P(a(out2) ∈ W)*P(d1 ∈ W).

Since agent a can’t control who is a(in) so we assume that P(a(in) ∈ W) = 1. Whereas, a(out2) is a friend agent, thus, P(a(out2) ∈ W) = 0. P(out1 ∈ W) is also very low probability since he is most trusted peer. Last agent who is needed to complete the colluding circle is d1, what encouraged us to use seeds in our protocols is the fact that seeds are already used in other reputation systems like [6, 7] and showed good behaving with probability to collude less than 0.01 [-1, +1].

From all that mentioned we find out that the privacy breaking of an agent = 1* P(out1 ∈ W) * 0 * 0.01 = 0

4.2.6. Complexity

This protocol has linear complexity $O(n)$. We describe the messages circulating during the protocol in the following table.

Table 7. Source and destination of different types of messages exchanged in Blind Calculation protocol.

<table>
<thead>
<tr>
<th>Form</th>
<th>To</th>
<th>Number of messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>t</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>q</td>
<td>1</td>
</tr>
<tr>
<td>q</td>
<td>d1</td>
<td>1</td>
</tr>
<tr>
<td>d1</td>
<td>Agents in St1, St2</td>
<td>2</td>
</tr>
<tr>
<td>n agents</td>
<td>Successor, high-trusted</td>
<td>2 * n</td>
</tr>
<tr>
<td>d1</td>
<td>d2</td>
<td>1</td>
</tr>
<tr>
<td>d2</td>
<td>q</td>
<td>1</td>
</tr>
<tr>
<td>Total number of messages</td>
<td>2 * n + 7</td>
<td></td>
</tr>
</tbody>
</table>
5 Analysis Framework

In this section we provide a framework by which the PPRSs can be compared. We then classify and explain the fundamental concepts that affect the privacy in reputation systems.

We identify the following dimensions to be the base of classification and analysis

5.1. Calculation Structure

Privacy preserving reputation systems have two different types of structures used to calculate the global reputation of a participant.

Centralized algorithms \([1, 2]\) are based on accumulating the different feedbacks that are given by peers at one central authority. This authority is used for long-term storage of all reputation data. Such algorithms suffer from critical situations: the first one is the missing of the central authority, or being under Denial of Service attack. When this breakdown happens, even for a short period of time, it will paralyze the total system. No more feedbacks can be sent to the CA, and no more reputation requests can be answered.

A second risk that can be considered is the trustworthiness of the CA. Since the CA is responsible about the whole values in the system, any blunder in choosing it, can drop the privacy of peers as the CA can easily observe the feedback values and learn their providers.

In modern decentralized structures as P2P reputation systems\([7, 17]\), the load is allocated at many participants, and the reputation is globally calculated in decentralized manner. Feedback values are locally stored at the peers, who collaborate and participate in a protocol to calculate the global reputation of an agent in a fully distributed manner. Despite the complexity of the calculation algorithms, such type of structures protects the system from having one single point of failure, and prevents malicious agents from subverting the feedback values easily. They also scale well and achieve the goal, with different limitations, of preventing malicious participants from deriving the local values of honest agents

5.2. Privacy preserving, attack defence

A privacy preserving reputation system is considered strong and robust as much as it is hard for malicious agents to derive the local feedback of a peer.

As long as it is difficult to learn the real opinion of agents, the participants will vote honestly, no matter what is the method to achieve this goal. Some systems employ mechanisms to protect the feedback value from being revealed, others choose to protect the feedback provider from being known, in other words, realize unlink-ability between a feedback value and its provider.

5.2.1 Protecting feedback values: using vague feedback values instead of pure ones proved good results in defending against malicious peers who attempt to reveal local feedback values of victims in peer to peer environments.
Adding noise: one manner to insure the privacy of a feedback value is to add perturbation noise in order to confuse other participants who are calculating the global reputation value of the target agent in additive manner.

Splitting the value: Another mechanism used to calculate the reputation value respecting the decentralization, is based on splitting the local feedback on an agent into many parts and sending these portions to some participants. This achieves protecting the total value of each agent from being learnt by other peers unless these latter collude with all peers who have the complementary parts, however this requires more messages to be exchanged between agents and varies the complexity of the calculation from $O(n)$ as our protocols explained in section 4 to $O(n^2)$ in [17].

5.2.2 Protecting Feedback Provider: anonymity is also desirable attribute for agents participating in P2P reputation systems, under the conditions of realizing unlinkability (i.e. hard to tell this value is provided by whom). These peers will be encouraged to give honest feedbacks.

Using pseudonyms: agents contact each other by their nicknames, or pseudonyms, where there is no link between these pseudonyms and the real identities of feedback providers [1].

Using anonymous routing: instead of masquerading the participant’s identity, anonymous routing techniques can be used to prevent eavesdroppers and curious peers from traffic analyzing, by passing the messages encrypted through some routers, each of these routers can only determine its two neighbours, the one it received the message from, and the successor, as explained above in section 4.2.2 this manner can be used to masquerade the sender (message source) from being known by the destination, and to hide the connection itself from other participants in the system.

5.2.3 Privacy Degree: we estimate the privacy degree of a reputation system by the probability to reveal the local value provided by a participant, in other words, the more it is hard to derive the feedback values, the stronger the system is, however, in this framework, we just consider the colluding as an issue to break the privacy of a system. The collusion itself depends on 2 factors: the trustworthiness of peers involved in a calculation process, the number of these peers.

Number of colluders: we are certain that dividing and distributing the local feedback value of an agent to many peers leads malicious agents for more efforts to discover the value of their victim, in the Splitting Key protocol the value is divided into n part and distributed to n-1 agents other than its provider, all these agents need to collude to reveal the local value of an agent. The presence of only one honest agents will prevent all others from breaking the privacy of a party (the victim), in protocol 2 in [7] explained in section 2.5 the feedback value is not broken, but, it is perturbed and exchanged with five agents, all the five need to conspire in order to satisfy their curiosity and know this value, whereas in [1, 2] a single feedback value is neither disrupted nor split, but it is transformed as a pure value from one agent to another.

Trustworthiness of value keepers (trivial peers, colluders, seeds and friends): even if the individual feedback value is broken into some parts, it still can be under attack, thus, the trustworthiness of peers also forms an essential edge of the protocol’s
privacy, for example, some reputation systems [7, 12, 15] consider the employing seeds. Also the action of choosing the most trusted agents and employing friends; 100% trusted agents. All these factors make it harder for malicious adversaries to derive a local feedback value of a peer. What can cause a problem is the leak of such types of trusted participants in the system. However, in P2P communities these agents are available very often.

5.3. Connection Status

The security of the transport channel, where the data is carried on, is as important as the security of the calculation algorithm, since eavesdroppers and malicious agents can listen to the connections, learn who is contacting whom, and reveal the message’s information.

5.3.1 Secure Socket Layer (SSL): it is a cryptographic protocol used to keep data secured during the communication throw the internet, where the sender and receiver establish a secure connection and the sender encrypt the data using the public key of the receiver[23], only the receiver can decrypt the data by its private key, unless this latter is compromised, privacy preserving reputation protocols [7, 17] assume using such mechanisms, we also have the same assumption for the fault tolerant protocol.

5.3.2 Anonymous Routing: employing SSL prevents curious peers from learning the contents of a message, but, an eavesdropper is still able to monitor the exchange of messages, this action is called traffic analyze, and know who is talking to whom. When traffic analyze is possible a bunch of colluders can exchange some external messages and derive the local feedback of a peer using the intermediate values they have. Anonymous routing, as explained in section 4.2.2 prevents colluding eavesdroppers from achieving their goals.

5.4. Analysis

In tables 9 we compare the privacy preserving reputation protocols that exist referring to the framework we discussed above. Table.8 contains the symbols used in this comparison.

Table 8. definition of symbols used in table.8.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Decentralized</td>
</tr>
<tr>
<td>C</td>
<td>Centralized</td>
</tr>
<tr>
<td>Nis</td>
<td>Feedback is perturbed by noise value</td>
</tr>
<tr>
<td>Spl</td>
<td>Feedback is split between some agents</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure socket layer</td>
</tr>
<tr>
<td>AR</td>
<td>Anonymous Routing</td>
</tr>
<tr>
<td>Protocol</td>
<td>System structure</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure sum[17]</td>
<td>D</td>
</tr>
<tr>
<td>Key splitting[17]</td>
<td>D</td>
</tr>
<tr>
<td>Secure sum[7]</td>
<td>D</td>
</tr>
<tr>
<td>Protocol 2[7]</td>
<td>D</td>
</tr>
<tr>
<td>Fault tolerant</td>
<td>D</td>
</tr>
<tr>
<td>Blind calculation</td>
<td>D</td>
</tr>
<tr>
<td>Privacy and incentive[2]</td>
<td>C</td>
</tr>
<tr>
<td>R.S for anonymous-networks[1]</td>
<td>C</td>
</tr>
</tbody>
</table>

In Table 9, we compare different PPRPs.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>System structure</th>
<th>Attack defense</th>
<th>Protection feedback</th>
<th>Privacy degree</th>
<th>Used connection</th>
<th>N° of messages</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure sum[17]</td>
<td>D</td>
<td>-</td>
<td>R id</td>
<td>2</td>
<td>SSL</td>
<td>n+2</td>
<td>O(n)</td>
</tr>
<tr>
<td>Key splitting[17]</td>
<td>D</td>
<td>Spl</td>
<td>R id</td>
<td>n-1</td>
<td>SSL</td>
<td>n²</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Secure sum[7]</td>
<td>D</td>
<td>Nis</td>
<td>R id</td>
<td>2</td>
<td>SSL</td>
<td>n+2</td>
<td>O(n)</td>
</tr>
<tr>
<td>Protocol 2[7]</td>
<td>D</td>
<td>Nis/spl</td>
<td>R id</td>
<td>5</td>
<td>SSL</td>
<td>3n+4</td>
<td>O(n)</td>
</tr>
<tr>
<td>Fault tolerant</td>
<td>D</td>
<td>Nis/spl</td>
<td>R id</td>
<td>4</td>
<td>SSL</td>
<td>[4n+3-m, 5n+3-m]</td>
<td>O(n)</td>
</tr>
<tr>
<td>Blind calculation</td>
<td>D</td>
<td>Nis/spl</td>
<td>R id</td>
<td>[4, n]</td>
<td>100 % AR</td>
<td>2n+7</td>
<td>O(n)</td>
</tr>
<tr>
<td>Privacy and incentive[2]</td>
<td>C</td>
<td>_</td>
<td>Pseudo</td>
<td>3</td>
<td>CS</td>
<td>n</td>
<td>O(n)</td>
</tr>
<tr>
<td>R.S for anonymous-networks[1]</td>
<td>C</td>
<td>_</td>
<td>Pseudo</td>
<td>2</td>
<td>CS</td>
<td>n</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

In Table 10, we compare the number of messages needed in both Fault Tolerant protocol and Protocol 2 in [7] in the case of successive missing of same number of peers.

<table>
<thead>
<tr>
<th>N° peers</th>
<th>Missed peers</th>
<th>Protocol 2</th>
<th>Fault tolerant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3n+4</td>
<td>[4n+3-m, 5n+3-2m]</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>3(50)+3(49)+8=305</td>
<td>[202, 252]</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>1529</td>
<td>[193, 243]</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>5510</td>
<td>[383, 483]</td>
</tr>
<tr>
<td>200</td>
<td>50</td>
<td>26475</td>
<td>[783, 883]</td>
</tr>
</tbody>
</table>

6 R.S means Reputation System
5.5. Analysis Observations

From the tables above we notice that:

- Protocols with the same structure have other shared characteristics. Systems with decentralized structure always use the real id of a feedback provider, whereas centralized ones use pseudonyms.
- Most of decentralized protocols masquerade the local value of each peer, where this concept is absent in centralized ones.
- Splitting key protocol requires the largest number of colluders to reveal the value of an attached peer.
- Blind calculation protocol achieves the highest degree of privacy.
- Splitting key protocol needs the highest number of messages.

Restarting the protocol 2 [7] entails sending very large number of messages, compared to the number of messages used in our Fault Tolerant protocol.

6 Conclusion and Future Work

In this paper we have proposed two privacy preserving reputation protocol. The first protocol is designed to be applicable in dynamic networks where peers enter and leave the system very often and to avoid as much as possible the “Restart Protocol” solution in case of missing agents. We were able to use the advantages of unstructured P2P networks even at highly dynamic situation in acceptable costs. We presented a second protocol that achieves 100% privacy. Based on a decentralized calculation it uses friend peers and anonymous routing techniques. This protocol assumes that a peer has at least one fully trustworthy “friend” peer in the system as whole.

Both protocols are fully decentralized and don’t suffer from any single point of failure; in the same time, they are resilient against the curious and semi-disruptive peers while respecting a linear complexity.

Our future work is to address the open challenges related to the attacks of PPRSs caused by some disruptive adversaries: (1) Adding of feedback values out of the afforded range, (2) Preventing the protocol to be complete by dropping the intermediate messages.

7 References


