

# Fair and Scalable Peer-to-Peer Games of Turns

Adam Wierzbicki\*

*adamw@pjwstk.edu.pl*

*\*Polish-Japanese Institute of Information Technology  
ul. Koszykowa 86, 02-008 Warsaw, Poland*

Tomasz Kucharski\*

## Abstract

*The article considers trust management in P2P games of turns without trusted, centralized resources and without using reputation. We improve a fair protocol for random drawing developed for P2P Scrabble [23]. We also modify the public state management to use Byzantine agreements. The complexity of the proposed mechanisms is analyzed and a new mechanism for faster reshuffling and drawing during the game stage is proposed.*

## 1. Introduction

Using the P2P model for the design of computer games would allow to avoid bottlenecks and single points of failure that occur in client-server applications. P2P games could make better use of computational resources at the edge of the network. P2P games could also be designed to have smaller reaction times than client-server games.

However, the development of P2P games faces several significant obstacles, such as resistance to node departures (failures), data replication, secure and scalable communications. Trust management is another significant obstacle to P2P game design. In a client-server architecture, centralized management of the game state allows simple enforcement of the game rules. In a game without trusted, centralized resources, how can competing parties ensure fairness on their own?

We have tried to answer the question of avoiding cheating – player actions that violate game rules – in P2P games of turns. There can be many examples of games that fit the chosen general model: bridge, poker, roulette, black-jack, dice etc. For a case study, Scrabble – the favorite game of one of the authors – seemed a natural choice. In [23] we have described a design of P2P Scrabble, and discussed the relevance of our results to the original question of general P2P game design. In that article, we have presented two protocols for random drawing in P2P games: drawing from an infinite set of objects (useful, for example, for a dice throw) and from a finite set of objects (used in P2P Scrabble). This article

extends the original design presented in [23] by a scalable method of object reshuffling, and by an improved design of public state. The most significant improvement is based on an analysis of trust management methods used in real life. Consider the enforcement of trust by police in road traffic. Our approach also proposed a method of self-enforcement of fairness in a P2P game. However, in road traffic, police is not present all the time. The occasional presence of police is sufficient to deter violations of traffic laws. We therefore propose a relaxation of our mechanisms by random checking of move correctness in a P2P game, that reduces overhead while still deterring players from cheating. We have also analyzed the complexity of the proposed algorithms.

## 2. Trust management using reputation

One of the most common forms of trust management is the use of agent reputations. Among many applications of this approach, the most prominent are on-line auctions (Allegro, E-Bay). However, P2P file sharing networks such as Kazaa, Mojo Nation, Freenet | Freedom Network [6,7,8] also use reputation. Reputation systems have been widely researched in the context of multi-agent programming, social networks, and evolutionary games [17, 2,3,4,13, 14, 18].

Reputation-based mechanisms could be used in P2P games. A player would receive a reputation based on a history of previous games, and this reputation could be used to exclude cheating players from a game. The reason for the use of reputation mechanisms in many networked applications is the lack of enforcement mechanisms that could be used to provide trust management, as noted in [17].

In our research, we have chosen not to use reputation mechanisms due to their systematic drawbacks, such as first-time cheating. Instead, we have developed cryptographic protocols for enforcement of fairness.

## 3. Authentication requirements for P2P games

A P2P game could use many different forms of authentication. At present, most P2P applications use weak authentication based on nick names and IP addresses (or IDs that are derived from such information). However, it has been shown that such systems are vulnerable to the Sybil attack [19].

Most of the mechanisms discussed in this paper would not work if the system would be compromised using the Sybil attack. An attacker that can control an arbitrary number of clones under different IDs could use these clones to cheat in a P2P game. The only way to prevent the Sybil attack is to use a strong form of authentication, such as based on public-key cryptography. For that reason, in our subsequent discussion we will assume that such an authentication is available and that any peer can authenticate any other peer in the P2P game.

## 4. Design of P2P Scrabble

### Scrabble rules

All users of P2P Scrabble shall be called players. We shall refer to a Scrabble player who makes a move in the game as the drawing player. All other players that at this time point play the same game as the player shall be referred to as the competitors. All players that play a game together shall be referred to as the game group or the game players.

Scrabble is a game with turns, played usually by 3 or 4 players. Each player has a secret pool of letters that he tries to use to create words on the board. The players are awarded points for the words they put on the board, depending on their location on the board and the type of letters used. The correctness of words on the board is verified using dictionaries. The letters are drawn from a letter sack. At the end of each turn, a player must draw new letters so that he will have a fixed number of letters (7). We shall not go into further detail of Scrabble rules, referring the reader to the game documentation. However, some additional features of Scrabble will be explained further in the text.

### Can we trust disinterested players?

The set of all players in P2P Scrabble is the set of all players currently playing all games. From the point of view of a game group, all other players are called *disinterested players*. Many problems with the design of P2P Scrabble could be solved if the game players could trust disinterested players.

However, this may not be as simple as it seems. There are two reasons why disinterested players cannot be wholly trusted: first, the players of a game could be in coalition with some disinterested players (in other words,

these players may not be disinterested at all). Second, the disinterested players could be malicious: for the sake of spoiling the game for others (and improving their own ranking), a disinterested player could reveal or falsify information related to the players of a game.

For this reason, the sharing of information with disinterested players must be limited to a minimum.

### The state of the game

The state of P2P Scrabble can be divided into several kinds. The simplest is public state: state available for everyone to read and to modify (under certain conditions). More difficult to manage is private state, available only for one player to read and modify. Other players must have some form of control over the private state – this will be the subject of next sections. The last type of state is concealed public state: any game player can modify this state, but the game players cannot read it. This type of state will be discussed in the section “Concealed public state”. In this section, we shall discuss the simplest form of state in P2P Scrabble: public state.

The public state of P2P Scrabble consists of the board and the letters on the board, and of information about the player that has the turn. Additional information that is required to manage the two other types of state may also become a part of the public state. The private state of P2P Scrabble are the letters that have been drawn by players and have not yet been put on the board. The concealed public state of P2P Scrabble is the letter sack that contains letters that can still be drawn by users.

### Policies for dealing with cheaters

One of the basic questions in the design of a fair P2P game is: how are we going to deal with cheaters? The answer will determine what mechanisms must be used in a P2P game.

There can be many different policies for dealing with cheaters. Here we shall discuss two such policies, that can be thought of as extreme. The first policy must be realized by any fair P2P game:

- allow any player to recognize an action that cheats him. A player that recognizes a cheating action leaves the game.

This basic policy is in many ways unsatisfactory. The player who is forced to leave the game by a cheater will feel justifiable frustration (especially if he was winning). Also, if the game includes rankings, how do we verify that the ranking is fair if a winning player is cheated, or a losing player leaves the game with a complaint of cheating?

Another policy can be thought of as a maximum requirement of fair P2P games:

- control all actions to make cheating impossible, without disrupting the game of honest players.

There exists a relationship between the two policies that also demonstrates the difference in implementation difficulty: if the first policy is realized and all players notice a cheating action, then the players can exclude the cheating player from the game (or punish him in another way, such as making him lose the turn or decrease his score). If the players can do this, then the second policy is implemented, since cheating becomes impossible and cannot disrupt the game of honest players.

However, this does not mean that the two policies are equivalent (note that if the second policy is provided, this implies that the first policy must also be realized. To see why this is so, consider that if a cheating action could go unnoticed by a player, then cheating actions cannot be controlled). In order to exclude a cheating player or punish him, the game players must agree that this is the right thing to do. In the case of a coalition of cheating players, this may be very difficult. To conclude, the two policies are equivalent if there exists a mechanism for fair agreement of all players in the game and more than 50% of the players play by the rules.

### **Fair and secure public state**

As has been noted above, the simplest game state is public state. We define the notion of “public” as available for reading and writing to any game player (but not to any peer). Thus we are relying on the assumption discussed above, that peers in a P2P game are strongly authenticated. It is not difficult to protect the privacy of public state from unauthorized peers (state can be encrypted using a group key). In this section, we consider the question of how to prevent cheating by maliciously modifying public state, or how to provide fair public state.

There have been many solutions for P2P storage or distributed memory in the literature that can be used to implement public game state, such as based on Distributed Hash Tables (DHT) [9,10,11]. The public state must be replicated. Replication mechanisms that use DHT has been discussed in the literature [9,10,11]. This mechanism should be supplemented by a light-weight, distributed transaction protocol that assures desirable properties of operations on public. However, we consider the issues of how to design P2P storage or distributed memory as beyond the scope of this paper. For the discussion of trust management, let us assume that P2P public game state is implemented simply as follows. All game players will store a copy of the entire public game state. All write operations are made on all copies, and all read operations are made on a local copy (in other words, the copies are kept strongly consistent). This is a solution that can be used in P2PScrabble, but

not in all P2P games. In P2PScrabble, the public state is not too large to be stored on all players, and since Scrabble is a game of turns, there are no concurrent write operations. Therefore, for concurrent P2P games or games with large state, there must be other implementations of public state; however, this is an issue beyond the scope of this paper.

Let us discuss the proposed public state implementation in terms of the introduced policies for dealing with cheaters. An honest player that is asked to update his local copy of the public state after the move of another player can immediately verify whether the modification of the game state violates the rules of a game. Also, when an honest player asks other players to update their copies after his move, he can verify whether all updates have been accepted and recognize any attempt to cheat him. Thus, the proposed implementation allows to realize the first policy: any player can recognize any action that cheats him, and leave the game. Note, however, that the policy is realized only with respect to actions that operate on public state, not on private or concealed public state.

What is required in order to realize the second policy? As has been discussed before, the players must agree whether a write operation is the result of a cheating action. Since the game players do not trust each other, they must use a byzantine agreement algorithm [22].

It has been shown [22] that if the participants are authenticated, Byzantine agreement algorithms can tolerate up to a third of cheating peers ( $2N+1$  honest peers can tolerate  $N$  cheating peers). Therefore, the proposed implementation of public state can realize the second policy even if coalitions of cheating game players exist, provided that not too many peers cheat. In P2PScrabble, this means even a coalition of 2 players can break the game, since in Scrabble usually at most 4 players play a game. Therefore, in practice it may be very difficult to realize the second policy in the presence of coalitions of cheating players. Note also, that Byzantine agreement algorithms exist that have constant running time with respect to group size, although their communication complexity is high.

### **Random draws in P2P Scrabble**

In order to prevent the possibility of cheating by the player, it must be possible to prove to the competitors that the player has legitimately drawn a letter that he wishes to put on the board. A straightforward solution to this problem would be to make all draws public. However, such an approach would make it possible for the competitors to cheat by using the information about the letters that have been drawn by the player. Such information could be exploited by the competitors to prevent the player from using his letters.

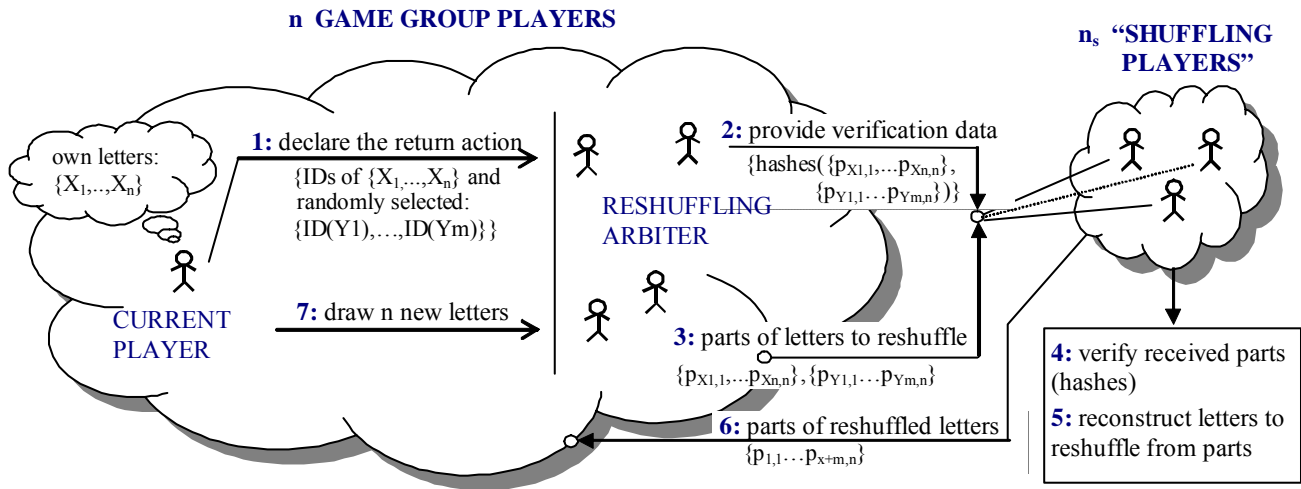


Figure 1 – More efficient reshuffling

Therefore, the player must be able to prove to the competitors that he/she has legitimately drawn a letter without revealing what letter has been drawn. The presented problem addresses the issue of “private” state maintenance.

### Concealed public state

Not only securing the private state is an issue in P2P games. Referring back to P2P Scrabble –Bob should only be able to draw a letter still left in the letter sack (finite-set draws), which motivates the need to maintain the state of the game. The real difficulty is how to let other players know that a given letter can no longer be drawn without revealing the drawn letter or the content of the sack itself. This is what we already have referred to as “public concealed” state.

The implementation of the public concealed state presents the real challenge within a P2P environment, where distrustful players must conceal the state from themselves without a trusted central authority. Let us briefly introduce the main problems that have to be faced when implementing the “concealed public” sack of letters in our P2P Scrabble case study:

- 1) drawing objects from the sack without ever knowing its content
- 2) preventing malicious modifications on the content of the sack
- 3) replication of the sack

Since none of the players can entirely rely on others in a P2P environment, any solution to the above problems should distribute the letter set (“sack” of letters) among the whole game group, so that all of the players have just partial knowledge of its content. Furthermore, the partial knowledge must be replicated by disinterested players in case any of the game players would leave the

game. Finally, there must be some public one-way knowledge (consider hash functions) that would allow to verify that none of the game players modified his part of the letter set.

Due to space constraints, a graphical presentation of the drawing protocol is omitted and the reader is referred to [23]. Let us briefly describe the protocol, which is necessary for the understanding of the complexity analysis and the discussion of scalable reshuffling.

The agents taking part in the algorithm are:  $n$  game players and  $(n_{(mk)} + n_s)$  disinterested players ( $n_s$  shuffling players,  $n_{(mk)}$  replication players). Shuffling players are involved in a distribution of the letter set among the game players (they shuffle the letters). Replication players are responsible for storing replicated parts of letters. The number of replication players -  $n_{(mk)}$  - depends on the level of replication. The relation will be explained later in this section.

The algorithm is composed of two stages: shuffling of letters and the game itself: drawing and revealing of letters. The first stage of the algorithm takes place at the start of the game. The stage is also used whenever any player leaves the game, and his letter parts are restored from the spare parts as well as the return of letters is about to take place. Both cases will be discussed further shortly.

In the first stage, shuffling players execute a secret sharing algorithm on each letter in the letter set. Each letter is divided into  $(n+k)$  parts:  $\{p_{i,1}, \dots, p_{i,n+k}\}$ , where just  $n$  are enough to reconstruct the letter back. The remaining  $k$  parts could be considered as spare, replicated parts not needed as long as all the players are in the game. Secret sharing algorithms have been widely presented in the literature, therefore we shall not discuss details here (see: [12]).

	COMMUNICATION	COMPUTATIONAL	MEMORY
shuffling stage	$O(\text{lettersNo} * (n + m*k))$	1. secret sharing: 1.1.matrix inversion mod p (lettersNo * (k + 1) times) 1.2.matrix multiplication (lettersNo * (k + 1) times) 2.hashing (lettersNo * (n + m*k) times)	1.set of letters to divide: $O(\text{lettersNo})$
	<b>SIMPLE</b>	<b>COMPLEX</b>	<b>SIMPLE</b>
game stage	$O(n) / O(1)$	1.secret sharing : 1.1.matrix inversion mod p 1.2.matrix multiplication 2.hashing (n times)	1.public knowledge (hashes, rep. parts location): $O(\text{lettersNo}*n + m*k)$ 2.private knowledge (secret parts, own letters): $O(\text{lettersNo} + \text{ownLettersNo})$ 3.history of draws: $O(n * \text{ownLettersNo})$
	<b>SIMPLE</b>	<b>SIMPLE</b>	<b>COMPLEX</b>

**Table 1. Analysis of complexity**

(n – number of game players, m\*k – number of replication players)

Later on, still in the shuffling stage, the letter parts are distributed among players:  $n$  of them among  $n$  game players and  $k$  – among  $n_{(mk)}$  replication players. The letter parts are sent by the shuffling players to the game players in groups (letter by letter). The game players determine a unique ID for each received group of letter parts. The ID is used to identify  $n$  parts of a letter when it has to be reconstructed. Please note that a single player knows only his part  $p_{i,j}$  of a letter and its ID. There is no way that the player might guess what the letter is knowing just its ID. In this way, game players may use IDs to refer to letters that are actually concealed from them (“public concealed” knowledge). The only ones that know the whole letter, not just parts of it, are the shuffling players. Therefore the mapping between letters and IDs must be kept secret from them. In the presented algorithm, encryption is a suggested solution to deliver the parts secretly to players. To prevent any malicious modifications of the letter parts, shuffling players calculate hash values of every single letter part and make those public to the game group.

The replication of letter parts is more complex than it may seem at first. It is not enough to distribute  $k$  spare parts among  $k$  replication players. In such an approach, any player could easily find out the result of any drawing being in a coalition with just one of the  $k$  replication players. (In the drawing stage all parts of the drawn letter become public except for the part of the drawing player. This single unknown part could be simply filled with one of spare parts.) To minimize the risk of such coalitions, each of the  $k$  spare parts could be divided into further  $m$  parts using again a secret sharing algorithm. Then, a player would have to establish a coalition with  $m$  random

disinterested players, which is very unlikely. Whenever any  $n_l$  players leave the game,  $n_l$  parts of any letter are lost, though spare parts can be used to reconstruct the remaining letters back. The recovered letters should be passed to shuffling players to be divided once again, this time into  $(n-n_l+k)$  parts, where just  $(n-n_l)$  are enough to reconstruct a letter. These divided parts should be distributed among game players the same way as described in the shuffling stage. Please note that  $k$  should be as large as the greatest number of players allowed to leave the game at once.

Let’s move on to the second stage – the game itself. Please note again that every letter has its unique ID and its  $n$  parts are distributed among  $n$  game players. To draw a letter, the drawing player picks one of the remaining IDs. To inform other players that the letter is no longer in the letter set, the picked ID has to be revealed. The drawing player also asks his competitors for their parts, determined by the revealed ID. These  $n-1$  parts become publicly known, the remaining one – kept by the drawing player – is a secret till the moment when the letter is used on the board. This single concealed part is enough to hide the drawn letter from other players. Note that the drawing player does not have to commit his secret part of the letter, since the hash values published by the shuffling players prevent tampering with the secret part during reconstruction of the letter. The drawing player should also verify that those  $n-1$  parts of the competitors were not modified in any way, by comparing hash values to the ones provided by shuffling players.

To prove that the letter put on the board is the one a player has drawn, the player must reveal the ID and his secret part of the letter. At this moment, all of the letter

parts are public. The competitors may reconstruct the letter to find out whether it is the one put on the board. The only thing left to verify is that none of the parts were actually modified at any point (hash values). Note that the verification of a draw uses the same mechanisms as a verification of any public state modification.

### Returns of letters and scalable reshuffling

A rule of Scrabble that has not been considered so far is the possibility of returning all letters to the sack and drawing new ones at the cost of losing a turn. The move can be implemented in a following manner (Figure 1). First, the drawing player declares to his game group that he wants to return  $l$  letters and at the same time specifies IDs of further  $j$  randomly picked letters, where  $j$  is some fixed number. The relation between both numbers will be explained shortly. Second, the game group elects a *reshuffling arbiter* that sends public hash values of all parts of  $l+j$  specified letters to shuffling players. Finally, all the game players send their parts of specified  $l+j$  letters again to shuffling players. The letters are reshuffled in exactly the same way as described before, except for the fact that they need to be reconstructed from the parts first. Reshuffling players should use provided hash values to make sure that none of the parts has been falsified. After the reshuffled parts have been returned to the game group, the game player is allowed to draw his  $l$  missing letters using the same approach as presented previously.

The reason to reshuffle just  $l+j$  letters instead of the whole set is naturally to improve the scalability of the suggested solution. Reshuffling a limited number of letters leaves, however, some potential for attacks. Attention should be paid to properly pick the value of  $j$ , such that the probability of guessing of the returned letters will be as small as required. Since it is reasonable to assume that the cheating player needs to find all of the  $l$  letters - otherwise the possibility of preparing a strategy is negligible - the probability of potential attack can be expressed by the formula (assuming that the letters do not repeat themselves):

$$P_{guess}(l, j) = \frac{1}{\binom{l+j}{l}} = \frac{l! j!}{(l+j)!}$$

In our P2P Scrabble case-study, where each player has 7 letters ( $l = 7$ ) – further 3 letters ( $j = 3$ ) suffice to keep the probability of potential cheating ( $P_{guess}$ ) below 0.01, while 2 additional letters ( $j = 2$ ) result in a still reasonable  $P_{guess} < 0.03$ .

## 5. Analysis of presented solution

### 5.1. Feasibility study of attacks

### Attack on cryptographic protocols and functions

The possibility of cryptanalytical attacks on the mechanisms described in the paper cannot be neglected. The finite-set approach was mostly based upon secret-sharing mechanism, which opens the possibility for the attack, where malicious player presents falsified share and obstructs further game. Secret sharing schemes exist that allow to identify the cheating player [20, 21]. These however are mostly based on “check vectors” – the concept where pairs of share owners can verify that the partner revealed the correct share. None of the solutions in the literature provided enough flexibility to accommodate our algorithm. Therefore we decided to address the problem of falsified shares with the help of public hash values, which unfortunately raises the possibility of another attack... on the hashing function.

A player may attempt to find the secret (share) of another player with the help of public hash value. A player may as well attempt to replace his secret part of the letter with some other having the same hash value. Both attacks seem very unlikely, considering the effort needed -  $O(n)$  to find the secret and  $O(n^2)$  to find the same hash value, where “n” is the number of all possible parts of all of the letters. This number is very large due to randomization properties of secret sharing algorithm that returns a different set of shares (parts) for the same secret (letter). This property makes both attacks on the hashes of letter parts very difficult.

### Coalitions

The presented algorithm has one serious shortcoming. Coalitions between distribution players and game players are possible. The first ones know the letters, the latter ones know the IDs. The combination of this knowledge would obviously make all draws unfair. One could possibly get away with that using anonymous proxy communication between distribution players and game players. That way the coalition would have to include two random disinterested players – proxy and distribution player to make it worthwhile.

Another approach would be to assume that the game contains trusted superpeers that function as distribution players. Note that the role of the superpeers is limited to the beginning of the game and to dealing with node leaves (redistribution of letters).

### 5.2. Performance analysis

The security solution may become accepted by the community of users only if it is sufficiently efficient. Therefore one of the key design goals has been to keep complexity of the algorithm reasonable, especially during the game stage.

According to our analysis of complexity – shown in table 4.1. – substantial computational/communication effort is required only in the shuffling of letters that occurs only in the initial stage of the game and just occasionally thereafter (players departures, returns). The complexity of the game stage seems reasonable. As the memory limitations in the present systems have less and less impact on the overall complexity constraints, we decided to put the lowest priority to memory resources, striving to reduce the complexity of the computations and communication.

To briefly summarize our complexity analysis – we believe that the finite-set drawing algorithm has reached set goals. The performance tradeoff is within acceptable boundaries, which makes it even well suited for mobile devices. It still needs further improvements to fully scale to more complex P2P games.

## 6. Summary

The presented design of P2P Scrabble has been developed as a case study of a more general question: how to design P2P games without central coordination so that no player can cheat?

Note that the presence of superpeers could greatly increase the efficiency and fairness of public state. For the simple public state implementation discussed for P2P Scrabble, a Byzantine agreement algorithm must be implemented to realize a policy that will exclude cheating players from the game. A superpeer could act as a simple vote counter for a game group, thus limiting the number of messages needed to reach an agreement ( $O(N^2)$  in Byzantine agreement,  $O(N)$  with a superpeer) and raising the tolerance of cheating players (only a simple majority of the game group must be honest players, compared to over  $2/3$  in Byzantine agreement). Other such improvements are probably possible and could be the issue of future work.

Returning once again to the original question, we have to consider what other issues might arise in other P2P games that have not been considered in our case study. In other words, how can other games differ from Scrabble?

Several other games could use the design presented here for P2P Scrabble (consider, for instance, bridge or poker). However, all of these games are games of turns. In our design, we have not considered the issue of concurrency of game player actions such as drawing letters, assuming that the drawing player is the player who has the turn. Considering multi-user games without turns is an issue of future work.

## 7. References

1. C.Farkas, G.Ziegler, A.Meretei, A.Lorincz (2002), Anonymity and Accountability in Self-Organizing Electronic Communities, Proc. ACM workshop Privacy in the Electronic Society, 81-90
2. K.Aberer, Z.Despotovic (2001), Managing Trust in a Peer-To-Peer Information System, Proc. tenth int. conf. Information and knowledge management, 310-317
3. M.Gupta, P.Judge, M.Ammar (2003), A Reputation System for Peer-to-Peer Networks, Proc. 13th int. workshop Network and op. sys. support for digital audio and video (ACM Press), 144-152
4. B.Yu, M.Singh (2002), An Evidential Model of Distributed Reputation Management, Proc. first int. joint conf. Autonomous agents and multiagent sys., part 1, 294-301
5. Wierzbicki, R. Strzelecki, D. Świerczewski, M. Znojek (2002), Rhubarb: a Tool for Developing Scalable and Secure Peer-to-Peer Applications, Second IEEE Int. Conf. Peer-to-Peer Computing, P2P2002,
6. Gnutella/ng, World Wide Web page, [http://mangocats.com/annesark/gnutellang/wego\\_pages.html](http://mangocats.com/annesark/gnutellang/wego_pages.html), 2000
7. Freenet, World Wide Web page, <http://freenetproject.org/cgi-bin/twiki/view/Main/WebHome>, 2002
8. Mojo Nation, World Wide Web page, <http://www.mojonation.net/>, 2000
9. Stoica, R. Morris, D. Krager, M. F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, Proceedings of ACM SIGCOMM'01 Conference, 2001
10. P. Druschel, A. Rowstron, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01), 2001
11. Zhao, J. Kubiawicz, A. Joseph, Tapestry: An infrastructure for fault-resilient wide-area location and routing, Technical Report CSD-01-1141, U.C.Berkeley, 2001
12. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of applied cryptography, CRC Press, ISBN: 0-8493-8523-7, October 1996
13. Singh, Ling Liu, TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems, Proc. IEEE Peer-To-Peer Conference, 2003
14. Y. Wang, J. Vassileva, Trust and Reputation Model in Peer-To-Peer Networks, Proc. IEEE Peer-To-Peer Conference, 2003
15. G. Caronni, M. Waldvogel, Establishing Trust in Distributed Storage Providers, Proc. IEEE Peer-To-Peer Conference, 2003
16. E. Sit, R. Morris, Security considerations for peer-to-peer distributed hash tables, Proc. IPTPS02 Workshop, 2002
17. L. Mui, Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks, Ph.D. Dissertation, Massachusetts Institute of Technology, 2003
18. P. Gmytrasiewicz, E. Durfee, Toward a theory of honesty and trust among communicating autonomous agents Group Decision and Negotiation 1993. 2:237-258.
19. J. Douceur, The Sybil Attack, In Proc. of the IPTPS02 Workshop, Cambridge, MA (USA), March 2002
20. Erik Warren Selberg, How to Stop A Cheater: Secret Sharing with Dishonest Participation, Carnegie Mellon University, 1993
21. Martin Tompa and Heather Woll, How to share a secret with cheaters, Research Report RC 11840, IBM Research Division, 1986
22. L. Lamport, R. Shostak, M. Pease, Byzantine Generals Problem, ACM Trans. on Programming Languages and Systems, 1982, 4, 3, pp 382-401
23. A. Wierzbicki, T. Kucharski, P2P Scrabble: Can P2P Games Commence?, Proc. IEEE International Conference on P2P Computing, August 2004