Secure Peer-to-Peer Trading for Multiplayer Games

Chris GauthierDickey and Craig Ritzdorf
Department of Computer Science
University of Denver
chrisg@cs.du.edu | critzdor@cs.du.edu

***Abstract*—A fundamental aspect of many multiplayer online games is the ability to trade items between players. This might take the form of items that were found in the virtual environment or of purchased assets, but in either case, any multiplayer game that supports trading or exchanging items in the game must do so in a secure manner. We have developed a protocol to solve the problem of secure, peer-to-peer trading in games in which the primary concern is that items are exchanged fairly, and additionally that items are not *duplicated*. Our protocol enables one-way and two-way trades and can be extended to multi-item trades. We show that our protocol addresses the security threats which it might encounter, and then provide an analysis to demonstrate the scalability of our protocol.**

I. Introduction

Almost all types of online multiplayer games provide some mechanism for players to exchange virtual items or game currency. Furthermore most, if not all, massively multiplayer online games also provide the ability to auction virtual items to other players. Unfortunately, little work has been done in the realm of peer-to-peer (P2P) games to address trading. Therefore, we have developed a protocol for secure, P2P trading that ensures virtual items can be traded fairly while catching the *duplication cheat*, which allows players to unfairly duplicate items in the game.

While one might assume that trading is something that should be refereed by a centralized server (i.e., to prevent the duplication cheat), game companies have a motivation to use peer-to-peer protocols in implementing features of their game to reduce computational overhead, network bandwidth, and downtime caused by hardware failures. By using a P2P approach, one can leverage the computational power of all of the players in the game without sacrificing security. Furthermore, the protocol we have developed is compatible with a centralized server such that the bulk of the transaction can occur between peers, but a centralized server can be the final arbiter in the trade if desired.

The duplication cheat is possible in P2P multiplayer games because without complete control of a player's machine, one cannot ensure that an item is actually deleted from their list of items *after* they have traded it to another player. Furthermore, since virtual items are exchanged, the use of the duplication cheat may allow the receiving player to view the trade as successful even though the originating player may still be using the traded item. By the same token, reputation systems do little good in preventing the duplication cheat. The receiving player would not necessarily know that the originating player had not deleted the item and thus would not report on them negatively. We do note, however, that a player might try to leave a transaction incomplete, and if they do so frequently, a reputation system might help in making it less likely for others to try to trade with them.

Prior research in distributed trading has been in the form of *fair exchange protocols*, where a trade is considered fair *if and only if* the trade either occurs, and both parties complete the exchange of items, or the trade is aborted, and neither party receives the items. Beyond fair trades, some research has looked at building secure and distributed auctions (e.g., for eBay), such as the work by Rolli et al. or Fontoura et al [1], [2]. While we do not address auctions in this paper, we believe that research in distributed and secure auctions is certainly applicable towards auctions in multiplayer games. Finally, Bitcoin [10], a digital currency, is directly related to trading since they allow P2P trading of their currency. However, we show why it is not a full solution for trading in P2P games.

Our algorithm works on a simple principle: provide proof of ownership and provably show that any trades that occurred are valid. We then use a weak third party, in the form of a distributed hash table (DHT), to prevent the duplication cheat by allowing players to look up items owned by other players on the DHT. Items are temporarily locked while trading to prevent either player from using those which are to be exchanged.

We analyze the performance of our protocol to show its scalability. Many modern multiplayer games have design limitations on the number of times an item can be traded. For example, most massively multiplayer online games (MMOGs) allow you to trade some items as long as you have not used them. Once they are used, they are *bound* to your character. Some items are further limited in tradability in that as soon as you pick them up, they cannot be traded. In essence, items typically experience an upper bound on the number of times they will be traded to other players.

This work provides the first look at trading virtual items in a completely distributed fashion and as such we believe it is an important step in developing a fully distributed architecture for multiplayer games. In addition, the trading protocol is agnostic towards the underlying network architecture (though we assume some type of DHT will be used with $O(\lg n)$ lookups and insertions), allowing it to be used with any P2P

architecture for games.

II. Background

The most closely related research to our work centers around fair exchange protocols. In these protocols, two parties are trying to exchange digital assets and desire the exchange to be simultaneous, such that either both receive the exchange, or neither receive it [3]. In addition, fair exchange protocols usually hope to prevent any useful information from being exchanged, such that the data of the exchange should also be hidden during the transmission until both parties receive it. The second area most closely related to our work centers around P2P architectures for multiplayer games.

A. Fair Exchange Protocols

Fair exchange protocols are extremely useful for various cryptographic operations such as certified mail (i.e., receiving a receipt for the mail simultaneously with the reader being able to read the mail), purchases (i.e., payment is received simultaneously with delivery of the digital asset), trading (i.e., exchanging two items simultaneously), and contract signing (i.e., players exchange non-repudiable commitments to the text of a contract). Obviously, the primary usage of fair exchange protocols in games relates to trading virtual items.

Research in fair exchange protocols can be broadly categorized as either *gradual* protocols or *third party* protocols. In gradual protocols, the exchange of data between the two parties occurs over several rounds so that the probability of receiving the correct data by the termination of the protocol increases with the number of rounds of execution. In third party fair exchange protocols, a trusted third party is used as an arbitrator to ensure the trade occurs fairly.

One of the first gradual fair exchange protocols was due to Ben-Or et al. and designed for signing contracts [4]. In this protocol, player X and Y exchange messages of the form "With probability p, this contract C shall be valid. (signed by X or Y)." With each iteration of the protocol, p is increased monotonically. An expiration date is set additionally, so that if either party chooses to abandon the protocol, a judge (i.e., trusted third party) can then choose a random value from 0-1 and determine whether the contract is valid or not.

In Damgård's protocol, instead of exchanging signed messages, a series of hashed blocks of the contract are exchanged first, and then the plaintext block in each iteration of the protocol is revealed [5]. If either participant attempts to give incorrect information, the judge can bias her decision based on how many valid plaintext messages were received by the complaining party. Okamoto and Ohta developed similar gradual fair exchange algorithms using one-way permutations and one-way functions [6]. Syverson allows weak secret bit commitment in fair exchange protocols to create a more efficient, but weaker version of gradual exchanges, with the idea that instead of using hash functions or encryption which are infeasible to break, weaker versions may be used as long as it would take longer to break than the amount of time that the information being protected is useful [7].

The major problem with gradual fair exchange protocols is the number of messages that might have to be exchanged, including an expiration date that could either be too soon or too late. Even in the case of gradual fair exchange protocols, a trusted third party is still sometimes needed (e.g., [4], [5]).

Trusted third-party protocols involve a third-party that is trusted by both participants in an exchange to resolve any issues in the exchange. Asokan et al. developed asynchronous protocols for optimistic fair exchanges where a trusted third party is only needed to abort an exchange or to resolve an incomplete exchange [3]. In their protocol, both parties exchange bit commitments (e.g., random strings of digits) in the form of cryptographically secure hashes which they sign. Once these are exchanged, they exchange the plaintext of their hashes to show that the exchange is complete. The trusted third party is only needed if the protocol needs to be aborted by either participant and is accomplished through the use of a special abort token. Bao developed a similar fair-exchange protocol but only required an off-line trusted third party [8], while Franklin and Reiter developed a protocol which relaxed the trust level of the third party participant [9].

Our protocol differs from both gradual and trusted third party fair exchange protocols in that: 1) the information being *exchanged* is publicly available and so it's not imperative to hide the information completely prior to exchange, and 2) uses a weak third party to store this information to prevent the duplication cheat. We note that the fair exchange protocols described do not handle the duplication cheat (though it's not normally an issue for the types of problems fair exchange is typically applied to).

Bitcoin [10], a digital currency, uses cryptographic signatures of "coin" transactions which are then wrapped in "proof-of-work" chains. This provides for a one CPU, one vote mechanism. As long as the majority of CPUs on the network are honest, dishonest transaction records will be removed from the network by virtue of the correct chain being longer. Bitcoin's procedure for creating one-way transactions uses a form of gradual fair exchange in that a transaction becomes more *valid* as the proof-of-work chain becomes longer. In addition, Bitcoin only describes one-way exchanges, requiring all transactions to be broadcast to all other users. Finally, Bitcoin's reliance on computation for consistency is disadvantageous in gaming where game performance is often bounded by the CPU or GPU.

B. Peer-to-Peer Architectures

In our protocol, we expect that any P2P architecture that is used will likely have some type of distributed hash table built for network communication between players. A distributed hash table (DHT) typically provides $O(\lg n)$ lookups on a *key* and returns a *value*. DHTs typically come in two flavors: structured and unstructured, with structured DHTs guaranteeing that a key will be located quickly (generally $O(\lg n)$ time) while unstructured DHTs not providing guarantees that if a key exists it will be found, though providing other guarantees such as locality.

TABLE I
ITEM PROFILE STORED IN DHT

Field	bits
UUID	128 bits
Date	64 bits
Owner signature	512+ bits

Because we need to be sure that if we look up an object by its universally unique identifier that we find it in the DHT, we need some type of structured DHT for our protocol (e.g., Chord [11], CAN [12], Pastry [13], Mercury [14], and Kademlia [15]). To the best of our knowledge, none of the peer-to-peer architectures for games in the literature has considered how to handle in-game trading without resorting to a centralized server.

III. PEER-TO-PEER TRADES

A. Assumptions

We make a few essential assumptions in order for our protocol to work. First, we assume that all game items are digitally signed by a signing authority. This is typically done through a centralized server, and is required for identifying valid items in the game. However, a centralized authority may not be needed to ensure the items in a P2P game are valid– for example, an architecture might have some subset of the current players sign an item to prove it was generated validly in game, or may further use a hybrid of a central authority for important items and P2P for the rest. We do not explore this further in this paper.

Second, we assume that all players have a universally unique ID (UUID) that is digitally signed with a date to show that they are currently valid players in the game. Again, this could be done by a central authority, P2P, or in a hybrid manner. Each player is able to sign items to show ownership, and other players can verify those digital signatures.

Third, we assume that the P2P game runs some variation of a distributed hash table (DHT). While the specifics of the DHT are not extremely important (though obviously security is important in games), we do assume that lookup and storage on the DHT occurs in $O(\lg n)$ time.

We use the DHT to store item profiles. An item profile includes the item unique identifier plus digital signatures indicating ownership. To show ownership, the UUID of the player is appended to the UUID and creation date of the item, which then has the digital signature of the player appended. Table I shows the fields of the item profile stored in the DHT.

We argue that any P2P architecture for multiplayer games would necessarily include all of these assumptions.

B. Definitions

We use the following terminology to describe our protocol:

- UUID: universally unique identifier, typically a 128-bit number
- msg = "I agree to something": a plain-text message, abbreviated by using msg as a variable (typically for digital signatures).
- $S_A(msg)$: A digitally signed message by A. Recall that a digital signature is usually an encrypted cryptographically secure hash of a message, and by itself does not leak information about the contents of the message.

C. One-way Trades

From a high-level perspective, a one-way trade works by having one player sign ownership of an item over to another player. When other players see this item, it includes the signatures of the original owner and all consecutive one-way trades so that other players can verify the signatures to check for valid ownership. Once a trade has occurred, the receiving player posts the new item on a DHT. Recall that a DHT stores items using a $(key, value)$ pair. Items are stored using their UUID as the key, with the value being the item profile.

The primary problem with simply signing items over to another player in a P2P architecture is that the original owner has the ability to cheat by signing the same item over to several players without deleting their version of the item. By doing so, they can still use the item which appears to be valid by other players even though they may have exchanged it for other items or game currency. We call this the *duplication cheat*.

We believe that game companies are interested in controlling the game items specifically for game design purposes and as such, the duplication cheat is fatal to almost all multiplayer games where items need to be controlled. In many multiplayer games, some items are considered *powerful*, as they grant unique abilities or are significantly better than other game items. If players are allowed to duplicate such items with impunity, the game balance can be significantly disturbed.

We prevent the duplication cheat with our protocol as follows. First, assume that the item was generated properly with a UUID, a creation date, and a valid signature (see Table I). To trade an item, the UUID and digital signature of the receiving player is appended to the item profile. The player trading the item then appends their UUID and digital signature to the item to show that they consent to the trade. Note that a digital signature is the encrypted secure hash of an item. Thus, each additional signature includes information from all prior signatures ensuring that the data cannot be altered without making the signature invalid.

After the trading player appends their UUID and digital signature to approve the trade, the receiving player posts the item with its new profile to the DHT. The DHT acts as a "weak" trusted third party, meaning that a node on the DHT may be cheating, and therefore we must use replication to reduce the probability of choosing a colluding node accidentally. Any player can look up the item on the DHT to verify valid ownership. This of course implies that one additional step is needed by our protocol. When a player joins the game, they post their inventory of items to the DHT so that any player can look up the item as needed. Figure 1 shows how an item is traded from Alice to Bob.

Interestingly, the data on the DHT can be old, and a player will be caught cheating as long as the data on the DHT is at least one trade newer than that of the cheating player. If

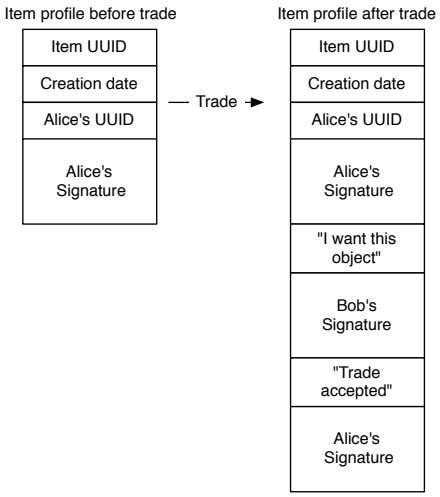

Fig. 1. Example of a one-way trade: Alice trades an item to Bob. Bob signs the item with a message showing that he wants the item, and Alice then signs it with a message showing that she agrees to the trade. Of course the messages "I want this item", and so forth, can be significantly smaller and are just used to make obvious what the players are signing.

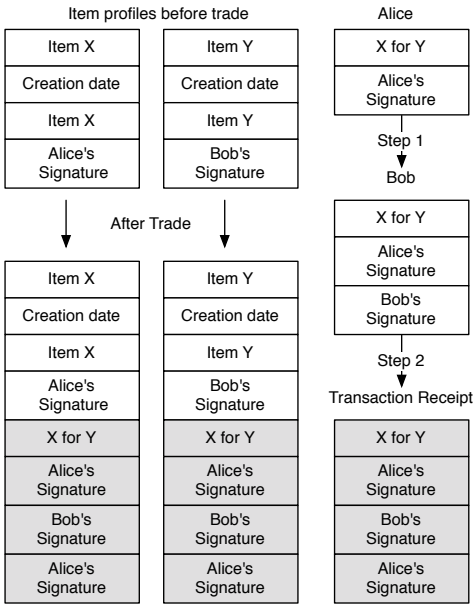

Fig. 2. Two-way trade: The left side of this figure shows the transformation of the two items from when they first start to after the trade has occurred and the transaction receipts have been appended to the items. The right side of the figure shows Alice sending her request to Bob, who signs it and returns it back to Alice. Once she signs it, the transaction receipt is complete.

a player queries the DHT and finds older data, then they can simply update it with the newer data they've seen.

D. Two-way Trades

One-way trades are applicable when one player wants to give another player an item. The more common scenario in multiplayer games is likely to be bartering: one player would like to exchange one or many items with another player. For a two-way trade, we again consider Alice and Bob, where Alice would like to exchange item X for item Y that Bob owns. For simplicity, without loss of generality, we layout a one for one trade. We begin with the case where neither item has been traded previously, either in a one-way trade or two-way trade.

From a high-level perspective, our protocol has two phases: a commitment phase, where both players commit to the trade, and a completion phase, where the trade is completed. The commitment takes the form of a message signed by both players that they agree on the trade. This message can then be appended (and stored in the DHT) on both items to "lock" them, and this lock encourages both players to continue through to the completion phase of the protocol. Once the initiating player receives the commitment, they can sign this message a 2nd time and this becomes the transaction receipt which is henceforth attached to the item.

We begin with the commitment phase and without a loss of generality, Alice goes first (see Figure 2):

1) Alice signs the message "X for Y with sequence number 1 on DATE", which we call $trade$, and passes this to Bob ($S_A(trade)$). The sequence number used in a trade is the greater of the two numbers obtained by adding 1 to the last transaction on the item. Thus, sequence numbers always increase monotonically. All transactions also include a date that should be checked to be reasonably close to what the other person thinks the correct date and time are.
2) Bob takes the digitally signed message $S_A(trade)$ and signs it to show his agreement. He passes this back to Alice ($S_B(S_A(trade))$). If he doesn't agree with the trade, he simply doesn't do anything else.
3) Once Alice has this message, or once Bob has signed this commitment to the trade, either player can post this message to Item X and Item Y to the DHT in both UUID positions. This acts as a "lock" on both items, and when it exists, they are unable to use the items that are committed to a trade.
4) When Alice receives this signed commitment from Bob, she signs it a final time to complete the transaction ($S_A(S_B(S_A(trade)))$). This can then be posted to the DHT to replace the lock on the items to show that the trade is complete.

Once the trade is complete, the players have now swapped items and must keep track of all trades associated with the item. Each trade of an item causes a message to be appended to it to show the trade so that the entire sequence of trades can be followed (A to B to C...). In addition, each trade has a sequence number associated with it, which allows circular trading paths (e.g., Bob trades to Alice who trades back to Bob). Again, when a player logs into the game, they post their inventory to the DHT (checking it first to make sure it's not already there), since the DHT may have lost information on their items. Many for one and many for many trades can be accomplished by simply including all items in the trade description.

TABLE II
SECURITY THREATS IN THE TRADING PROTOCOLS

Threat	Resolution
Alice does not store her items on the DHT when she first logs in	Old data is not an issue as players can store items on the DHT when they find one that a player has which doesn't exist yet.
Alice refuses to append her signature on the item profile to validate the trade (one-way trade)	Bob can simply end the transaction since it's invalid without her signature.
Alice attempts to alter the item before trading it	Prevented by digital signatures when the item was first created. All appended signatures are on the entire virtual item profile.
Alice does not delete the item after trading it (duplication cheat)	Players randomly lookup items on the DHT that are owned by other players when they interact with them.
Bob does not sign the commitment to trade (two-way trade)	The trade is not valid until Bob signs it, but the item can still be used.
Alice does not sign the completion to trade (two-way trade)	Alice's item is locked so she cannot use the traded item until she does so.

E. Multiple Item Trades

Both the one-way trade and the two-way trade protocols may be used to trade multiple items. For example, rarely do two players wish to simply exchange items that have the same value. Instead, one player might offer a single desirable item in exchange for several items. The two-way trade protocol can handle this by simply extending the messages "X for Y" to be "X for Y, Z, P, Q". These items would all then be locked via messages which would need to be signed by both parties. One-way trades are trivially transformed into multi-one-way trades by performing m one-way trades.

F. Scalability

Of course, in it's current incarnation, the protocol is not very scalable. To assist with the scalability issues, we require players to examine the DHT only randomly on items they see. The higher the probability they examine the DHT for an item, the more likely that they will discover a player has used the duplication cheat. We explore this further in Section IV.

G. Protocol Threats

We now discuss the security threats to this protocol and show how they are addressed. Table II lists the security threats to the protocol, and how they are addressed, when Alice is trading an item to Bob. We assume that all simple threats, such as malformed protocol messages can easily be checked by either party and can be avoided at the commitment stage by not signing the message.

There are a few issues with our protocol that merit discussion. First, the major problem with all fair exchange protocols is that someone can walk away in the middle of a transaction leaving the other player stuck. By using two phases, one player can at least lock the other player's item if they are not completing the transaction. The worse possible situation is if they disappear for some significant amount of time, then return and try to complete the transaction. To handle this problem, an additional restriction could be placed on messages in the form of an expiration. In other words, once Bob commits to the trade with Alice, a date could be added so that if Alice doesn't complete the transaction by then, Bob can go ahead and trade with someone else. Indeed, Bob's item is locked in the interim, but if this time limit is reasonably small (e.g., 1 hour to account for clock differences), then they are not inconvenienced for a long time.

Second, in our protocol, a player in theory could commit to trading the same item to multiple players (assuming they all asked to trade simultaneously). However, as soon as the first commit occurred, that player now has proof that they were trying to cheat. Even if multiple trade completions occur for the same item, this will be discovered as soon as those players try to take ownership of the item and post to the DHT. Since we are using digital signatures, they now have proof of the cheating player and severe consequences can be levied.

Finally, as discussed previously, since we query the DHT on randomly chosen items from players we interact with, anyone using the duplication cheat will eventually be caught.

IV. PERFORMANCE ANALYSIS

In order to understand the performance and scalability of our protocol, we analyze the effect that trading items has on the growth of the item profile that has to be stored in the DHT, the number of signatures that have to be verified as the item has been traded, and finally the probability that when you're verifying an item you have asked someone on the DHT who is colluding with the person cheating. We assume that digital signatures are approximately 1024 bits in size.

For item size growth on the DHT, we measured how large the items grow as the number of times the item is traded increases. Figure 3 shows the results and demonstrates that the item size increases linearly with the number of trades. This is expected since we append three signatures for each trade. Our results also show that the computational cost of verifying traded items increases linearly with the number of trades. We note again that in many games, once an item is equipped or used, the item can no longer be traded, so we expect the total number of trades on an item to remain relatively low.

One problem with using a DHT for checking the ownership of items is that if a group of players are colluding, we have some probability of choosing them when we are looking up the item. The distribution of UUIDs for items helps prevent some of these issues, but some DHTs are weaker than others in randomizing a node's storage address space, so we still have some probability of having the item live on a colluding player's node.

To reduce this possibility, we can increase the number of places an item is stored on the DHT by replicating it so that the odds that all of those replications will be colluding players drops significantly. Just finding *one* non-cheating node would allow you to identify all of the colluding players. Thus, we look at the probability that we will pick someone that is colluding with the player we are checking. To do this, we

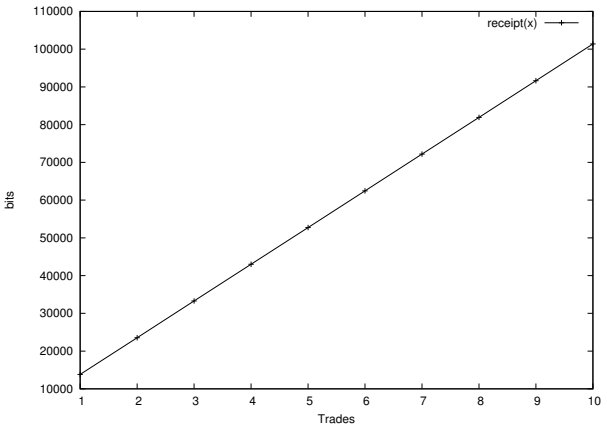

Fig. 3. Item profile size versus number of trades: The item profile size increases linearly with the number of times the item has been traded.

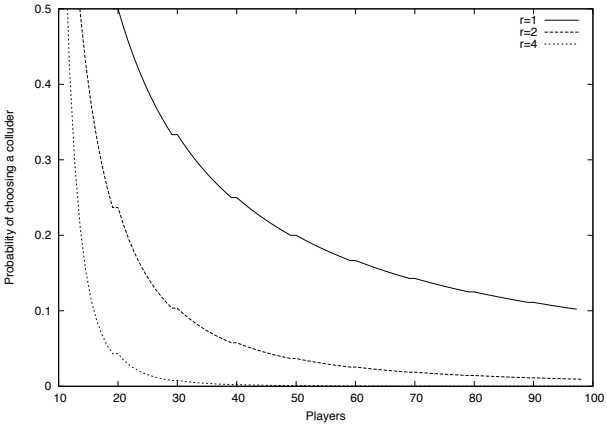

Fig. 4. The probability of picking someone else that is cheating versus the number of alternate people we might pick for replication (the r value). In this case, 10 players are colluding.

assume that items are replicated on the DHT up to r times. If c players are colluding, then ideally we want to find a tolerance value:

$$T < \binom{c}{r} / \binom{n}{r}$$

to ensure that the odds of running into *only* colluders is sufficiently low. When T is higher, cheaters using the duplication cheat will take longer to be caught. Figure 4 shows the results of using 1, 2 and 4 replications on the DHT with 10 players colluding. With 100 players, this implies that 10% of the players are colluding together, but also shows that with as few as 4 replications, the likelihood of those 4 nodes being in collusion is less than 1%.

To help reduce the number of colluding players, one would need a mechanism to reduce the ability of a player to have multiple accounts and play simultaneously. However, even if the players collude, they only need one item stored on a non-colluding node in the DHT to have irrefutable evidence that they are cheating.

V. Conclusion

In conclusion, we have developed the first protocol for exchanging items fairly in a peer-to-peer game. Our protocol requires the backing of a DHT to act as a "weak" trusted third party in order to prevent the duplication cheat. We have seen that as the number of times an item is traded, the storage and computation costs grow linearly. Unlike traditional fair exchange protocols, we do not need to keep items completely secret from each other, allowing us to develop a more efficient protocol for P2P games. We then showed that by replicating items on the DHT we can ensure that the chances of picking a colluder is sufficiently low.

As part of our future work, we plan on exploring ways in which we can simulate trades so that the actual networking costs of exchanging items and digital signatures can be measured further in a detailed analysis. We also plan on exploring how items can be added and deleted from the game.

References

[1] D. Rolli, M. Conrad, D. Neumann, and C. Sorge, "An asynchronous and secure ascending peer-to-peer auction," in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, ACM, 2005, pp. 105–110.

[2] M. Fontoura, M. Ionescu, and N. Minsky, "Decentralized peer-to-peer auctions," *Electronic Commerce Research*, vol. 5, no. 1, pp. 7–24, 2005.

[3] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *IEEE Symposium on Security and Privacy*, 1998, pp. 86–99.

[4] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE Transactions on Information Theory*, vol. 36, no. 1, pp. 40–46, January 1990.

[5] I. B. Dåmgard, "Practical and provably secure release of a secret and exchange of signatures," in *Advances in Cryptography — EUROCRYPT'93*, ser. Lecture Notes in Computer Science, T. Helleseth, Ed. Springer Berlin / Heidelberg, 1993, vol. 765, pp. 200–217.

[6] T. Okamoto and K. Ohta, "How to simultaneously exchange secrets by general assumptions," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM, 1994, pp. 184 – 192.

[7] P. Syverson, "Weakly secret bit commitment: Applications to lotteries and fair exchange," in *11th IEEE Proceedings of Computer Security Foundations Workshop*, June 1998, pp. 2–13.

[8] F. Bao, "Efficient and practical fair exchange protocols with off-line ttp," in *Proceedings of IEEE Symposium on Security and Privacy*, May 1998, pp. 77–85.

[9] M. K. Franklin and M. K. Reiter, "Fair exchange with a semi-trusted third party," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 1–5.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/about.html, 2008.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, 2001, pp. 149–160.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, 2001, pp. 161–172.

[13] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*. Springer-Verlag, 2001, pp. 329–350.

[14] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *Proceedings of ACM SIGCOMM*, August 2004.

[15] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin / Heidelberg, 2002, vol. 2429, pp. 53–65.