A Secure Event Agreement (SEA) protocol for peer-to-peer games

Amy Beth Corman†, Scott Douglas†, Peter Schachte†& Vanessa Teague
†National ICT Australia, Victoria Lab
Department of Computer Science & Software Engineering
The University of Melbourne
Email:{amy,scdougl,schachte,vteague}@csse.unimelb.edu.au

Abstract

Secure updates in a peer-to-peer game where all of the players are untrusted offers a unique challenge. We analyse the NEO protocol [5] which was designed to accomplish the exchange of update information among players in a fair and authenticated manner. We show that of the five forms of cheating it was designed to prevent, it prevents only three. We then propose an improved protocol which we call Secure Event Agreement (SEA) which prevents all five types of cheating as well as meeting some additional security criteria. We also show that the performance of SEA is at worst equal to NEO and in some cases better.

1 Introduction

Massively multi-player online games (MMOGs) typically consist of many thousands of players interacting in a virtual world. The persistent game state and the large investment of time by players makes the prevention of cheating critical to a successful game. In the traditional client server architecture for MMOGs, security is addressed through secure communication with a trusted central server, usually using off the shelf tools. These central servers require a large investment in CPU, bandwidth and maintenance resources. They also represent a single point of failure and congestion. P2P networks can be used to distribute the responsibility for game state and logic over participating machines, removing the need for a central server. Doing so can improve performance, scalability and robustness.

The use of P2P networks for this purpose poses a unique security problem since all parties are untrusted. In essence this means that MMOGs must facilitate a method for participants to reach a consensus on the current state of the game in a way that prevents malicious individuals and groups from cheating. There are two main classes of communication that are required for this; low frequency messages, such as for trading, group forming and permanent world altering operations, and high frequency updates which communicate rendering information such as position, direction and action of virtual entities, such as avatars and non-player characters. This paper is concerned with a protocol that facilitates the latter type of communication, which has additional performance constraints. The protocol provides a mechanism for peers to reach agreement, presenting a consistent view of the game state to all players.

Scalability is maintained in the network by ensuring updates are sent only to peers holding interested entities, i.e. those which consume the information, for example to render a remote player's avatar. The two main approaches to interest management in P2P networks are region based and neighbour based methods. Region based methods provide either a subscription to a space within which all published updates from other entities will be received [3, 4] or the construction of a multi-cast tree of all peers with entities within a given region which is used to propagate updates to all local members [8]. Neighbour based approaches form connections in the network based on the virtual proximity of the entities on those peers [7, 6]. Updates are then passed directly between neighbours.

Region based methods incur a communication cost from routing over the entire P2P network, for example $O(\log n)$ for Chord [13], where n is the number of peers, both to and from the peer responsible for a region. The benefit of this method is that since these regions are generally distributed randomly over all peers the chance that a player's peer is responsible for the region they are in is greatly reduced. The direct connections made in the neighbour based method reduces communication cost. However, it is limited by the fact that messages sent between non-neighbour peers, for example to pass messages between players, can only be routed if the location of the recipient is known.

While the diversity of users in a P2P network poses additional security issues it can also be leveraged to provide resources dedicated to cheat detection. One way to do this is to form a verification group. Actions requested by peers must be received and verified by a subset of other peers, re-

Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)
0-7695-2567-9/06 $20.00 © 2006 IEEE

ducing the chance that illegal actions can be performed. The choice of the verification group thus becomes important, as it is desirable to choose peers that are not likely to collude.

We envision a direct communication model in which routing can also be performed over all peers. This facilitates the formation of groups which is independent to locality in the virtual world. Updates are passed between peers and are verified by their respective groups.

The NEO [5] protocol for peer-to-peer games, based on the work of Baughman and Levine [2], is intended to prevent a number of cheats by attempting to verify the identity of the sender and register updates in advance to ensure every party has chosen their update for a round before they may read the updates of the other players. It also attempts to ensure game state consistency by reaching agreement on which updates have been received by which peers. We analyse this protocol and show that of the five forms of cheating described in the paper, it prevents only three. We then propose an improved version of this protocol which prevents all five types of cheating, as well as meeting some additional security criteria.

The rest of this paper is organised as follows: Section 2 describes the threat model and requirements we have used in our analysis, Section 3 describes the NEO protocol, Section 4 describes the attacks we have found which compromise the NEO protocol, Section 5 describes our modified protocol SEA, Section 6 describes the advantages of SEA and Sections 7 and 8 conclude and suggest further work.

2 Threat model and requirements

For this study we assume an attacker with the ability to observe, modify, insert and delete network traffic. We also assume that the attacker may be a valid player in the game and may corrupt some number of other players. In the presence of such an attacker we cannot prevent all forms of interference but we are able to detect when tampering has occurred. Denial of service attacks are possible but given that they are also possible in the underlying network they are beyond the scope of this work.

While we do not discuss here the exact model used for the formation and coordination of the verification group, it is important to note that our threat model is not limited by any particular routing implementation. For example it may, for performance, be beneficial to construct a multicast tree within larger verification groups to distribute messages. This obviously incurs a security threat since it means messages are routed via untrusted peers. Our threat model assumes that packet manipulation is possible in the underlying network and so could occur in a point to point communication anyway.

Neither NEO nor SEA prevent players from willingly disclosing any information they possess to another player. It is easy for an attacker to directly send their updates in the clear to other players they are colluding with. It is also possible for players to disclose their private keys to other players, which has the affect of the attacker being in control of multiple players. Given that group verification is based on a majority, if a majority of players are ever corrupted they will be able to cheat the minority of honest players. It is possible to mitigate the probability of a majority of players colluding by adding random "disinterested" third parties to the group, but this is a matter for the group selection protocol.

It is important to note that we do not analyse the related protocols needed for negotiating the number of players involved, who these players are, or the round length. All of these steps have the potential to contain security critical problems and need to be treated carefully. We assume that all participants are in agreement about the membership of the group, the length of a round, and a synchronised start time. We also assume an established public key infrastructure and an honest majority of players. We expect that all players are able to synchronise their starting time and then count round lengths against a local clock. This synchronisation takes place each time a new group is formed or the membership of an existing group is changed.

It is possible in SEA that two players world views will diverge because they disagree about whether a certain update should be accepted. For example, one player may receive a majority of votes supporting a given update while another player may not, due to lost packets. This is an inevitable problem in all systems with unreliable communication channels [9]. We expect this problem to be mitigated by continuing to request other players to resend missed votes.

In the following sections we discuss both attacks and cheats. We consider an attack on a protocol to be a vulnerability in a protocol and a cheat to be a method the attacker may use to gain some benefit from a vulnerability. We use the names for the following "cheats" that are given in the original paper for clarity, but we do not consider all these to be cheats by our definition of the word.

2.1 Types of cheats

Cheating is somewhat difficult to define, but we consider it to be any action which is accepted but is against the rules of the game. Given an ideal situation with a trusted and secure server and secure links between each of the players and this server, we consider any action that is possible but would not be possible in this ideal situation to be a cheat. To prevent protocol level cheating we must ensure that every message is properly authenticated and is not modified in transit. We must detect any tampering at the protocol level which allows bad messages to be discarded quickly and should spend the least amount of time possible doing

M_A^r	Message sent by player A in round r
$M_{I(A)}^r$	Message sent by the attacker I which appears to be from A in round r
$S_A(x)$	x signed by player A
U_A^r	Update for player A for round r
$\{x\}_{K_A^r}$	Encryption of x with the key K_A^r which was chosen by player A and used in round r
V_A^{r-1}	Votes for which updates A received in round $r-1$
Vh_A^{r-1}	Votes for which updates A received in round $r-1$ including a hash of the update
x, y	Concatenation of x and y
$H(x)$	Cryptographic hash of x
n^r	A nonce (fresh unpredictable value) used in round r
$SessID$	A unique identifier for the session depends on the time and group of players in the game
ID_A	A unique identifier for player A

Table 1. Explanation of notation used in equations

so. GauthierDickey, Zappala, Lo and Marr [5] define five specific cheats that they would like to prevent.

- *Fixed-Delay Cheat* - The attacker receives packets faster than she sends them which enables her to use information about her opponents actions in the selection of her own actions (before they know her actions).
- *Time-stamp Cheat* - The attacker uses information about the actions of other players in the selection of her actions and then puts false timestamps on her updates which make it appear as if her actions occurred before theirs.
- *Suppressed Update Cheat* - The attacker does not send every update to all the players, this keeps those players she omits from knowing what she is doing.
- *Inconsistency Cheat* - The attacker sends different updates to different players.
- *Collusion Cheat* - The attacker shares information with other players she is colluding with which would not normally be available to everyone.

2.2 Types of attack

In addition to these five specific cheats we introduce two new types of attack which are well-known in the protocol literature [11]. These two types of attacks have significant potential for damaging the security goals of the protocol because they undermine the authentication of the source of a message.

- *Replay Attacks* - The attacker reuses a message previously sent by another player in whole or in part and it is accepted as valid. This can be used to duplicate the move of an opponent without necessarily knowing what the move is.
- *Spoofing Attacks* - The attacker can construct messages which appear to be from another player and are accepted as valid.

We will compare the security of the NEO protocol to our improved version, which we call Secure Event Agreement (SEA), based on the five cheats and these additional criteria.

3 Description of NEO

GauthierDickey, Zappala, Lo and Marr [5] present a protocol designed to improve on Baughman and Levine's lockstep protocol [2] by reducing latency while continuing to prevent cheating. They achieve this by adding a voting mechanism to compensate for packet loss in the environment. They call this protocol New Event Ordering (NEO). The security attributes of this protocol can be described in brief as a commit and reveal method with majority agreement on valid commitments. This is implemented by having fixed length rounds and adding a voting function such that players communicate which updates they have received by the end of the round. Players must commit to their update within the round. They do not reveal their update until the next round begins. The majority of players must receive the update within the round time limit in order for the update to be considered valid.

$$M_A^r = \{S_A(U_A^r)\}_{K_A^r}, K_A^{r-1}, S_A(V_A^{r-1}) \qquad (1)$$

The basic NEO protocol has messages of the format given in Equation 1 with notation explained in Table 1. Each player sends a message of the format described to each of the other players in their group each round. The purpose of this message is to inform the other players of the actions they would like to take in the round (this is called the update portion of the message).

4 Problems with NEO

We have discovered several possible ways to cheat which are not prevented by the NEO protocol. In brief, these are:

1. Attacker can replay updates for another player.
2. Attacker can construct messages with any previously seen votes attached. Since the votes are signed, the messages will appear to come from another player.
3. Attacker can send different updates to different opponents.

The first two of these result from the components of a NEO message being concatenated but not bound by any cryptographic means. The NEO protocol also neglects to include the identity of the originator in the message which violates Abadi and Needham's third principle for prudent engineering of cryptographic protocols [1]. We will now discuss each of these attacks in detail.

4.1 Attacker can replay updates for another player

The attacker replays a signed update from a previous round forcing the player they are attacking to make the same move as they have already made in a previous round. We refer to the honest player as A and the attacker as I in our equations. An example of a message constructed to achieve this attack is given in Equation 2. The fact that the updates are signed ensures that at the protocol level the recipient will assume that the person who signed the update sent the update, but this may not be the case.

$$M^r_{I(A)} = \{S_A(U^{r-1}_A)\}_{K^r_A}, K^{r-1}_A, S_A(V^{r-1}_A) \quad (2)$$

$$M^r_{I(A)} = \{S_A(U^{r-1}_A)\}_{K^r_I}, K^{r-1}_A, S_A(V^{r-1}_A) \quad (3)$$

In order to construct this replay attack, the attacker must have seen both the round with the update to be replayed and the following round with the key to decrypt the update. The attacker may then decrypt to extract the signed update and either re-encrypt it with an arbitrary key (as shown in Equation 3) or replay it exactly as it is since the attacker now knows the key it was encrypted with. The two example equations given differ only in that in Equation 2 the attacker reuses a key chosen by the original sender and in Equation 3 the attacker generates a new key. The attack is also illustrated in Figure 1.

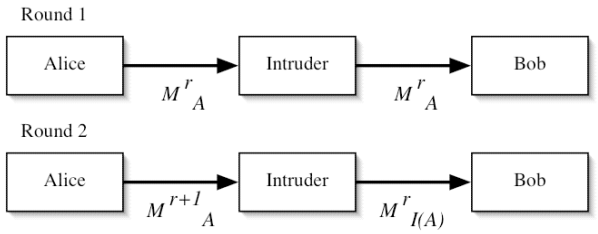

Figure 1. An illustration of the replay attack described in Section 4.1.

4.2 Attacker can spoof messages with previously seen votes

The NEO protocol has a message element which is called the votes. It is a bit vector sent by each player in which they vote either 1 or 0 for each other player in the group. A 1 means that the player did receive an update before the end of the round, a 0 means that the player voting did not receive the update in time. The purpose of the addition of voting to the protocol is to provide a way to judge consensus on whether a player sent their update within the round limit. Unfortunately, because the voting part of the message is signed by the voter but not bound to any particular round, it may be abused by an attacker to forge this consensus and break the commit and reveal nature of the protocol.

Once the attacker has recorded the votes from many other players, the attacker may then construct spoofed messages which appear to come from other players. By appending votes which accept the attacker's message, it is possible for the attacker to send her update after the round is finished and the other players will still accept it as valid. This allows the attacker to read the updates of the other players before constructing her own.

$$M^r_{I(A)} = \{S_A(U^r_A)\}_{K^r_A}, K^{r-1}_A, S_A(V^{r-4}_A) \quad (4)$$

An example of a message to achieve this attack is given in Equation 4. An illustration of this attack is given in Figure 2. The steps for this attack are:

1. I records votes from player A looking for one in which A votes "yes" for I (round $r-4$ in the example equation)
2. I constructs a spoofed packet for A and attaches the vote from $r-4$
3. The round finishes.
4. I reads the updates of all the other players for the round.

5. I constructs her own update using this information and sends it to the other players.

6. The other players mistakenly believe that A received the update before the end of the round.

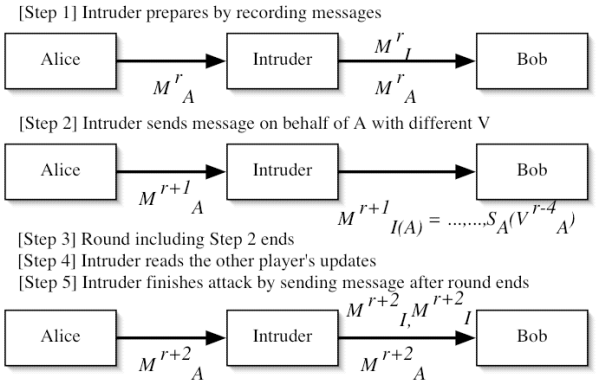

Figure 2. An illustration of the voting attack with steps corresponding to the description in Section 4.2.

This subverts the voting and defeats the purpose of the round limits. This attack will only work if I successfully spoofs packets for a majority of players (or I may have accomplices who will always vote "yes"). In relation to the original five cheats, this attack is effectively a time-stamp cheat.

4.3 Attacker can send different updates to different players

NEO provides no assurances at the protocol level that all players have received the same update. It is claimed that because the updates are signed, this behaviour will be detected, but because the updates are not tied to a round at the protocol level it is possible for the attacker to replay signed updates to different players which would convince them to accuse an innocent player of cheating. In relation to the original five cheats, this attack is effectively an inconsistency cheat.

We propose the addition of a hash of the update be included in the vote. This means that if a player A receives an update from player B within the round limit for round r, instead of voting simply "yes" for player B in round r, A will vote "yes" for an update with the hash x in round r. This allows the players to determine if a majority of players received the same update. We assume that a player should be able to construct their vote based on protocol level information alone (without passing the message up for higher level processing). We will discuss the details of our proposed protocol in the following sections.

5 Description of SEA

We propose a modification of the NEO protocol to fix the problems described in Section 4. We call our protocol Secure Event Agreement (SEA). SEA is described in Equation 5 and Equation 6. We replace encryption as a commitment method with a cryptographic hash because it provides the same security but is faster [12] and removes possible issues with key tampering and selection.

$$Commit_A^r = H(U_A^r, n^r, SessID, ID_A) \quad (5)$$

$$M_A^r = S_A(Commit_A^r, U_A^{r-1}, Vh_A^{r-1}, n^{r-1}, r) \quad (6)$$

For the sake of future analysis, it is useful to describe the intended purpose of each element of the message format. The purpose of signing the entire message is to authenticate the creator of the message. The hash serves to commit the player to the values included for the next round. We have included a nonce inside the hash to reduce the possibility of pre-computing all possible values for the hash. A nonce is a pseudo-random value which needs to be fresh (never used before, never used again) and unpredictable. The nonce may be unnecessary if the likely values for the U_A^r are from a sufficiently large range of possibilities. We have included the $SessID$ inside the hash to prevent replaying this message in a different session or with a different group of players. We have also included the ID_A inside the hash to bind the message to one particular sender, this will prevent an attacker from playing the same move (without necessarily knowing what it is) by copying the hash and playing it as their own (and supplying the appropriate update and nonce after the player reveals them in the following round). We have included the round number in the signature to prevent the entire message being replayed in a following round.

5.1 Checks to perform on a SEA message

The security assurances of a SEA message depend on the proper checks being performed before a message is accepted as valid.

- The signature is checked (we assume that players possess the public keys of the other players and that the identity to key relationship has been validated previously).

- The update and nonce for the previous round are used with the $SessID$ and ID_A (which must match the identity associated with the signature on the previous round's message) to construct the hash which must match the value committed to in the previous round.

Player	A	B	C	D	E
A	$Commit_A$		$Commit_{C1}$	$Commit_D$	$Commit_{E1}$
B		$Commit_B$	$Commit_{C1}$	$Commit_D$	$Commit_E$
C			$Commit_C$	$Commit_D$	$Commit_E$
D	$Commit_A$		$Commit_C$	$Commit_D$	$Commit_E$
E					
Tally	2	1	2	4	3

Table 2. Example voting matrix from player A's perspective for five players

- The round number, r must be for the current round.
- The V_{Player}^{r-1} are examined with the following conditions:
  - A majority of messages must have been received (based on the number of players in the group) and the result of the vote for each player must be clear. If missing messages could change the outcome, they must request that the messages be resent.
  - A majority of the hash values for a particular player must match for their update to be accepted. Only the update matching this hash value is accepted as valid.

5.2 SEA vote construction

In the NEO protocol the voting part of the message consists of a bit vector of length P, where P is the total number of players in a group. We have not fixed a maximum group size but expect the group size to vary between 5 and 30 in practise. Each player gathers these vectors and puts them together into a matrix and tallies the number of votes for each player. If the result of the tally for a particular player could not be changed by missing votes, the vote stands. If the missing votes could change the outcome of the tally (with respect to the majority), the player asks abstaining players to resend their votes.

In the SEA protocol we send the $Commit$ part of the player's update (shown in Equation 5) as the "yes" vote. This lengthens the message but provides additional security. Given a game with 5 players, A, B, C, D, and E, an example voting matrix from the perspective of player A is given in Table 2. In this example all players would immediately discard the update for B, and accept the updates for D and E given that a majority of votes agree on each of these players' updates. For players A and C however, the result is affected by E's missing vote. The update received by players A and B from player C, $Commit_{C1}$ differs from the update received by players C and D, $Commit_C$. It is necessary to contact E to resolve the issue. If a response from E does not arrive in a timely manner, these updates should also be discarded.

6 Advantages of SEA

The SEA protocol offers security advantages and equal or better performance to the NEO protocol.

6.1 Five original cheats

We have already shown that the NEO protocol did not prevent all five of the cheats it was designed to stop. Our SEA protocol does prevent all five of these cheats which are described in Section 2.1.

- *Fixed-Delay Cheat* - Forcing all players to commit to a move in a particular round before revealing any moves for that round prevents the attacker from gaining any knowledge about the other players' moves before they must choose their own. A hash function is a secure method of accomplishing this goal. The voting protocol ensures that a majority of players have received the same commitment before the round finishes.
- *Time-stamp Cheat* - All updates sent in the same round happen simultaneously. The round numbers are used to order events appropriately. If an update was not received within the round duration with the correct r, it will not be accepted.
- *Suppressed Update Cheat* - The $Commit$ value included in the voting part of the message will let any players who did not receive an update from a particular player know what update that player committed to without receiving it directly. Since all messages are signed, they may be forwarded on to players that did not receive them. This means that a player only needs to get voting information from one other player that did receive the update.
- *Inconsistency Cheat* - As in the above suppressed update cheat, the $Commit$ value included in the votes makes it possible for the players to detect if they have

Primitive type	*Example primitive*	*Clock cycles*
Hash function	SHA-1 (160 bits)	15/byte + 1040
Symmetric encryption	AES (128 bit key)	25/byte + 504
Digital signature	RSA-PSS (1024 bit key)	42,000,000

Table 3. Approx. CPU cycles used on a Pentium III Processor [10]

received different updates and only if a majority of players have received the same $Commit$ will it be accepted.

- *Collusion Cheat* - Players may still share any information they possess with other players outside the typical game channels but given the commit and reveal method, this information is limited to revealed updates.

6.2 Additional security criteria

We have added some additional security requirements which we feel are important to the security of a peer-to-peer game protocol.

- *Replay Attacks* - The $Commit$ value includes the $SessID$ and ID_{Player} which should prevent another player from replaying the message in a different context. The $Commit$ value also includes a nonce n_r to provide freshness. If this nonce is unique (and from a large enough possible range of values), it should make the $Commit$ value both unpredictable and unique.

- *Spoofing Attacks* - In SEA the entire message is signed which should prevent another player from generating spoofed messages with valid signatures (provided the key is kept secret).

6.3 Performance

To form a message, SEA uses one hash operation and one signature operation. This is faster than NEO which uses one encryption operation and two signature operations. To give the reader a sense of how expensive each type of operation is to compute, the approximate costs of, and an example of, each type of operation are given in Table 3 [10]. This shows that the cost of a signature operation is so many times greater than either a hash function or encryption operation that it is the most significant part of the computation. Therefore the reduction from two signature operations to one will have the most significant impact on the performance. We have chosen the particular hash function, symmetric encryption and signature examples given in Table 3 because they are well-known, there is no requirement to use these particular functions in an implementation of SEA and these are not necessarily the best choices to make for these functions.

In the NEO protocol, the loss of a message results in the loss of two rounds of updates due to the chaining nature of the messages. In our SEA protocol this is mitigated by the addition of the $Commit$ values to the voting section of the message. This allows a player to accept an update even if they did not receive the $Commit$ on time or at all (provided that a majority of other players did). A player who is missing a $Commit$ may randomly select any other player who has received the update (and voted as such) and request that they resend the missing value.

We also propose a variation which adds to the message length but reduces the effect of lost messages as shown in Equations 7 and 8. By including updates for the two previous rounds, the loss of a message results in the loss of only the round that is committed to in that message (or possibly none at all when considered in conjunction with the voting format discussed above).

$$Reveal_A^r = U_A^{r-1}, Vh_A^r - 1, n^{r-1}, r \quad (7)$$

$$M_A^r = S_A(Commit_A^r, Reveal_A^r, Reveal_A^{r-1}) \quad (8)$$

7 Conclusion

We have shown that the NEO protocol fails to prevent two of the five attacks its authors claim. These attacks are possible because the elements of the message are simply concatenated. This allows the formation of new illegal but valid messages from previously seen events. We have proposed a new protocol, SEA, which addresses these issues. The SEA protocol signs an entire event message, binding the parts of the message into a whole. It also includes additional information such as the $SessID$, n_r, and r which bind the message to a particular round and group.

In addition to being significantly more secure, the SEA protocol offers performance advantages. The SEA protocol is shown to require approximately half the computation to create and process each message. The addition of update hashes to the voting section of the message adds the ability to securely reconstruct and propagate updates which have not reached all relevant players. This means that instead of losing two communication rounds with each missing message, as is the case for NEO, the SEA protocol only loses

one, effectively giving twice the delivered updates upon failure.

The SEA protocol is a strong foundation upon which a comprehensive secure peer-to-peer game may be built. A complete solution has many layers and each layer must ensure adequate security. SEA provides this security for the protocol level.

8 Future work

Given that the group selection and round negotiation protocols are crucial to the security of any group consensus event agreement protocol, we intend to develop secure group selection and round negotiation protocols to work in conjunction with SEA. It is important that all members of a group agree on the membership of the group for concepts like "majority" to make sense. This is particularly challenging in a peer-to-peer game environment where the group membership may change frequently as players move around the virtual world or enter and exit and the game altogether.

Acknowledgment

The authors would like to thank Aaron Harwood for helpful discussions and comments on this paper. We would also like to thank NICTA Victoria research lab for supporting this work.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[2] N. E. Baughman and B. N. Levine. Cheat-proof playout of centralized and distributed online games. In *Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications*, pages 104–113, Apr. 2001.

[3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *ACM SIGCOMM*, pages 353–366, Portland, OR, September 2004.

[4] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for Internet games. In *NetGames*, pages 3–9, Bruanschweig, Germany, April 2002.

[5] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr. Low-latency cheat-proof event ordering for peer-to-peer games. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2004.

[6] S. Hu and G. Liao. Scalable peer-to-peer networked virtual environment. In *NetGames*, pages 129–133, Portland, OR, September 2004.

[7] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *PDPTA 2003*, pages 262–268, Las Vegas, NV, June 2003.

[8] B. Knutsson, H. Lu., W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *IEEE Infocom*, Hong Kong, China, March 2004.

[9] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[10] B. Preneel, B. V. Rompay, S. B. Ors, A. Biryukov, L. Granboulan, E. Dottax, M. Dichtl, M. Schafheutle, P. Serf, S. Pyka, E. Biham, E. Barkan, O. Dunkelman, J. Stolin, M. Ciet, J.-J. Quisquater, F. Sica, H. Raddum, and M. Parker. Performance of optimized implementations of the NESSIE primtives, February 2003.

[11] B. Schneier. *Applied Cryptography (2nd Edition)*. John Wiley & Sons, 1996.

[12] B. Schneier and D. Whiting. A performance comparison of the five AES finalists. In *The Third Advanced Encryption Standard Candidate Conference*, pages 123–135, New York, NY, USA, April 13–14 2000.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.