Probabilistic Event Resolution with the Pairwise Random Protocol

John L. Miller
University of Cambridge
Computer Laboratory
+44 1223 123456
jlm60@cl.cam.ac.uk

Jon Crowcroft
University of Cambridge
Computer Laboratory
+44 1223 123456
Jac22@cl.cam.ac.uk

ABSTRACT
Peer-to-peer distributed virtual environments (DVE's) distribute state tracking and state transitions. Many DVE's - such as online games - require ways to fairly determine the outcome of probabilistic events. While trivial when a trusted third party is involved, resolving these actions fairly between adversaries without a trusted third party is much more difficult. This paper proposes the Pairwise Random Protocol (PRP), which uses secure coin flipping to enable adversaries to fairly determine the result of a probabilistic event without a trusted third party. Three different variations of PRP are presented, and the time impact and network overhead are examined. We conclude that PRP enables DVE's to distribute the work of determining probabilistic events between adversaries without loss of security or fairness, and with acceptable overhead.

Categories and Subject Descriptors
I.6.8 [**Simulation and Modeling**]: Types of Simulation – *gaming.*

General Terms
Algorithms, Security.

Keywords
Distributed Virtual Environment, Security, Bit Commitment, Secure Coin Flipping, Fairness, Cheating, Pairwise Random Protocol.

1. INRODUCTION

Distributed Virtual Environments (DVE's) are virtual environment (VE) simulations run on two or more nodes. Nodes are defined as individual software instances contributing to the DVE, usually running on separate computers connected by a network. DVE's are used for a variety of purposes, such as military simulations [1], immersive educational and therapeutic environments[2], cyberspace virtual environments[3], and networked computer games. Blizzard Entertainment's World of Warcraft[4], for example, is a DVE with more than eleven million paying subscribers[5], and more than a million active nodes at its busiest times.

Conference'04, Month 1–2, 2004, City, State, Country.

Virtual Environments are implemented as DVE's to allow more resources to be applied to the simulation, ideally providing better scalability and higher simulation resolution than possible with a fully centralized simulation. DVE's usually follow one of two models: client-server or peer-to-peer. Client-server and DVE's perform important operations on trusted nodes, and so can typically trust state representation and state transition calculations. Peer-to-peer DVE's, however, distribute more of the state-keeping and transition work to untrusted nodes, requiring additional steps to secure the DVE.

Several solutions have been proposed to facilitate fair resolution of competition between participants in peer-to-peer DVE's. Solutions which address resolution of conflict between peers typically either focus on event ordering rather than supporting probabilistic transactions, or rely upon quorums or disinterested third parties to take on the role of trusted third parties, sidestepping the problem. While useful, approaches which proxy trusted third parties can't guarantee fairness. Quorums can be subverted, and arbitrary 'disinterested' third parties can be malicious for the sake of being malicious, whether or not they know their victim. How, then, can two adversaries interacting in a DVE – for example engaged in combat in a military DVE - determine whether a probabilistic event such as an attack succeeds or fails when both parties are incented to cheat?

This paper outlines a pairwise random protocol (PRP) for untrusted nodes to fairly generate random bit sequences which can be used to resolve probabilistic events. PRP allows adversaries to fairly resolve sequences of actions without requiring intervention from a third party, trusted or otherwise.

The remainder of this paper presents PRP and analyzes its benefits. Section 2 provides a brief overview of DVE security research, and the foundation of bit commitment and secure coin flipping. Section 3 presents two variations of PRP. Section 4 discusses PRP's attributes and performance compared to a trusted third party (TTP). Section 5 presents a final summary.

2. RELATED WORK

Relevant related work falls into two categories: DVE security research, and secure coin flipping. DVE security research covers a variety of different aspects of DVE correctness, but doesn't generally address fair resolution of probabilistic events without a trusted third party. Secure coin flipping is a well-known cryptographic technique for resolving probabilistic events between adversaries.

2.1 DVE Security research

Distributed Virtual Environments distribute simulation work across two or more nodes. Distributing simulation rule enforcement opens the DVE to exploitation by participants wishing to bias the simulation. In online game DVE's, this exploitation is usually defined as cheating.

Known cheats have been analyzed and categorized in a variety of ways. Yan and Randell provide a useful cheat taxonomy with examples in [6]. Webb and Soh present an interesting overview of cheating and their own taxonomy in [7]. Yee et. al. present a threat model for MMOG DVE's in [8]

A variety of approaches have been proposed to combat cheating in game DVE's. Some - such as Mönch's work on mobile guards[9] - suggest protecting binaries and network transmissions from modification as the primary defense. Others[10] argue that relying on such protections alone is akin to participating in an arms race with the cheaters, one there is no hope of the DVE authors winning.

Most proposals target a specific type of cheat. For example, event reordering and update-suppress cheats can be addressed by lockstep protocols such as asynchronous synchronization[11], NEO[12], and trusted timestamp servers[13].

Auditing can be used to detect and deter cheating[14], though it requires a trusted or semi-trusted auditor with sufficient resources to validate suspect behavior.

Deferring state tracking and transition to a disinterested third party or a quorum has been proposed several times[15][16][17]. While sound in general, ensuring quorum members or disinterested third parties aren't aligned with the interests of one of the affected parties – or simply malicious – is problematic.

Finally, Knutsson et al. provide a thorough proposal for a peer-to-peer DVE in their 2004 SimMud[18] paper. They call out the need for interacting nodes to have a reliable, verifiable stream of random numbers for resolving their interactions - for example by sharing a seed for a random number generator. However, their paper doesn't specify how this seed should be generated, and relying on a deterministic sequence based on a random seed introduces weaknesses, as discussed later in this paper.

Each of these techniques has merit and mitigates real threats. However, none of them enable fair resolution of probabilistic events.

2.2 Secure coin flipping

The pairwise random protocol is a variation of secure coin flipping. Secure coin flipping was first introduced by Blum in [19]. In essence, Blum proposes using a secure one-way function $F(x)$ to enable *Alice* and *Bob* to verifiably flip a fair coin, even though they are adversaries. In the simplest case, this is a three step process, where Bob tries to guess if a bit sequence R chosen by Alice is even or odd. If he's correct, he wins the coin toss. Otherwise he loses.

1. *Alice* chooses a bit vector x, then tells Bob $F(x)$.
2. *Bob* tells *Alice* his guess as to whether x is even or odd.
3. *Alice* reveals x to Bob.

At the end of the exchange, Bob can calculate $F(x)$ to ensure *Alice* didn't change x after learning *Bob*'s guess.

We consider the (currently) unbroken hash function SHA-256 a suitable secure one-way function for our implementation of secure coin flipping.

3. PAIRWISE RANDOM PROTOCOL (PRP)

The pairwise random protocol (PRP) provides a way for two competing nodes in a DVE to fairly resolve probabilistic events.

Consider a DVE with nodes, *Alice* and *Bob*. Each node controls an avatar, and those avatars are interacting. Given a consistent, verifiable view of the simulation state, we wish to enable *Alice* and *Bob* to fairly resolve a set of probabilistic events. For example, *Alice* and *Bob* are engaged in combat, with a certain probability of each successfully attacking their opponent, and a variable amount of damage inflicted per successful attack.

Each node is incented to cheat to resolve actions in their favor. *Alice* wants all of her attacks to succeed, and all of *Bob*'s attacks to fail. *Alice* wants each of her hits to inflict maximum damage, and each of *Bob*'s hits - should he manage to get any - to inflict minimum damage. PRP ensures that - given consistent views of world state - *Alice* and *Bob* can fairly resolve probabilistic interactions such as determining attack success and selecting the amount of damage inflicted within the specified range.

As *Alice* and *Bob* are participating in the same DVE, we can make some simplifying assumptions.

1. *Alice* and *Bob* each know the correct DVE rules. Even if *Alice* is running a modified version of the DVE software, she has the unmodified code at her disposal for verifying validity of *Bob*'s activities.
2. *Alice* and *Bob* have access to identical pseudo-random number generators, and these generators provide 'suitably random' sequences for the DVE to resolve probabilistic sequences of activities.
3. *Alice* and *Bob* can communicate with each other.

Given these assumptions, we describe any probabilistic activity which affects either party as an adversarial activity.

Before resolving the success or failure of an adversarial activity, *Alice* and *Bob* must specify the activity to be decided. For example, *Alice* and *Bob* must agree that they are performing PRP to calculate whether or not *Alice* succeeds in attacking *Bob*. This has two benefits:

1. It ensures that the losing party in a PRP exchange can't claim the exchange was intended to determine outcome of a different activity, e.g. whether *Alice* gets crumbs on her jacket from eating a donut, rather than success in combat.
2. It allows a cryptographic proof of participation in the activity to be generated. This reduces the utility of the loser refusing to continue the exchange.

We assume this binding can preface the PRP exchange, or be performed as part of it. Discussions of methods for doing this are out of scope of this paper.

Section 3.1 describes the core PRP protocol to resolve a single probabilistic event. Section 3.2 proposes a refinement for generating a pseudo-random sequence without either adversary controlling the sequence.

3.1 Resolving a single action

Probabilistic actions can be resolved by a series of secure coin flips with a pre-agreed interpretation. For example, *Alice* and *Bob* can agree that *Alice* has a 5 in 8 chance of successfully attacking *Bob*. *Alice* therefore needs to generate a random number between 1 and 8, and if it is 5 or less, her attack succeeds. *Alice* and *Bob* can generate this number by flipping a fair coin three times to generate a 3-digit binary number, with heads being a '1' and tails a '0'. As long as we can guarantee sequencing of flip results used as bits, resolving a single arbitrarily scaled probabilistic event – such as this one - can be reduced to ensuring a single coin can be fairly flipped.

The basic protocol for *Alice* and *Bob* to determine a random bit without requiring a trusted third party is described below, and illustrated in Figure 1. Note this exchange is roughly equivalent to Blum's secure coin flip protocol[19].

1. *Alice* and *Bob* each privately choose a bit vector of length 1, B_A and B_B respectively.
2. *Alice* generates a (possibly zero-length) nonce N_A known only to her, and uses a cryptographic hash $H(x)$ to generate a digest $D = H(N_A, B_A)$. She sends D to *Bob*.
3. *Bob* makes a note of *Alice*'s digest D, and sends his bit vector B_B to *Alice*.
4. Upon receipt of *Bob*'s bit vector, *Alice* transmits her nonce N_A and bit vector B_A to *Bob*. *Bob* verifies that the hash of these values $H(N_A, B_A)$ matches the previously received digest D.
5. *Alice* and *Bob* XOR their own bit vector with their adversary's bit vector to determine the outcome of the exchange. In the case of a single-bit bit vector, if $B_A = B_B$ then the result is 0. Otherwise it is 1.

As long as each message is eventually received, *Alice* and *Bob* can be assured that the binary result is fairly determined. It doesn't matter whether *Alice* and *Bob* randomly or deliberately select their bit vectors. As long as *Alice* and *Bob* are not collaborating, there is a 50% chance of the bit being 1, and a 50% chance it is 0.

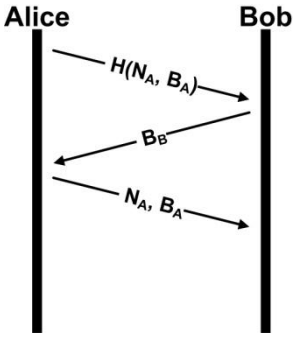

Figure 1 - Single Bit PRP Exchange

Barring retransmissions, a minimum of three messages comprising one-and-a-half round trips are required to complete a single PRP exchange, as shown in Figure 1. If low latency is more important than a low message count, latency can be reduced to a single round trip by adding a message and making the exchange symmetric, as shown in Figure 2.

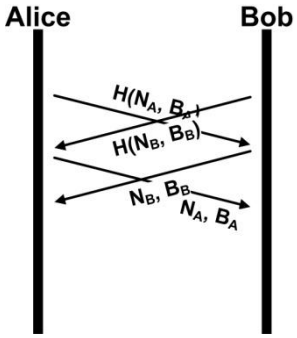

Figure 2 - Symmetric Single Bit PRP Exchange

This protocol can be trivially extended to provide an arbitrarily long random bit vector by changing the number of bits in B_A and B_B. For example, rather than performing three sets of exchanges for *Alice* to generate her three-bit random number, she can simply replace B_A with a 3-bit bit vector, and instruct Bob to do the same with B_B.

This version of PRP is secure, but requires several network messages for each random value provided. Depending upon the security requirements of the DVE, it is possible to obtain acceptable results with less overhead, as detailed below.

3.2 Resolving an unbounded random sequence

Interactions in DVE's are often comprised of long sequences of actions. Requiring a three or four message exchange for each action by each participant is secure and fair, but slow and expensive.

An alternative is to resolve more bits than are required for the current event, and to use the next sequence of unused bits for each subsequent activity. While efficient from a protocol perspective, this extension suffers from a look-ahead vulnerability in terms of consumption. Once *Alice* and *Bob* finish the exchange and determine the bit sequence, neither can change the bits. However, they can modify their behavior to consume the bits in an advantageous way.

For example, suppose *Alice* can execute any of four actions interchangeably: she can tie her shoes (random chance of failure), skip a rock (random number of skips), pick a flower (random length of stem), or build a house (random number of rooms). Each action has a different cost and benefit for *Alice*. If *Alice* knows the sequence of bits which will be consumed to determine the outcome of her probabilistic actions, she can 'look ahead' to determine the most favorable sequence to execute. For example, she can pick flowers to consume undesirable bits, waiting to build a house until the next set of bits guarantee she builds a house with the maximum number of rooms.

Another alternative to provide random values for a series of activities is to use PRP to determine a random seed for a pseudo-random generator. *Alice* and *Bob* agree on the use for a pseudo-random stream, then use PRP to create a bit vector of an appropriate size to seed the generator. Since both Alice and Bob have copies of the random number generator, they can each validate the sequence generated using the resolved bits seed, and the subsequent results. Note that the idea of using a pseudo-

random generator to create a sequence of random numbers which can be verified by all participants is suggested in [18].

4. RESULTS AND DISCUSSION

PRP as described in Section 3.1 provides a reliable but relatively expensive source of bits to fairly resolve adversarial probabilistic events. Section 3.2 describes a less expensive variant of PRP, but at the cost of enabling look-ahead cheats, and allowing participants to unfairly optimize the order of events which consume those bits.

DVE authors should carefully examine impact of look-ahead exploitation before using the random seed or pre-generation approaches to generating bit sequences. Real-time interactive DVE's such as network games may be so dynamic that the look-ahead vulnerability is of no practical concern, especially if the bit stream is refreshed every few seconds. For example, *Alice* may have only a small number of action choices at any given time, and attempting to bias her choice according to attributes of the random bit stream may provide less value – even when done via an automatic enhancement hack - than selecting the most appropriate action at the time.

While PRP performance overhead is greater than the overhead of working directly with a TTP, the cause is not solely the algorithm itself. Distributing activities normally performed by a TTP to unreliable, untrusted nodes can introduce significant overheads to DVE activities, as noted in several of the previously cited DVE security works. Still, informed choices can minimized this overhead. We provide performance analysis below to help DVE authors understand trade-offs in different PRP usage scenarios.

DVE's often rely upon congruent random generators running on a TTP to determine the outcome of probabilistic actions. For example, Quake-3 uses calls to their random generator to determine variations in projectile direction due to weapon recoil. The random number generator typically used in deployed network games is from the 'C' standard library, and typically provides a two byte random number.

In a game with trusted third parties, the TTP can often produce and consume random numbers locally on behalf of a given client. Alternatively, it can provide the random number to the client for the client to consume in its local activities. In peer-to-peer DVE's, the system should not rely upon a TTP for common activities. The client should ideally be able to resolve probabilistic events without TTP intervention. For the DVE to be fair, this must be done in a way which doesn't allow the value to be chosen by the same client consuming it.

The following two subsections discuss PRP's security and performance properties, using this scenario to illustrate those properties.

4.1 Security

Most deployed DVE's are implemented as client-server applications. From the client perspective, the server acts as a trusted third party (TTP). The server is explicitly trusted to fairly resolve probabilistic events on behalf of its clients. In other words, over the course of many trials, the client expects the distribution of results to roughly match the probability of each outcome.

Even in the TTP case, outcome can be biased by many factors, such as the source of random numbers for event resolution. We do not propose to discuss methods for generating suitably random numbers here. Instead, our goal is to support the proposition that given two adversaries, neither adversary can predictably bias the resulting random bit vector. As long as appropriate precautions are taken, this should provide probabilistic event resolution of a quality no worse than that available from a TTP.

Let *Alice* be a node undertaking PRP to create a random bit vector for her consumption, and *Bob* an adversary participating in that PRP exchange. We wish to prove that so long as *Alice* and *Bob* cannot predict the value of their adversary's bit vector B, neither can bias the result of the PRP exchange. To do this, we need to prove four properties:

- P1. Once *Alice* commits to a choice for B_A by transmitting a digest D to *Bob*, she cannot change her choice without detection.
- P2. *Bob* cannot ascertain *Alice*'s choice of B_A from the digest D.
- P3. If *Bob* has no knowledge of *Alice*'s choice for B_A, then *Bob* cannot choose a B_B which will bias the result.
- P4. Given the sequence of messages exchanged in PRP, neither *Alice* nor *Bob* can dispute the value of the resulting bit vector $B_A \oplus B_B$.

P1: Alice transmits the digest D, a SHA-256 hash of an input of length at least 256 bits. In this, case input is a 255-bit nonce N_A and Alice's 1-bit bit vector B_A. In order for *Alice* to change her selection of B_A to B'_A after transmitting D to *Bob*, she must find a new nonce N'_A such that $SHA_{256}(N'_A, B'_A) = SHA_{256}(N_A, B_A)$. Since SHA_{256} isn't broken, this would require a brute force attack, on average $2^{256}/2$ attempts, which means trying every value for the 255-bit nonce. This is computationally infeasible. Even if every computer on earth were employed and each was capable of testing a million candidates per second, more than 10^{56} years would be required. Alternatively *Alice* could try to find two 256-bit vectors with a different last bit whose hashes collide, but even this would require $\boldsymbol{O}(\sqrt{2^{256}}) = \boldsymbol{O}(2^{128})$ attempts.

P2. Since SHA-256 is an unbroken cryptographic one-way function, and since *Alice* has given it an input of at least 256 bits, there is no way for *Bob* to predict the value of the input solely based upon its output, or to limit that input to a specific candidate pool other than brute-force attack, which as shown above is computationally infeasible.

P3. For each bit in B_B, a 1 will invert *Alice*'s choice for the same bit in the result bit vector, while a 0 will leave *Alice*'s choice intact. Since *Bob* cannot determine the bit chosen by Alice for each position in B_A at the time he must commit to B_B, he has no way of choosing a value for B_B to maximize chances of a specific outcome.

P4. For a given input, XOR is a deterministic operation, so $B_A \oplus B_B$ is deterministic for given B_A and B_B. By the time $B_A \oplus B_B$ can be calculated by either *Alice* or *Bob*, both are committed to their bit vector values, and cannot change that commitment without detection from their adversary.

Like most protocols, PRP in its basic form is vulnerable to abort attacks, e.g. *Bob* refusing to acknowledge receipt of Alice's final PRP message after he determines the resulting bit vector does not yield his desired outcome. This can be mitigated by standard cryptographic techniques such as signing each message in the

protocol, and using anti-replay and sequencing protections to prove message sequence order and contents.

4.2 Performance

Suppose *Alice* wishes to generate a single 16-bit random number for consumption for a pre-agreed purpose. Suppose as well that *Alice* has 100 ms RTT to *Bob* on the network. Table 1 compares the network latency and traffic required for Alice to obtain this random bit vector. We assume IPv4 UDP on Ethernet as the transport medium, inducing transport overhead of 42 bytes per packet. PRP uses SHA-256 as the one-way hash, and a nonce size equal to (hash length – target bit vector size) for bit vectors smaller than the hash value size.

PRP requires one and a half round trips, with each packet containing 42 bytes of headers. The first packet contains the SHA-256 hash of *Alice*'s 240-bit nonce and 16-bit bit vector. The second packet contains *Bob*'s 16-bit bit vector. The final packet contains *Alice*'s nonce and bit vector, and completes the PRP protocol transaction.

Table 1 - Random number generation cost

	TTP	Adversary	Additional cost
Latency	50 ms	150 ms	300%
Network Bytes	44 bytes	192 bytes	436%

The cost in terms of latency and network bytes for the PRP protocol version described in section 3.1 is significant compared to obtaining the random bit vector directly from a TTP. Fortunately there are a few ways we can decrease overhead without realistically compromising security.

First, we can reduce the number of bytes transmitted in payloads by reducing the size of the transmitted hash, and of the nonce itself. PRP uses SHA-256 because SHA-256 is not yet broken, rather than because 256 bits of protection are required. Secrets in a PRP exchange are short-lived – less than a second in the example above – so the hash value only requires enough bits to prevent an attacker from determining *Alice*'s bit vector before it is revealed in *Alice*'s second message. The most significant threat is a dictionary attack, because of its short execution time.

In the 16-bit bit vector case, it would be trivial for *Bob* to create a dictionary with the SHA-256 hash values for the 2^{16} possible values for *Alice*'s bit vector. To prevent this, we include a large nonce in the hash to make lookup impractical for *Bob.* We can establish a size of lookup table we wish to defeat – for example one petabyte – and choose a hash and nonce size to enable that level of protection. A petabyte is approximately 2^{58} bits. Each entry in a sparse lookup table would include the lookup hash value and the expected 16-bit bit vector. If the hash is truncated to 64-bits, then each lookup table entry would consume 80 bits, resulting in a table capacity of about 2^{52} entries. With a nonce size of 48 bits (and a 16-bit bit vector), this would give *Bob* a probability of about $2^{52} / 2^{64}$ = 0.02% chance of successfully looking up *Alice*'s bit vector from the hash in her first PRP packet with a petabyte index. This optimization reduces the network bytes required for a 16-bit bit vector PRP exchange from 192 bytes to 144 bytes, dropping the network cost from 436% of TTP transaction cost to 327%.

Another way to improve both latency and network overhead associated with PRP – though at the cost of some security - is to pre-calculate a large bit vector for consumption, and then use successive parts of that vector for the next *k* random contests. For example, suppose Alice needs an average of ten 16-bit random values per second. She can request a bit vector with enough bits to satisfy five seconds of her requirements, or $50 * 16 = 800$ random bits. For a request this large, assuming a sufficiently random input bit vector on Alice's side, a nonce is no longer needed. Precalculating a series of random values amortizes PRP latency PRP across several seconds of bit consumption, reducing its effective performance impact. It also reduces the relative overhead of generating the 800 random bits. Total PRP network byte cost – assuming 64-bit truncated hash - is 340 bytes, which compared with a TTP-sent packet size of 44+(800/8) = 144 bytes is 236% more, less overhead than our previous optimizations.

A slightly weaker choice would be to use PRP to create the 32-bit seed for *Alice*'s DVE random number generator, and have *Alice* use the resulting pseudo-random sequence for a set interval or number of operations. This approach would consume 46 bytes to obtain the seed from a TTP, or 146 bytes using PRP. While the relative overhead in this case is still more than 300% greater than obtaining the seed from a TTP, the absolute cost to the DVE for generating e.g. 800 pseudo-random bits is quite low.

5. CONCLUSIONS

This paper presented the Pairwise Random Protocol (PRP), based on secure coin flipping. Using PRP, adversaries can fairly determine and agree upon the outcome of probabilistic actions. Three different variations of PRP were presented, along with high-level performance analysis of the algorithms. The variations range from a perfectly fair approach which requires a three-way handshake per random event, to creating arbitrarily long pseudo-random sequences using a fairly determined random seed, up to the tolerance of the DVE.

PRP makes it possible for adversaries to fairly determine the results of probabilistic events in a DVE with the same security a trusted third party – such as a game server – could provide. For DVE's which do not frequently need random numbers, or which are tolerant of the 2 to 4 times overhead required for the most secure versions of PRP, this can be done without loss of fairness or security. If the DVE is performance-sensitive, then compromises can be used such as pre-generating a set of random bits to use over time, or seeding a random number generator, which allow reasonable security without significant performance impact.

For future work we hope to integrate PRP into a peer-to-peer adaptation of a deployed game, such as Quake III which is available in open-source form.

6. REFERENCES

1 IEEE standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1-1995* (1996), -.

2 Pagdin, Frances A. and Taylor, Ian C. Virtual Reality - a new therapeutic medium (2001).

3 Research, Inc. Linden. *Second Life: Official site of the 3D online virtual world*.

4 Entertainment, Blizzard. *World of Warcraft Community Site*.

5 Entertainment, Blizzard. World of Warcraft Surpasses 11 Million Subscribers Worldwide. *Press Release* (2008).

6 Yan, Jeff and Randell, Brian. A systematic classification of cheating in online games. (2005), ACM Press, 1-9.

7 Webb, Steven Daniel and Soh, Sieteng. Cheating in networked computer games: a review. (2007), ACM, 105-112.

8 Yee, George, Korba, Larry, Song, Ronggong, and Chen, Ying-Chieh. Towards Designing Secure Online Games. (2006), IEEE Computer Society, 44-48.

9 Mönch, Christian, Grimen, Gisle, and Midtstraum, Roger. Protecting online games against cheating. (2006), ACM Press, 20.

10 Pritchard, Matt. How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It. *Gamasutra* (2000).

11 Baughman, Nathaniel E., Liberatore, Mark, and Levine, Brian Neil. Cheat-proof Playout for Centralized and Peer-to-Peer Gaming. *IEEE/ACM Transactions on Networking*, 15 (2006), 1-13.

12 GauthierDickey, Chris, Zappala, Daniel, Lo, Virginia, and Marr, James. Low latency and cheat-proof event ordering for peer-to-peer games. (2004), ACM Press, 134-139.

13 Shusuke, Tatsuhiro Yonekura. Time-Stamp Service makes Real-Time Gaming Cheat-Free. (2007), ACM Press, 135-138.

14 Fung, Yeung Siu. Hack-proof synchronization protocol for multi-player online games. (2006), ACM Press, 47.

15 Cecin, F. R., Real, R., Oliveira, R. de, Resin, C. F., Martins, M. G., and Victoria, J. L. A Scalable and Cheat-Resistant Distribution Model for Internet Games. (2004), IEEE, 83-90.

16 Fan, Lu, Taylor, Hamish, and Trinder, Phil. Mediator: A Design Framework for P2P MMOGs. (2007), ACM Press.

17 Kabus, Patric, Terpstra, Wesley W., Cilia, Mariano, and Buchmann, Alejandro P. Addressing cheating in distributed MMOGs. (2005), ACM Press, 1-6.

18 Knutsson, Björn, Lu, Honghui, Xu, Wei, and Hopkins, Bryan. Peer-to-Peer Support for Massively Multiplayer Games. (2004), IEEE.

19 Blum, Manuel. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15 (1983), 23-27.