Multimedia Systems
DOI 10.1007/s00530-012-0291-z

Match+Guardian: A Secure Peer-to-Peer Trading Card Game Protocol

Daniel Pittman · Chris GauthierDickey

Received: 30 Nov 2011 / Accepted: 5 June 2012
Original publication available at: `http://www.springerlink.com`

Abstract In trading card games (TCGs), players create a deck of cards from a subset of all cards in the game to compete with other players. While online TCGs currently exist, these typically rely on a client/server architecture and require clients to be connected to the server at all times. Instead, we propose, analyze and evaluate Match+Guardian (M+G), our secure *peer-to-peer* protocol for implementing online trading card games. We break down actions common to all TCGs and explain how they can be executed between two players without the need for a third party referee (which usually requires an unbiased server). In each action, the player is either prevented from cheating or if they do cheat, the opponent will be able to prove they have done so. We show these methods are secure and may be shuffled into new styles of TCGs. We then measure moves in a real trading card game to compare to our implementation of M+G and conclude with an evaluation of its performance on the Android^{TM} platform, demonstrating that M+G can be used in a peer-to-peer fashion on mobile devices.

Keywords Games · Peer-to-peer network protocols · Distributed systems

1 Introduction

Trading card games (TCGs), also known as collectible card games, are a type of card game where players purchase, collect, or trade cards, allowing them to create a playing deck from their collection which can be used to compete with other players. Cards in the game have different features or abilities and may be used strategically and in conjunction with other cards in their deck. Recently, TCGs have started to move to the computer game realm and typically use client/server architectures primarily because a centralized system is needed for handling game transactions and because servers can act as a referee for game play, thereby preventing players from cheating. The question naturally arises: can TCGs be played without cheating in a purely peer-to-peer fashion? We present a protocol showing that this is indeed possible.

In designing a protocol for online TCGs, player expectations must be considered. Two players competing against each other typically have several expectations. First, they expect to be able to play their TCG wherever and whenever they want. One can imagine starting up a game on a bus or train while commuting, but while being disconnected from the internet. Players should be able to create an ad-hoc network and play the TCG. Second, they expect that games which other people play do not affect their ability to play. Thus, Alice and Bob's game should not be adversely affected by any other game currently being played. Further, Alice and Bob should be able to play when they desire, without having to worry about overloaded servers. Third, they expect that the other player cannot easily cheat without being caught. Last, they expect to be able to play *instant*-type cards, which are cards played in response to an opponent's card(s) and are usually played in rapid succession between players.

From these requirements, we have designed a peer-to-peer protocol called Match+Guardian which:

1. Allows players to play with ad-hoc network connections since they do not rely on internet connectivity to a server;

D. Pittman · C. GauthierDickey
University of Denver
Department of Computer Science
Denver, CO, USA
E-mail: dpittman@cs.du.edu

C. GauthierDickey
E-mail: chrisg@cs.du.edu

2. Is extremely scalable since servers are not required to arbitrate the games;
3. Either prevents an opponent from cheating in the game or at minimum detects and provably shows the opponent has cheated;
4. Provides low-latency communication, which facilitates TCGs with *instant*-type cards.

Peer-to-peer protocols typically introduce the problem of cheating when used with games and further due to network address translation (NAT), peers may have difficulty connecting to each other. Modern trading card games may in fact consist of several styles of play, including sealed-deck tournament games, draft games, and constructed deck games. We decompose these types of games into a primary sets of actions: from randomly and fairly generating decks of cards, to ensuring that any card played came from a deck that was fixed prior to when play commenced. We then show how to securely perform these steps in a peer-to-peer manner so that all styles of play can be supported. While NAT issues with peers are beyond the scope of this paper, we discuss work in this area in Section 2.

One might assume that the problems in TCGs are exactly those in the game of *mental poker* [12], a fictional game where two people can play poker without seeing each other's cards (e.g., over a phone without a built-in video camera). However, TCGs are different in that the deck of cards is not shared between players, which nicely side-steps the impossibility results of mental poker. Further, TCGs typically have different types of rules of play and therefore require different techniques to prevent cheating.

As with many card games, most TCGs do not have specific time limits for playing individual cards to resolve a turn, except those limits associated with social norms. While players do expect to be able to play *instant* cards quickly in response to another card, in general the timeframe for playing an individual card is longer than that in most other types of games. This gives TCGs some advantage in preventing cheating since certain types of cheats no longer apply when time constraints are not as tightly bound as other genres of games. For example, research in the past has looked at cheating in specific types of games such as role-playing, first-person shooters, and real-time strategy games [1, 5, 7]. However, this prior work does not address cheating *within* the actual game such as by not actually shuffling cards, or choosing your most desired card instead of the next one from the top of your deck. In particular, we see that many of the issues related to cheating in TCGs can be solved by securely and fairly generating random numbers. This is mainly because TCGs rely on random generation of decks and the fair shuffling of those decks for play.

For M+G to be viable as a P2P protocol for TCGs, its performance must allow a game to be played at the same pace that players using a physical TCG would play. We compare the performance of our implementation of M+G on the Android$^{\text{TM}}$ platform to real-world measurements of a popular TCG and we also benchmark its performance in terms of latency and time. Our results show that M+G works well on modern mobile devices and would allow players to play P2P TCGs.

2 Background

Most modern trading card games have their roots in Magic: The Gathering$^{\text{TM}}$, which was released in 1993. The game consists of a complete library of cards where players build their own collection by purchasing packs of cards. Each individual card has some level of rarity such that the more rare a card is, the fewer copies are produced and randomly distributed in card packs. Each card pack has some guarantee of containing a set ratio of very rare, rare, and common cards. For example, a card pack may be guaranteed to contain at least one very rare card, three rare cards, and the rest common cards. As one may guess, the rarest cards tend to be the most valuable and are often the most *powerful* in terms of gameplay, thereby creating an economy around collecting the rarest cards. Other well-known examples of trading card games include Pokémon$^{\text{TM}}$, the World of Warcraft Trading Card Game$^{\text{TM}}$, and the Yu-gi-oh! Trading Card Game$^{\text{TM}}$.

2.1 Mental Poker and Distributed Random Number Generation

The idea of playing cards in a distributed fashion without cheating, termed *mental poker*, was first discussed by Shamir et al. [12]. The authors show that its impossible to deal from a *shared deck* of cards without one player knowing what card another player received. They then described a protocol that relies on commutative encryption to solve this problem. Note that their impossibility result still holds: if one could break the encryption in a reasonable amount of time, dealing from a shared deck of cards without revealing who received which card to the other player is still impossible. The primary difference between this problem and our problem is that in our case, the deck of cards between players *are not shared*. Without a shared deck, our solution can avoid using commutable encryption. As with their solution, we rely on the encryption not being easily breakable, where *easy* means within the span of time it takes to complete a game.

The protocol in this paper relies on being able to generate a random number between two or more players fairly. In this case, we define fairly as either player not being able to influence the outcome of the generated random number and this is solvable by the well-known coin flipping by telephone protocol [3]. With this method, Alice picks a random number r_A, cryptographically hashes it and sends the hash,

Cheat	Level	P2P	Client/Server
Denial of Service	Network	✓	⋆
Fixed Delay	Protocol	✓	⋆
Timestamp	Protocol	✓	⋆
Suppressed Update	Protocol	✓	
Inconsistency	Protocol	✓	
Collusion	Protocol & App	✓	✓
Secret revealing	App	✓	⋆
Bots/reflex enhancers	App	✓	✓
Breaking game rules	Game	✓	⋆

Table 1 *A Taxonomy of Cheating:* Check marks indicate whether this type of cheat is possible under the listed architecture. Stars indicate cheats which are partially possible.

$H(r_A)$ to Bob. Bob picks a random number, r_B, signs it and sends it to Alice. Alice then XORs r_A and r_B to determine the final value. Alice can then reveal the result later by giving Bob her private number allowing Bob to compute the same value. Note that since Alice has no idea what number Bob will pick, she cannot influence the final random value. Furthermore, Bob cannot influence the final value since he has no idea what initial value Alice chose. Expanding this to n players requires that each player generates a private number and 1 public number to share with the other $n-1$ players. Each player then XORs their private number with the $n-1$ other public numbers.

2.2 Cheating in Games

We define cheating as any action by a player that circumvents the normal course of actions in an game that gives the player an unfair advantage in the game or over another player. Cheats are primarily possible due to security flaws in an application, protocol, or network. We can taxonomize cheats into categories based on the *layer* which they occur [7], which is useful when we are considering what we can and cannot prevent at a particular layer. We note that Yan and Randell also created a classification of cheating in games [13], which differs in classification technique. Table 1 lists several common types of cheats and the layers they occur at.

Cheats occurring in the first category, the network level, allow players to gain an advantage in the game by exploiting security flaws in network and routing protocols. Cheats occurring at the application level occur from applications modified from their original intent. Typically network and application level cheats both occur through modifying the application, though network cheats specifically target security flaws with the network protocols. Last, cheats occurring at the game level are cheats that occur in the game by breaking game rules (possibly by exploiting bugs or sidestepping rules in some way). These may also occur by modifying the application, such as adding new beneficial cards to a deck during play.

Much of the research in cheat prevention in games has looked at preventing protocol or application level cheats, i.e., those cheats which occur by modifying network protocol behavior or altering the application in order to gain an unfair advantage. For example, Baughman and Levine developed *asynchronous synchronization*, one of the first protocols that prevented some network level cheats [1]. Cronin et al. added pipelining to the simple lockstep protocol and secured it from several types of cheats introduced by adding a pipeline to actions [6]. GauthierDickey et al. developed the NEO protocol as a replacement for a simple lockstep protocol that further prevented network level cheats [7].

At the application layer, Li et al. describe information dissemination strategies that limit state information exposure to clients [9]. Chambers et al. showed how to prevent *maphacks*, a type of information exposure cheat [5] in real-time strategy games. Schluessler et al. showed how to detect when players were using bots, which fall under application-level cheats [11].

In this paper, we focus specifically on avoiding or detecting *game level* cheats in Online TCGs, which are those cheats that occur by breaking the rules of the game, specifically centered around preventing players from inserting/removing cards, seeing the opponent's cards, or not fairly shuffling a deck of cards. The original protocol for detecting and preventing cheating was described in [10].We detail this further in Section 5.

2.3 NAT and P2P Protocols

As a peer-to-peer protocol, users invariably have to deal with the problem of network address translation (NAT). Using consistent endpoint translation in the NAT and *TCP hole-punching*, 80-90% of the peers can successfully connect to each other [4]. However, this does not account for the case where two users are behind the same NAT trying to connect to other users. In this case, a 3rd party would need to be used for relaying port numbers prior to TCP hole-punching. Our architecture addresses the problems introduced by NAT through the use of a relay.

3 An Architecture for P2P TCGs

In order for Match+Guardian to work properly, our protocol relies on some services which are typically provided by a centralized authority. Of course, in an ideal design, one would have a purely peer-to-peer framework without the need for a centralized server, and this could eliminate the major hardware and network costs required for provisioning the game. But to date, researchers have not come up with

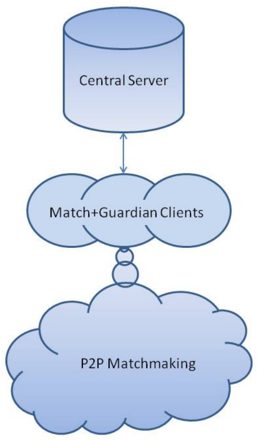

Fig. 1 A High-level Architecture for P2P TCGs: Match+Guardian is used for gameplay between players, but connects to a centralized server for card purchases and trades. In addition, M+G connects to P2P services for matchmaking.

Responsibility	Description
Signature and Purchasing Authority	The server will act as a trusted signature and purchasing authority to allow players to purchase and verify ownership of new cards
UUID Generator	The server will generate unique identifiers for cards and players
Trade Broker	The server will act as a trusted trade broker between players
Communications Broker	The server will act as a trusted relay for communication between peers if they are unable to traverse NAT

Table 2 Centralized server responsibilities in the high-level architecture.

reasonable solutions for some of the services required in a P2P TCG, and therefore our architecture requires a centralized server. The high-level architecture for P2P TCGs is shown in Figure 1.

3.1 The Central Server

In our design, the central server is used for any player action that requires global connectivity or a trusted authority. However, these player actions involve creating an initial account, purchasing cards in the game, trading cards with other players, and connecting between players when both are behind non-traversable NAT. In other words, the most common actions, which are finding other players and actually playing the TCG, occur in a completely decentralized fashion.

First in our architecture, the server acts as a signature and purchasing authority. When players purchase the game, or a set of cards in the game, they are given the digital certificate of the central server so that they can verify anything signed by the server even when not online. Whenever a card is purchased, the server digitally signs it with information described in Section 5. Because players need to verify card ownership, all purchases must go through the central server. On a practical note, purchases must also go through the central server since credit cards need to be processed and records of purchases need to be kept.

Second, the server is used to generate unique identifiers for cards and players. When cards are added to the set of cards in the TCG, the server generates a new and unique identifier for them. This identifier is then used in all transactions involving the cards. The server also generates unique identifiers for the players. This identifier is created when the player first creates an account to play the game. The identifier is then digitally signed by the server so that players can exchange identities (i.e., certificates) to show that they are valid players in the game.

Third, the central server acts as a trade broker. Because cards are digitally signed by the server when purchased, trades must be handled by the server since it must digitally re-sign each card to show that the trade occurred–player A cannot use player B's cards which have signed player B IDs associated with them. This re-signing of the cards shows that the trade was legal.

The major difficulty that arises from trades in a peer-to-peer system is that even though a player may trade a card, he or she can keep a digital copy of the card and may still attempt to use it. This problem is not much different than digital certificate revocation, except that we are trying to prevent the use of a card after it has been traded. The problems can be mitigated somewhat by using expirations on the signatures, requiring players to periodically re-sign cards. However, we plan to further investigate handling trades in TCGs in our future work.

Fourth, the central authority is used as a rendezvous point for clients who need to play their game, but who are both behind NAT. Players can connect to this server, which in the worse case can simply act as a relay, or in the best case, can help both players traverse their NAT.

3.2 Peer-to-Peer Services

As the high-level architecture in Figure 1 shows, game-play with M+G, matchmaking and data caching occur as peer-to-peer services. First, the primary purpose of M+G is to provide distributed cheat-proof game play of the TCG. Once a player has created their account (and therefore received a digitally signed unique identifier) and purchased cards, they can play games with other players while being completely disconnected from the server. As long as two players can connect to each other, whether through the internet or through ad-hoc networking, they can play their TCG using

M+G since all cards and moves can be validated via downloaded or exchanged signatures.

For matchmaking, players can join a peer-to-peer publish/subscribe service as long as some type of range queries are possible. For example, Mercury [2] and SkipNet [8] both allow ranges to be searched via internal ordering of the data. SkipNet, based on the concept of skip-lists, orders the nodes by content and then provides pointers in the list to jump quickly to farther nodes in the data. Since content is ordered, one can search for the start of your query and then walk through adjacent nodes to locate all content that is in the range of your query. Mercury provides multi-attribute range queries in a publish subscribe system. In either case, to provide matchmaking, players can insert match information into the system so that other players can search and locate possible competitors. For the more simple case of two players competing directly over perhaps an ad-hoc connection, a simple service can be created to locate each other.

4 Play Styles

In modern trading card games there are multiple styles of play. For each style, there are different steps and techniques associated with them. When referring to the different gameplay steps, play styles, and card selection techniques, it is important to be clear on the terminology involved.

4.1 Definitions

1. **Universe deck** (D_u) - The set of all cards that exist in the game. This set is defined by the publisher of the trading card game.
2. **Base deck** (D_b) - The set of all cards a player owns and is therefore authorized to use during a gameplay session. Note that, $D_b \subseteq D_u$, since any card outside of the *universe deck* cannot exist. Each player has their own *base deck*, determined by their purchases or trades.
3. **Play deck** (D_p) - The set of cards from their *base deck* a player has selected to use during this gameplay session. The *play deck* must be a subset of the *base deck*, thus $D_p \subseteq D_b \subseteq D_u$. The *play deck* is typically constructed ahead of time based on the synergies of using particular cards together.

4.2 Styles of Play

In modern trading card games, play styles can be divided into the following common forms:

1. **Sealed deck**, where each player begins with a fresh random set of cards. The random set of cards becomes the player's *base deck* for the duration of the session.
2. **Draft deck**, where each player *drafts*, or picks, cards from a random set of cards. The drafted cards become the player's *base deck* for the duration of the session.
3. **Constructed deck**, where each player has already purchased, collected, or traded cards to create their *base deck* from which a subset of cards are chosen to construct a *play deck*.

Although the gameplay styles of these three forms are unique, the individual steps that compose these styles have significant overlap. By decomposing these styles into discrete securable steps, we can reduce the problem space, allowing us to present a common solution for each kind of problem faced by these different play styles. Note that from these common steps, new gameplay styles can be crafted.

4.3 Sealed Deck Play

In a sealed deck game, each player is given an unopened deck of cards which is used to strategically construct a play deck prior to the actual match. This set of cards represents the player's *base deck*. Sealed deck games come from tournaments, where the idea is that if the decks are chosen randomly (consisting of some distribution of common, rare and very rare cards), then the matches are more representative of the skills of the players. Beyond this initial step in creating the play deck, sealed deck games are similar to constructed play styles. In the online equivalent of this type of game, a randomly generated deck of k cards (from the entire universe of cards) must be chosen fairly for each player. The securable steps needed to play this game can be described as:

1. Randomly generate a set of cards from the *universe deck* to represent each player's *base deck*. Typically a server would perform the random deck generation, but in a peer-to-peer system, the protocol must handle this step.
2. Have each player select a *play deck* from their *base deck* in a verifiable manner.
3. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
4. Verify when a card is played that it came from the set of that players' *play deck*.

4.4 Draft Deck Play

In draft deck play, each player participates in *N draft steps*. In each draft step, a player starts with *draft deck* consisting of a small number of random cards from the *universe deck*. The player then selects one card from this deck and then passes the deck to the next player. This "select and pass" step repeats until all cards are selected, at which point the next draft step starts except with the order of passing reversed.

After all draft steps have completed, each player uses their drafted cards as their *base deck* and then constructs a *play deck* from it. The securable steps needed to play this style of game are:

1. Randomly generate *N* sets of cards from the *universe deck* to represent each player's starting *draft deck* each draft round. Note that this problem can be reduced to holding *N* rounds of the random base deck generation step used in the sealed deck play style.
2. Pass a player's *draft deck* to another player in a verifiable manner. This verification is similar to the verification of a card during gameplay. The main difference is that all cards are verified at once instead of one at a time as they are played.
3. Have each player select a *play deck* from their *base deck* in a verifiable manner.
4. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
5. Verify when a card is played that it came from the set of that players' *play deck*.

4.5 Constructed Deck Play

Constructed play is a game where each player creates a *play deck* by strategically choosing a subset of cards from their *base deck* and then plays according to the card game rules. Their *base deck* consists of all cards which they have purchased or traded with other players and verification that a player owns a particular card can be achieved by verifying the signature from a purchasing authority. Constructed games represent those games where players or friends compete with each other. The securable steps needed to play this game can be described as:

1. Have each player select a *play deck* from their *base deck* in a verifiable manner. Selecting a *play deck* from a player's collection is no different than the problem of selecting a *play deck* from a randomly generated deck, so this problem can be solved similarly.
2. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
3. When a card is played, verify that it came from the set of that players' *play deck*.

4.6 Uniform Deck Play

In many online games which have tournaments, players are given identical equipment or gear so that player skill is tested on a fair playing field. This can be simulated in a TCG by giving both players identical *base decks* of cards. Players then construct their *play decks* and compete according to the card game rules. The securable steps are then:

1. Have each player select a *play deck* from the *uniform deck* in a verifiable manner. Selecting a *play deck* from the uniform deck is no different than the problem of selecting a *play deck* from a randomly generated deck, so this problem can be solved similarly.
2. Draw a card at random from the *play deck*. This simulates the step of shuffling cards.
3. When a card is played, verify that it came from the set of that players' *play deck*.

4.7 Securable Steps

Now that we have enumerated the popular play styles and their individual steps, we can create a master list of securable play steps for online trading card games:

– Randomly generate a set of cards from the *universe deck* to represent each player's *base deck* (or *draft deck*).
– Pass a player's *base deck* (or *draft deck*) to another player in a verifiable manner.
– Have each player select a *play deck* from their *base deck* in a verifiable manner.
– Draw a card at random from the *play deck*.
– Verify when a card is played that it came from the set of that player's *play deck*.

With this common securable framework, game designers could mix and match game steps to create completely new styles of play, allowing flexibility in the game style. We acknowledge that while this list may not be completely comprehensive for creating all possible game styles, it may be possible to add new securable steps using similar ideas to those presented here.

5 Protocols

For each of the play steps which must be secured, we have developed an appropriate method to ensure that a player cannot cheat in that step. Composing a game from these steps leads to a particular play style. We begin by describing our assumptions, notation, the list of threats we are attempting to prevent, and then detail the protocols individually.

5.1 Preliminaries

In order for our protocols to work successfully, we make a few important but reasonable assumptions. First, each player has a unique identifier. Second, the size of the *universe deck* in the TCG is *n*. Without a loss of generality, we assign the numbers $1...n$ as unique identifiers for each card. Third, we assume that since duplicates of each card in the set of cards can exist in a player's deck of cards, each valid card has a

Field	Purpose
cuid	Card ID: each card must be distinguished from another card by an ID
sn	Sequence number: each duplicate card a player owns has an increasing sequence number
pid	Player ID: each player has a unique identifier, which is attached to the card on purchase
S_c	Company digital signature: each card is signed, after a purchase, by the game company

Table 3 Fields in a digital version of a trading card for our protocol.

second number from 1...*m*. When a player purchases a card from a company, this card first has its unique identifier, and second has this monotonically increasing sequence number (1...*m*) depending on how many of that card the player owns. Cards are then signed by the company with both numbers and with the player's unique identifier to prove that they were purchased legally.

5.1.1 Notation

We use the letter U to indicate the entire universe of cards in the trading card game, where $|U|$ is the number of cards in the set. $H(i)$ is the cryptographically secure hash of i while $E_A(i)$ is i encrypted by A (usually Alice in this case). A digital signature of i is noted as $S_A(i)$. Recall that $S_A(i) = E_A(H(i))$. As such, $S_A(i)$ does not reveal any information about i but can be held as proof that i was the value when i is either revealed (i.e., if using a public-key cryptography system) or if the key K_A was revealed with i since $K_A(S_A(i)) = H(i)$ and the cryptographic hash functions are known to all parties.

$S_c(cuid, sn, pid)$ indicates a card which is digitally signed by the company (S_c), with its card unique identifier (*cuid*), its purchased sequence number (*sn*) and the player's unique identifier (*pid*). *sn* is not a unique number, but the signed tuple $S_c(cuid, sn, pid)$ is a unique tuple. Table 3 lists the contents of a digital trading card. Note that the functionality of a card is distributed by the game company with the purchase of a game, so it need not be included.

5.2 Threat Model

For our threat model, we assume a typical computationally-bounded adversary, capable of injecting packets and passive listening. We assume standard cryptography will prevent the attacker from decrypting packets in a reasonable amount of time (i.e., before the end of the game). Peer-to-peer TCGs are susceptible to the following types of threats:

1. **Unfairness in Card Selection** - We must be sure that while cards are being generated for a player that the player cannot unfairly influence the outcome of that operation. For example, during the generation of a random deck for a sealed deck game, we must prevent a player from influencing the set of cards generated for the random deck.
2. **Discovery of Private Information** - We must ensure that an opponent cannot garner private data concerning which cards another player has during the information exchanges needed in the setup and running of a gameplay session.
3. **Changing Cards at Playtime** - As with a real TCG, players cannot be allowed to add or remove cards from their play deck *during* game play. Thus, any played card must be able to be verified during game play that it was actually a card from the player's play deck.
4. **Collusion** - The mechanism developed for generating and verifying cards must be resistant to collusion. Group operations (such as generating a random *base deck*) must be performed in such a way as to mitigate the case where some, but not all, of the players in the game are attempting to influence the outcome.
5. **Replay** - We must be able to prevent an adversary from replaying moves they eavesdropped on so that they cannot fool another player into thinking that the replayed packet is a cheating attempt by the originator of the move.

To verify if specific game rules (such as how cards behave) are broken or not, each player must keep a log of the game. Since each card is identified and digital signatures are used with moves, one can prove if a player cheats by their signed invalid actions during gameplay. However, for a log system to work, an additional sequence number is required for each move in a match so that a total ordering on the TCG moves can be identified.

5.3 Securely Generating a Base Deck

We begin by describing how Alice and Bob fairly generate a random *base deck* from a *universe deck* in a purely distribution fashion. This deck forms the basis for providing a sealed deck for the sealed deck game style and the base decks for the draft play styles.

1. Alice randomly generates a private number i_A and a public number j_A.
2. Alice signs her private number and only sends the signature $S_A(i_A, \text{nonce})$ to Bob. Recall from our notation that this is simply Alice's digital signature of the tuple (i_A, nonce) and does not include the actual values.
3. Bob randomly generates a private number i_B and a public number j_B.
4. Bob signs his private number and gives $S_B(i_B, \text{nonce})$ to Alice.
5. Alice and Bob exchange the tuples $(j_A, S_A(j_A))$ and $(j_B, S_B(j_B))$. In other words, they exchange their public num-

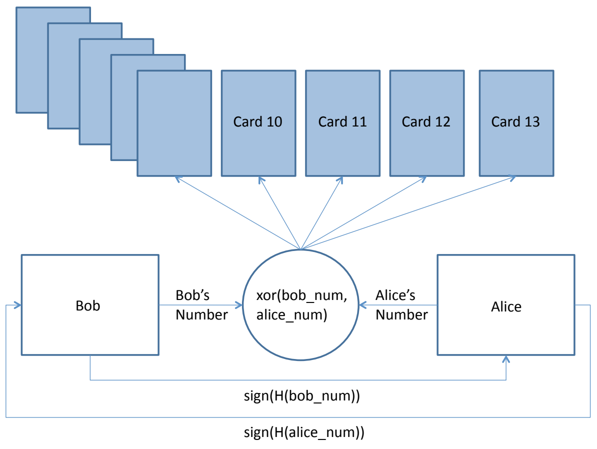

Fig. 2 *Random Card Selection*: This diagram shows how Bob and Alice can participate in random number selection in a verifiable manner while not revealing information about their random number to each other

bers and sign those numbers (so that they cannot later argue that they gave *different* public numbers).

6. Alice XORs j_B with i_A to create a new random number, k_A, while Bob XORs j_A with i_B to create k_B.
7. $k_A \mod |U|$ is the unique identifier of Alice's card from U, while $k_B \mod |U|$ is Bob's first card from U.

At any step, either of the players may refuse to continue. For example, after Alice gives Bob her $S_A(i_A)$ for a particular card, she may wait for Bob's $S_B(j_B)$ in step 5, but not give her $S_A(j_A)$ to Bob. If so, Bob can simply refuse to continue playing as nothing (but time) has been lost. As with the coin-flipping protocol, Alice cannot choose her i_A in such a way so that the resulting k_A is a card that she wants because she does not know j_B before she has encrypted and sent i_A to Bob. The same holds for Bob's choice of j_B—he cannot influence k_A so that Alice gets a poor card from the deck because he has no idea what i_A is. Thus, Alice and Bob can fairly and randomly choose a card from U to be part of their tournament deck.

The above sequence of steps can be repeated k times so that each player has an *base deck* of k cards. However, the players can speed up the processes by generating a sequence of private and public numbers. In the first step, Alice generates k private numbers $i_{1A} \ldots i_{kA}$ and public numbers $j_{1A} \ldots j_{kA}$. Bob does the same thing for private and public numbers. In the second and fourth steps, each private number is signed individually (instead of encrypting the entire sequence) since the values and nonces are revealed as the cards are played to show that they indeed came from the *base deck* of k cards.

At this point, Alice and Bob have *base decks* consisting of k cards. Figure 2 diagrams the flow of steps for random card selection.

5.4 Play Deck Creation

Once a *base deck* of k cards has been selected, Alice and Bob must typically choose a subset of s cards from the *base deck* to form their *play deck*. Note that Alice and Bob choose cards for their *play deck* strategically as certain cards may work better with other cards. Further, the *play deck* does not have a specific size (i.e., Alice and Bob need not have exactly the same sized *play deck*) since for strategic reasons they may choose to construct a larger (for more variety) or smaller (for a higher probability of certain cards) deck.

The primary rule for creating the *play deck* is that it must be done entirely before the game begins. One cannot add cards to the *play deck* from the *base deck during* gameplay. Thus, the following steps must occur to fairly choose the *play deck*:

– Alice chooses a card for her *play deck*. Recall that Bob sent her k public numbers for each of the k cards in her *base deck*. Alice sends $(p, S_A(p))$ where p corresponds to the order of the $1 \ldots k$ values Bob sent her. For example, if the card she chose for her *play deck* was selected by XORing her 5th private number with his 5th public number, she sends $(5, S_A(5))$ to Bob. Alice repeats this for every card she adds to her *play deck* from her *base deck*. This prevents Bob from knowing Alice's play deck, though he knows her base deck.
– Bob chooses a card for his *play deck*, sending Alice the tuple $(p, S_B(p))$ for his chosen card, where p is the order of the public values corresponding to the card he chose. Bob repeats this step for every card he adds to his *play deck* from his *base deck*.

Choosing the *play deck* must occur before gameplay begins and both Alice and Bob may create their decks simultaneously, though order does not matter in this case. When Alice or Bob play a card from their *play deck*, they reveal the associated private number and the order value (which they sent to represent each card). For example, when Alice plays the card that was chosen by Bob's 5th public number, Alice sends the tuple $(5, i_5, S_A(5, i_5))$ to Bob. As Bob knows his 5th public value and was previously sent the hash of Alice's 5th private value, he can calculate the hash of i_5 to see if it matches the previously sent hash. Further, he can XOR i_5 with his 5th public number to determine the *cuid* of the card and verify that the correct card was played.

A diagram describing the process of selecting a card from the *base deck* is shown in Figure 3.

5.5 Drawing a Card from the Play Deck

Once a *play deck* has been created, we need to ensure that when a player chooses a card randomly from their deck during gameplay that the card they chose is truly random. In

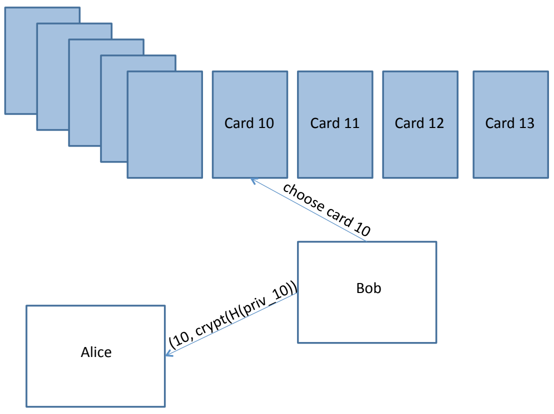

Fig. 3 *Play Deck Selection*: This diagram shows how Alice can be informed what cards Bob is selecting for his deck ahead of time without revealing the value of the card

a physical game, the decks are shuffled and opponents may even *cut* the cards to introduce additional randomness. Players in fact typically want their cards shuffled so that they get an even distribution of various types of resource cards while playing as they cannot predict how the game will unfold. However, since the players cannot observe each other during play, we have to ensure that we get the equivalent of a deck shuffle without cheating. The protocol for this scenario is described below:

1. Alice's *play deck*, consisting of p cards are shuffled. Recall that she has already told Bob which cards are in her *play deck* by referring to them by their pth order value.
2. For each card in her deck, Alice sends $S_A(p, \text{nonce})$ to Bob.
3. Bob further shuffles the deck to ensure that Alice shuffled it properly. When Alice needs a card, she simply asks Bob for the next one, which he sends.

Bob repeats the procedure for his *play deck* so that Alice can ensure his cards are shuffled. When either player plays a card, they can reveal the values (p, nonce) so that the other player can verify that the card is not still in his or her deck from which cards are being dealt.

5.6 Playtime Verification of Cards

Playtime verification of cards is a two step process. First, the card has to be determined to be legitimately selected from the *base deck*. Second, the card has to be verified as a card that exists in the *play deck*.

Base deck verification depends on how the cards were generated, either randomly or user-selected. If the card comes from a user-selected *base deck*, the verification step is simply verifying the purchasing authority's signature on the card to ensure that the player has legitimately purchased that card.

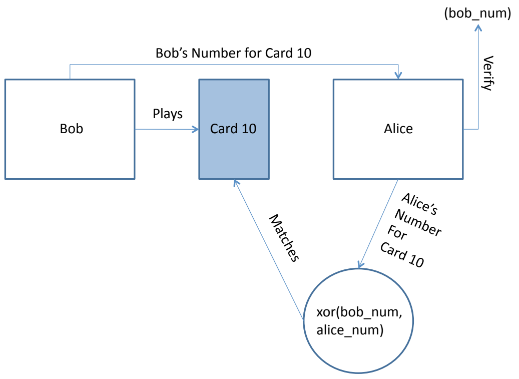

Fig. 4 *Card Verification*: This diagram shows how Alice can verify a card played by Bob during gameplay, allowing for real-time verification of correctness in gameplay

In the randomly generated *base deck*, since both Alice and Bob know the set U, and hence all the cards have unique identifiers, it suffices for a player to reveal the kth private value with its associated nonce to the other player which the other player can then XOR with the kth public value and match the unique identifier with the card just played. For example, Alice plays a card that was generated from the 5th public number, $(j_{5B}, S_B(j_{5B}))$, Bob gave to her. When she plays the card, she also sends $(i_5, \text{nonce}_5), S_A(i_5, \text{nonce}_5)$ to Bob. Using i_5 and nonce_5, Bob can check to see if this was the same value that Alice gave to him previously. If not, he knows she is cheating and has proof of it since he already has her hash of $H_A(i_5, \text{nonce}_5)$ that she sent in step 2.

A diagram describing the process of verifying that a card came from a player's *base deck* is shown in Figure 4.

Play deck verification occurs when Alice needs to play a card from her *play deck*. Alice sends the tuple $(p, i_p, S_A(p, i_p))$ where p indicates the order value of the public and private numbers for generating the card. Since she has already told Bob what order values she was using previously, he can easily verify if she is lying or not about its real value.

5.7 Passing a Base Deck to Another Player

When a *base deck* (or *draft deck*) is passed amongst players, it is important to make sure that the deck of cards being passed is not changed. Assuming the decks being passed were generated using the algorithms described above, verification occurs when the private values used to choose the base deck are revealed. Note that this occurs only when using the Draft Deck or Sealed Deck play styles. For example, Alice who generated a random base deck would reveal all the k private numbers $i_{1A} \ldots i_{kA}$.

With multiple players, revealing the private numbers for the randomly generated base decks does not leak information as all players must know all finalized base decks before play begins. Furthermore, a player cannot insert a new card for this exact same reason.

6 Evaluation

In order to better understand the performance characteristics of our protocol we have designed a realistic set of success criteria under which M+G must perform. First, we observed people playing Magic the Gathering^{TM}, a popular TCG, and modeled the time it took players to perform certain actions. These results serve as a benchmark to compare the performance of M+G against. Second, we designed a simulation environment using the Android^{TM} development platform. Given the proliferation of mobile games, and the very likely scenario that someone might be in an environment in which they have access to a mobile device but no Internet connection (such as on a bus, train, or in a remote location), we thought this platform appropriate. By comparing the performance of our simulation with real measurements of a popular TCG, we demonstrate that M+G can indeed be used for peer-to-peer trading card games.

6.1 Observed Behavior

There is little work done to date in the area of modeling player behavior in TCGs, so we decided to start with the most basic sample set we could: real-life behavior. While observing players, our goal was to focus on the time it took to perform the following activities:

– Draw a card from their deck
– Play a card from their hand
– End their turn

To gather these times, we video-recorded people playing Magic the Gathering^{TM}. After the game was complete, we played back the recordings and measured the time of the players' actions to derive a model for comparison with our simulations. To measure the time to draw a card from their deck, we found the difference between when they first touched a card they were about to draw from their deck and when they placed the card they drew in their hand. To measure the time to play a card, we measured the difference from when they first touched a card they were going to play to when the card was placed on the table. To measure the length of a turn, we measured the time from when a player first started performing actions (for example, moving cards or drawing new cards) to when they played their last card on the table, if any. In other words, we tried to eliminate, as best as we could, all time which was spent talking, thinking, reordering cards, or any other action that was not specifically an action we were measuring.

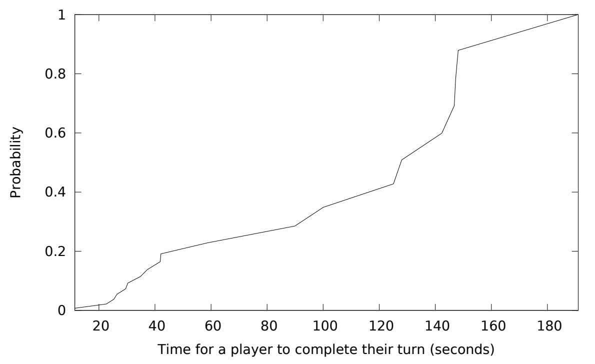

Fig. 5 *Observed turn times*: This graph shows the CDF of turn lengths throughout the observed gameplay session.

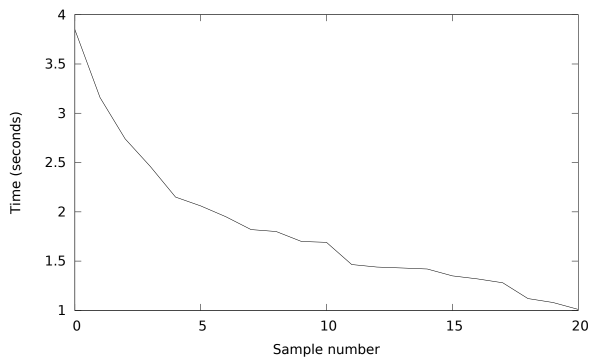

Fig. 6 *Observed card drawing times*: This graph of all measured times for drawing a card from the play deck.

The cumulative distribution function of the measured turn lengths is shown in Figure 5. From this we can see that even short turns last close to 20 seconds, while 80% of the turns are between 30 and 140 seconds long. This length implies that background processing of information, such as signature validation, could easily be done while the player decided what to do in the game.

Figures 6 and 7 show the lengths of drawing cards and playing cards. From these figures we see that drawing a card typically ranges from 1 to 4 seconds in duration while playing a card ranges from just over .5 seconds to 5 seconds. Note that the fidelity of measurements were at 1/30th of a second (i.e., we could advance frame by frame), though the margin of error was likely several video frames due to the subjectiveness of determining when a player was starting or finishing the drawing or playing of a card. These much shorter times imply that card verification and move signatures must occur relatively quickly since we must also include network latency into any networked game.

6.2 Simulation

Once we had a set of evaluation criteria we designed a Peer-to-Peer simulation using the Android^{TM} platform to bench-

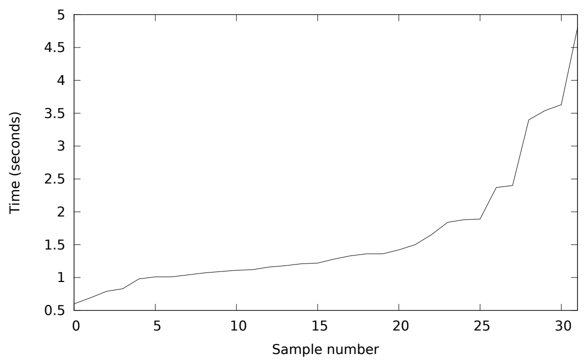

Fig. 7 *Observed card playing times*: The graph of all measured times for playing a single card.

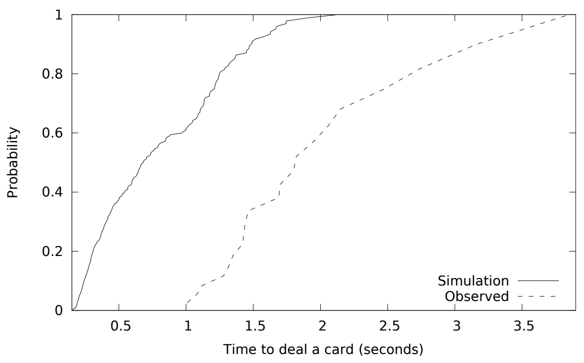

Fig. 8 *Card deal/draw time comparison*: A comparison of the distribution of times to draw (or deal) cards between our simulation and observed play behavior

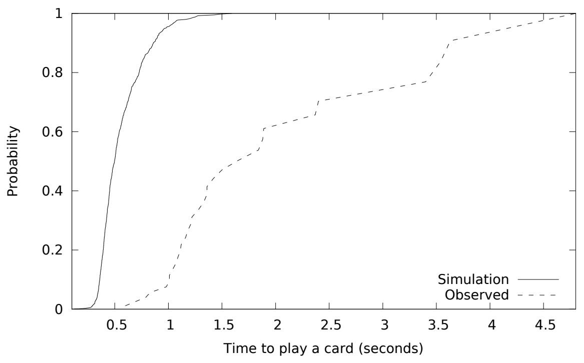

Fig. 9 *Card play time comparison*: A comparison of the distribution of times required to play a card

mark our protocol. The Android^{TM} environment uses Java and has a large user community, which made it ideal for us to develop a reference implementation that can be distributed widely.

The simulation was run under an Android^{TM} emulation environment in which each client's OS reported a BogoMIPS (an artificial benchmark of system performance) of 287.53. By comparison, the Motorola Droid Razr^{TM} reports a BogoMIPS of 1996.68. *Therefore, we consider the emulation environment to be representative of a low-end smart phone with worse-case performance characteristics.* In testing on Motorola Droid Razr^{TM}, we found that the required calculations were at least order of magnitude faster. For asymmetric encryption and signatures we used SHA-256 and an RSA 1280-bit key. For symmetric encryption we used AES with a 128-bit key.

6.3 Results

After running the simulation and processing the video data, we produced combined graphs of two of the performance metrics we were interested in: card draw time and card play time.

In the card draw time graph shown in Figure 8 you can see that our simulation clearly beats the performance requirements of the observed data, in most cases by over a full second. In particular, 90% of the samples from our simulation were able to complete the card drawing operation in under 1.5 seconds. In comparison, only about 30% of the observed card drawing occurred quicker than 1.5 seconds.

In the simulation, draw times were measured as all of the calculations, including signatures, that were required to draw a card from the deck. Play times were measured as all of the calculations that were required to play the card, including signing moves and card verification.

In the play time graph displayed in Figure 9 once again our simulation was able to outperform the observed behavior, in some cases by several seconds. When players actually played cards, 95% of the time, they took between 1 and 5 seconds. However, M+G was able to complete its operation in 1 second or less approximately 95% of the time.

However, an interesting affect of playing an online game, versus a real-life game, is that player expectations may be different. For example, when we play other people in real-life, we often take real-life social cues into consideration. We can tell, for example, when they're concentrating on the cards in their hand trying to decide how to play a move. In an online game, those cues are lost and instead the player must just wait for their turn. Are players less patient in online games? Thus, while our results reflect performance versus real-life measurements, this may not be a completely accurate methodology for comparison and future work is definitely needed.

The results of our analysis is promising and shows that it is possible, given the current state of technology, to implement the encryption required by our protocol while still realizing a performance model that is consistent or better than what players expect in real life.

7 Conclusions and Future Work

In this paper we have demonstrated how using a set of common peer-to-peer protocols, one can develop multiple TCGs using different play styles while ensuring cheat-proof play. This common framework for peer-to-peer based card games enables the community to develop new styles of play without having to resolve the common problems facing games of this genre.

In order to measure the ability of our peer-to-peer protocol for actually playing a P2P TCG, we implemented M+G in Java using the Android^{TM} development environment and emulated a low-end smart phone. For a benchmark, we measured players playing a real TCG and categorized their actions into playing cards and drawing cards. We then compared the timings measured from our simulation with the observed timings and found that indeed M+G worked well even in a constrained mobile environment.

We believe there is a connection between generating sealed decks for online TCGs and automatically generating the monsters, treasures and maps for a game level. One could apply these ideas for fairly generating game content between players, especially if those levels are of a competitive nature. We plan on further exploring these connections. We also plan to design and evaluate an entire architecture around P2P TCGs, including many important elements such as matchmaking, rankings, and trading.

References

1. Baughman NE, Liberatore M, Levine BN (2007) Cheat-proof playout for centralized and peer-to-peer gaming. IEEE/ACM Trans Netw 15:1–13
2. Bharambe AR, Agrawal M, Seshan S (2004) Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, ACM, New York, NY, USA, pp 353–366, DOI http://0-doi.acm.org.bianca.penlib.du.edu:80/10.1145/1015467.1015507
3. Blum M (1983) Coin flipping by telephone a protocol for solving impossible problems. SIGACT News 15(1):23–27
4. Bryan F, Srisuresh P, Kagel D (2005) Peer-to-peer communication across network address translators. In: Proceedings of the USENIX Annual Technical Conference
5. Chambers C, Feng WC, Feng WC, Saha D (2005) Mitigating information exposure to cheaters in real-time strategy games. In: Proceedings of ACM NOSSDAV, pp 7–12
6. Cronin E, Filstrup B, Jamin S (2003) Cheat-proofing dead reckoned multiplayer games. In: International Conference on Application and Development of Computer Games
7. GauthierDickey C, Zappala D, Lo V, Marr J (2004) Low latency and cheat-proof event ordering for peer-to-peer games. In: Proceedings of ACM NOSSDAV
8. Harvey N, Jones MB, Saroiu S, Theimer M, Wolman A (2003) Skipnet: A scalable overlay network with practical locality properties. In: In proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA, URL `http://citeseer.ist.psu.edu/harvey03skipnet.html`
9. Li K, Ding S, McCreary D (2004) Analysis of state exposure control to prevent cheating in online games. In: Proceedings of ACM NOSSDAV
10. Pittman D, GauthierDickey C (2011) Cheat-proof peer-to-peer trading card games. In: Proceedings of the 10th international Workshop on Network and Operating System Support for Games (NetGames'11)
11. Schluessler T, Goglin S, Johnson E (2007) Is a bot at the controls?: Detecting input data attacks. In: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games (NetGames'07), pp 1–6
12. Shamir A, Rivest RL, Adleman LM (1981) Mental Poker. The Mathematical Gardner pp 37–43
13. Yan J, Randell B (2005) A systematic classification of cheating in online games. In: NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games