Cheating detection in P2P online trading card games

Rodrigo R. Leal , Marcos A. Simplicio Jr , Mateus A. S. Santos , Marco A. L. Gomes , Walter A. Goya

[1]Escola Politécnica - Universidade de São Paulo
Av. Prof. Luciano Gualberto 158, trav. 3, 05508-900, São Paulo, Brazil.
e-mail:{rleal,mjunior,mateus,mgomes,wgoya}@larc.usp.br

Abstract. *We present FairShuffle, a solution that allows players participating in Trading Card Game (TCG) match to detect cheating attempts right when they occur, and without the intervention of trusted third party (TTP). The protocol relies basically on commitment protocols built using hash functions, thus displaying a reduced computational cost. In addition, it displays many appealing characteristics, such as: support to multiple players; tolerance to players' dropouts during a match; resistance to collusion among any number of players. As such, FairShuffle is well-adapted for securing TCG games played in a peer-to-peer (P2P) environment.*

1. Introduction

Online gaming is today a very lucrative market, with millions of users all around the globe. Namely, the retail value of online games was U$ 3 billion in 2011 [D2D 2012], and analysts believe that this industry's global revenue will grow by more than 10% per year until 2016 [yStats 2012]. This growth is boosted by factors such as the expansion of broadband Internet access all around the world [Hsu and Lu 2007] and by the continuous increase in the number of users that play games on their mobile devices (which recently surpassed the 100 million mark in USA [Newzoo 2012]). Meanwhile, many games have lead to the formation of very dynamic communities and even to in-game economies whose items have real-world value [Salomon and Soudoplatoff 2010]. At the same time, the increasing popularity and complexity of such games implies the need of deploying cheating-detection mechanisms for ensuring the continuous interest of honest users in playing. However, the development of secure and efficient solutions for online games is a challenging issue. Although the straightforward approach of relying on a trusted third party (TTP) for handling the data from all players at all times can solve this issue, this approach displays low scalability, which motivates the development of solutions game architectures based on peer-to-peer (P2P) [Jardine and Zappala 2008, Fan et al. 2010]. On the other hand, the inherent scalability of peer-to-peer (P2P) environments comes with a price: the lack of a central trusted entity monitoring the game greatly facilitates the activity of malicious players.

A popular multiplayer online game genre with potential for deployment in a P2P architecture is the so-called Trading Card Game (TCG), also known as Collectible Card Game (CCG), a category that includes titles such as *Magic: The Gathering*[1], *Lord of the Rings TCG*[2], and *Duels Warstorm*[3], to cite a few. Unlike traditional card games (e.g.,

[1]Wizards of the Coast – `http://www.wizards.com/magiconline/`
[2]Decipher Inc. — `http://lotronline.decipher.com/`
[3]Challenge Games – `http://www.warstorm.com/`

poker) in which all players share a common and fixed set of 52 cards, players in a TCG use their own decks of cards, acquired through trading or purchasing, when playing against each other. Personal decks are built from a very large pool of cards, which grows periodically with new releases from the game provider. This diversity is undoubtedly one of the most attractive features in this genre of game, but, unfortunately, it also creates additional opportunities for cheating: since the players are not necessarily aware of the composition of their opponents' decks, it becomes more difficult to determine (1) if a card played was actually part of that deck when the match started, (2) if that deck was fairly shuffled and (3) if the order of the cards was not manipulated during the match.

Aiming to tackle the above issues, this paper proposes FairShuffle, a solution for preventing cheating in P2P multiplayer online card games in which each player uses his/her own deck of cards, such as TCGs. FairShuffle supports multiple players while preventing collusion attempts among them. It is also robust enough to allow any number of players to leave the match without affecting the remainder player's experience. In addition , FairShuffle has a reduced computational cost, achieved by the use of lightweight cryptographic methods (namely, hash functions), being thus easily implementable even in constrained (e.g., mobile) devices. Finally, as discussed in section 7, FairShuffle is to the best of our knowledge the first protocol able to prevent cheating and collusion in TCG matches involving more than two players.

2. Scenario description

As discussed in [Pittman and GauthierDickey 2013], the overall architecture of P2P Trading Card Games is composed by two main entities : the game provider's server and the players.

2.1. Game Server

The game server is responsible for defining which cards are available in the game, informing the users of new editions when they are released, and also for managing the users' accounts. However, the server's interactions with players should ideally occur only between matches, while the matches themselves should be handled in a purely P2P manner. More specifically, the game server in a P2P TCG assumes the following responsibilities: generation of unique IDs for players (P_{uid}) and cards (c_{uid}); signature of player identities and card ownership; card trading intermediation.

2.2. Players and decks

As briefly discussed in section 1, online TCGs are usually played by two or more players and they connect each other by P2P or TTP connections. And the game provider's server can offer rendezvous point as mentioned in section 2.1.

Each player uses his/her own deck built from a large pool of cards made available by the game provider. The cards can be obtained via trading or purchasing during the interval between matches. Following the notation of [Pittman and GauthierDickey 2013], we call the set of all cards available the *Universal Deck* (D_u), the set of cards the player is authorized to use the *Base Deck* (D_b) and the set of cards actually used in a match the *Play Deck* (D_p), so that $D_p \subset D_b \subset D_u$ (see Figure 1). As pointed out in section 2.1, each player is identified by a unique ID and each card in a player's base deck is bound

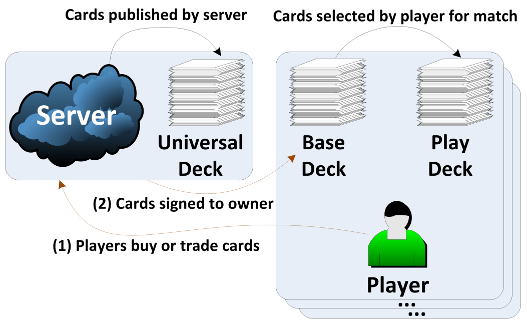
Cards published by server
Cards selected by player for match
Server
Universal Deck
Base Deck
Play Deck
(2) Cards signed to owner
(1) Players buy or trade cards
Player
...

Figure 1. Different types of decks in a TCG: universal, base and play decks.

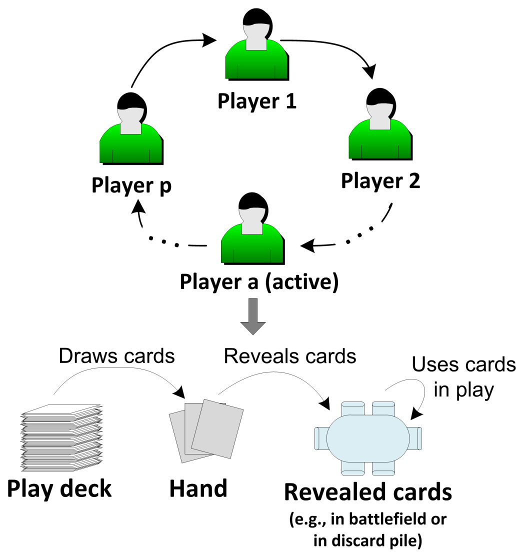

Figure 2. Typical turn dynamics of a TCG.

to that player's ID. This allows the players to design personal play decks from their own base decks for competing with other players. The exact size of a play deck constructed from the base deck may vary from game to game, and in some cases it may be arbitrarily chosen by each player. Moreover, most TCGs allow players to have repeated cards in their decks, albeit such repetitions may be limited in number.

2.3. Game Dynamics

After the play deck is built, the actual match can begin. As many traditional card games, TCGs are played in turns. In this context, the larger the number of players in a same game instance, the longer they have to wait for their own chance to play. For this reason, the "massive" nature of TCGs typically comes from the large number of users participating in different matches rather than playing in a single match.

The turn dynamics depends on the rules of each specific TCG. In addition, this dynamics can be changed by some card placed into play, which may have particular abilities such as: forcing all players to randomly remove a certain number of cards from their hands or play decks; allowing a player to peek at the cards in another player's hand or play deck; allowing a player to draw extra cards; and many other "unusual" effects. Despite this variability, the typical actions that can be performed by the turn's active player are the following (see Figure 2):

- Draw cards: the active player takes a certain number of cards from his/her own play deck and place them in his/her hand without disclosing them to other players.
- Reveal cards: the players reveal some of the cards from their hands, putting them into the battlefield if the card's effect is permanent, or in a discard pile if the card's effect is only temporary. Rules may apply regarding who can reveal cards in each turn, although usually every player is allowed to reveal a card in response to actions of the active player.
- Use the cards in play: permanent cards in the battlefield can be used repeatedly by their owner (e.g., to attack an adversary), until they are removed according to the game rules.

The match ends when a player achieves some goal, such as reducing the number of “life points” of all other players participating in the match to zero.

3. Preventing cheating in TCGs: system requirements

In the scenario described in section 2, the violation of some game rules involving a card that has already been revealed may be easily detected by other players: after all, any player can verify if its effect over the game is the one defined by the set of game rules. The same does not apply, however, to the detection of cheating attempts related to the cards in the player’s hand or play deck. With this issue in mind, and using some of the ideas introduced by Crépeau [Crépeau. 1985] in the context of mental poker [Shamir et al. 1978] and also from [Pittman and GauthierDickey 2013] in the context of TCGs, we can identify the following generic security requirements that apply to a wide variety of TCGs and are the focus of this work:

1. *Cheating detection independent of a trusted third party (TTP)*: the security of the game should not depend on the intervention of a trusted party.
2. *Deck consistency*: even though players in a TCG may build their own play decks, the cards in a play deck cannot be modified after the start of a match.
3. *Uniform random distribution of cards*: each player’s hand must depend on contributions from all players, in such a manner that none of them is able to predict or control the order in which cards are drawn by any given player.
4. *Complete confidentiality of cards*: normally, players are only allowed to learn the cards they hold in their own hands, not the cards faced down in any deck or in an opponent’s hand.
5. *Minimal effect of coalitions*: in matches involving more than two players, some of them can decide to establish a parallel communication channel and then exchange information about the match (e.g., the cards in their hands) or about the game protocol (e.g., a secret key). If this happens, the amount of information gained by those players should be equivalent to what they know separately; for example, they can share information about their own hands, but they cannot learn anything about the cards in an honest player’s deck or hand as a result of this coalition.
6. *Cheating detection with very high probability*: any attempt of cheating must be detected by the solution, either at the end of the game or, preferably, at the moment it happens.

We note that general security measures such as establishing a secure communication are not made explicit in the above requirements, since they can be easily solved using standard mechanisms such as SSL/TLS [IETF 2008].

In addition to security, the following efficiency and flexibility features also apply:

1. *Reduced computational cost*: the protocol should have low dependency of costly operations, such as those commonly involved in asymmetric cryptographic algorithms. Moreover, the players should not need to store large data structures during the game. Finally, the protocol should not involve a large amount of communication between players.
2. *Scalability and support for multiple players*: the protocol must be flexible enough to support two or more simultaneous players in the same match.

3. *Tolerance to player dropout*: if a player leaves the game, either intentionally or by accident, the remaining players should be able to continue playing [Roca 2005]. This is a crucial requirement in the context of online games, since a player may be suddenly disconnected due to network problems or simply be eliminated from the match after being defeated.

4. Proposed Solution: FairShuffle

In what follows, we describe how FairShuffle can be used for preventing cheating attempts from going unperceived during a game involving the elements and dynamics described in section 2. For this, we decompose the match in two phases. The first is the match *Initialization*, in which FairShuffle ensures that the cards picked by each player for his/her own play deck remain confidential but cannot be changed afterward. The basic idea is that all players commit to the cards in their own decks, in a given order, and also to a sequence of pseudo-random numbers that guarantees that all players' cards will be drawn in an unpredictable manner. The second refers to the actions taken during the actual match, namely *Card Drawing* and *Card Revealing*. In this case, FairShuffle provides methods for verifying cheating attempts when a player draws or reveals a card, which is accomplished by analyzing whether the action violates the commitments provided during the match Initialization. Even though the exact action to be taken toward cheating attempts is out of the scope of the protocol, its design is flexible enough to allow many different approaches, such as asking the cheating player to undo his/her move, banning that player from the match, or ending the match itself.

4.1. Preliminaries and notation

Consider a match with p players, each of which having a unique identifier P_i ($1 \leqslant i \leqslant p$) and a deck of d_i cards. Consider also that all players agree on the adoption of a suitable hash function $hash$ of length h_{len} (e.g., SHA-256 [NIST 2008]), as well as on a pseudo-random function prf (e.g., one of the hash-based functions described in [NIST 2009]).

Without loss of generality, we assume that the turn sequence is defined by the index i. In other words, P_1 is the first to play, followed by P_2, and so on until P_p's turn, after which P_1 becomes the active player once again.

The set of card identifiers in P_i's play deck is denoted by $\{ID_i^1, ..., ID_i^{d_i}\}$, where ID_i^j unequivocally identifies a card and its underlying properties, as well as its owner. For example, for identifying cards in an official match, one could make $ID = (c_{uid}, P_{uid}, sign)$, where $sign$ is the server digital signature linking the player identified by P_{uid} to the card identified by c_{uid}. In informal matches, in which the players are more interested in testing generic decks of freely chosen cards rather than using the cards they actually own, making $ID = c_{uid}$ would be enough.

4.2. Match initialization and construction of play deck

For the construction of the play deck, right before the start of the match, the players perform the following *Initialization protocol*, which is also illustrated in Figure 3:

1. After P_i chooses a total of d_i cards from his/her own base deck, that player needs to associate each of those cards to a h_{len}-long secret random number $mask_i^j$ ($1 \leqslant j \leqslant d_i$). Each card in P_i's play deck is then represented as $c_i^j = (mask_i^j, ID_i^j)$, while

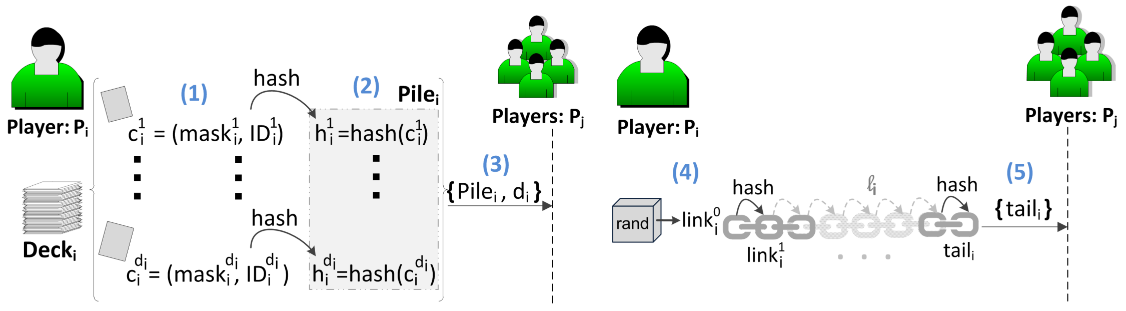

Figure 3. Initialization protocol.

the corresponding play deck is represented by the sequence $Deck_i = [c_i^1, ..., c_i^{d_i}]$. The $Deck_i$ array is stored by P_i, and all of its contents are kept secret until it is time to reveal some card (e.g., when the card is placed into play).

2. P_i then computes the hash of each of the cards in his/her own deck, respecting the previously chosen (arbitrary) order, thus generating the sequence $Pile_i = [h_i^1, ..., h_i^{d_i}]$, where $h_i^j = hash(mask_i^j \parallel ID_i^j)$. The bit-length of h_i^j, h_{len}, must be large enough so that every card from $Deck_i$ will have a unique correspondent in $Pile_i$ with overwhelming probability; otherwise, the corresponding cards become exchangeable, opening way for cheating.
3. Every P_i then broadcasts a message containing $Pile_i$ and d_i to all other players, committing themselves to the corresponding cards and sequence in their play decks.
4. After the reception of the messages from all other players, P_i generates and stores a h_{len}-long random value denoted $seedC_i$, which remains secret during the whole duration of the game. That player then computes and stores a hash chain [Lamport 1981] of length $\ell_i = 1 + \sum_{\alpha \neq i}^{p} d_\alpha$ as follows: first, P_i sets the chain anchor $link_i^0$ to $seedC_i$, and after that computes the remainder $\ell_i - 1$ chain values as $link_i^k = hash(link_i^{k-1})$, $1 \leqslant k \leqslant \ell_i$.
5. Every player P_i then broadcasts the last value in his/her chain, $tail_i = link_i^{\ell_i}$.
6. Finally, P_i locally stores the following information for every player $P_{\alpha \neq i}$: P_α's chain tail, $tail_\alpha$; the sequence of hashes for the cards in that player's deck, $Pile_\alpha$; the hashes for the cards in that player's hand, $Hand_\alpha$, which is initially empty; and the set of hashes for cards already revealed by that player, $Used_\alpha$, which is also initially empty.

In this protocol, the values of $mask$ generated in step 1 are used to mask the real IDs of the play deck cards, while still allowing any player to verify that a card was indeed part of the deck by means of the commitment broadcast in step 3. Hence, $mask$ must be kept secret until the corresponding card is revealed, when its value is then used to verify attempts of cheating (see section 4.4).

4.3. Drawing a card

Suppose that P_a wants to draw a card. Obviously, P_a cannot just choose any card from $Pile_a$. Instead, the following *Card Drawing protocol* takes place for determining, in a fair (i.e., uniform and random) manner, which is the next card P_a must draw (see Figure 5):

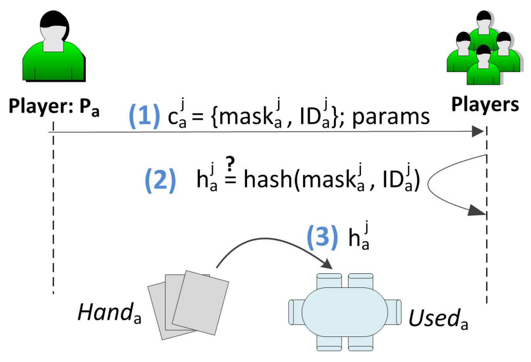
Player: P_a
(1) $c_a^j = \{mask_a^j, ID_a^j\}$; params
Players
(2) $h_a^j \overset{?}{=} hash(mask_a^j, ID_a^j)$
(3) h_a^j
$Hand_a$
$Used_a$

Figure 4. Revealing a card: P_a reveals a card from his/her hand, which is then verified by the other players.

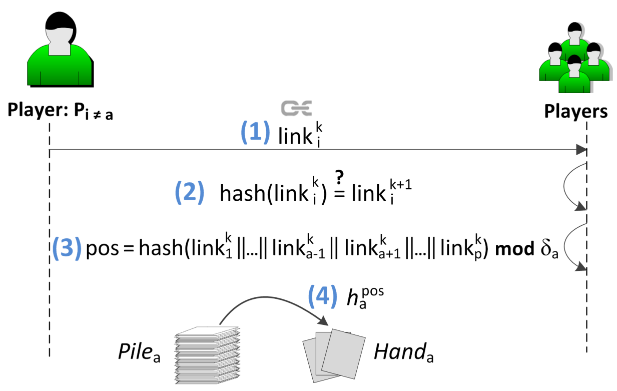
Player: $P_{i \neq a}$
(1) $link_i^k$
Players
(2) $hash(link_i^k) \overset{?}{=} link_i^{k+1}$
(3) $pos = hash(link_1^k \| ... \| link_{a-1}^k \| link_{a+1}^k \| ... \| link_p^k)$ mod δ_a
(4) h_a^{pos}
$Pile_a$
$Hand_a$

Figure 5. Drawing a card: after the interactions shown, P_a draws a randomly chosen card.

1. Each player, except for P_a, broadcasts his/her next chain value $link_i^k$ to all other players.
2. All players verify if the data broadcast correspond to valid chain values, i.e., if the value of $link_i^k$ informed by P_i ($i \neq a$) in the previous step satisfies $hash(link_i^k) = link_i^{k+1}$, where $link_i^{k+1}$ is the chain value currently associated to P_i. If this is the case, every player P_j ($1 \leqslant j \leqslant p$), updates the chain values for the other players, replacing $link_i^{k+1}$ by the recently revealed $link_i^k$; otherwise, they run an adequate mechanism for dealing with cheating, ending the Card Drawing protocol.
3. In order to determine the index of the card to be drawn, all players employ the received chain links as follows: supposing that P_a still has a total of δ_a cards in his/her deck, the index of the card drawn from $Pile_a$ is computed as $pos = hash(link_1^k \,\|\, ... \,\|\, link_{a-1}^k \,\|\, link_{a+1}^k \,\|\, ... \,\|\, link_p^k) \bmod \delta_a$.
4. All players update the information of P_a, removing h_a^{pos} (which corresponds to the card drawn) from $Pile_a$, and adding that same value to $Hand_a$.

The resulting procedure imposes no restrictions on the order in which the players draw cards. For example, suppose all players need to draw a same number of cards at the beginning of the game. In this case, the protocol described allows a player to draw his/her full hand before the next player starts drawing, or one card at a time (taking turns with the other players) until all hands are full, or any other method desired.

4.4. Revealing a card

Suppose that P_a wants to reveal a card to the other players, placing it into the game. Obviously, P_a cannot play any card, but only one of those in his/her own hand (i.e., in the set $Hand_a$). In the proposed solution, all players can verify the honesty of P_a during this process using the following *Card Revealing protocol* (see Figure 4):

1. In order to play the card whose identification is ID_a^j, P_a reveals the value of $c_a^j = (mask_a^j, ID_a^j)$ to all other players, defining the corresponding game-dependent parameters $params$ of that card (e.g., its target).
2. All players compute $h_a^j = hash(mask_a^j, ID_a^j)$, and check if the result corresponds to one of the cards in $Hand_a$. If the previous verifications are successful for all players, no cheating attempt is detected; otherwise, the players run an adequate mechanism for dealing with cheating, ending the protocol.

3. The players remove the value of h_a^j from $Hand_a$, which is then placed in $Used_a$ to indicate that this card was already used by P_a.

5. Security analysis

In this section, we analyze the proposed cheating-detection mechanism taking into account the security requirements presented in section 3.

5.1. Cheating detection independent of trusted third party (TTP)

The proposed solution does not require the intervention of a trusted third party (TTP) during the whole match. More specifically, the activities that involve the game provider's server are restricted to transactions that take place before the match starts, such as card trading and purchasing.

5.2. Deck consistency

By revealing the hash values of all cards during step 3 of the Initialization protocol, the players commit to the cards in their decks and also to the order in which they are organized. Therefore, unless a dishonest player P_d is able to compute a $c_d^{j'}$ satisfying $hash(c_d^{j'}) = hash(c_d^j)$ for some j, violating the security properties of the hash function, P_d cannot modify the data in the locally stored arrays $Deck_d$ and $Pile_d$ without creating inconsistencies with the arrays stored by other players. Such inconsistencies can then be easily detected during the Card Revealing protocol: when P_d would try to place $c_d^{j'}$ into play, the other players would be able to verify that $hash(c_d^{j'})$ does not appear in their own versions of $Hand_d$.

5.3. Uniform random distribution of cards

The order in which the cards are drawn by player P_a ultimately depends on the pos variable computed in step 3 of the Card Drawing protocol. This variable follows a uniform distribution, since it is computed from the application of the hash function over the chain links provided by all players except P_a. Therefore, as long as the input to the hash function is not manipulated, each card has the same probability of being drawn. In the proposed solution, this input manipulation is infeasible due to the use of hash chains: all players reveal $tail$ during step 3 of the Initialization phase, thus becoming committed to a given sequence of pseudo-random contributions for the computation of pos.

It is instructive to note that the absence of such commitment mechanism would allow undesirable situations such as the following: when player P_a needs to draw a card, a dishonest player P_d could wait until all other players disclose their contributions in step 1 of the Card Drawing protocol, and then choose a value that, in combination with the previously revealed contributions, results in some desired card index. This maneuver would not benefit P_d directly, since the contributions from that player are never used to determine the cards drawn by him/her, but two players P_a and P_d could collude in order to place some interesting cards in each other's hands.

In summary, the Card Drawing protocol achieves a similar result as "re-shuffling the deck following each card drawing": after the players reveal a chain link, they are unable to determine *a priori* which will be the index of the next card drawn by themselves or by their opponents, as each card is once again equiprobable. Even if some players collude, they would be unable to circumvent this property as long as there is at least one honest player in the game, whose contribution to the computation of pos would still introduce the necessary randomness to the drawing process. Finally, even though a collusion involving all players except P_a does allow the computation of pos beforehand, it does not unveil any useful information because P_a is the only one who knows the real card IDs corresponding to the hash values in $Pile_a$.

5.4. Complete confidentiality of cards

The irreversibility of the hash function ensures that knowing h_i^j alone does not allow a player to compute the corresponding value of $(mask_i^j, ID_i^j)$. Concatenating a random $mask_i^j$ to the card ID is required because, in the absence of such value: two cards with the same ID would map to the same h_i^j, revealing the existence of repeated cards in a player's deck; a player P_d who knows which cards are in P_i's deck (e.g., from a previous game against that deck) could simply compute the hash of each known ID_i^j and try to match the results to the revealed hash values in $Pile_i$, promptly identifying those cards. Moreover, the non-repetition of $mask_i^j$ used together with each card is in the best interest of P_i, since using a same value for the same card in different games would allow adversaries to recognize those cards from their respective h_i^j enclosed in $Pile_i$, not bringing any benefit to P_i.

5.5. Minimal effect of coalitions

Players in a coalition cannot gain any useful information about the cards in a non-colluding player's hand or deck, since all players have access only to their own cards. Moreover, as discussed in subsection 5.3, colluding players cannot subvert the unpredictable nature of the card drawing process as long as there is at least one honest player participating in the match.

5.6. Cheating detection with very high probability

The proposed solution prevents players from learning each other's hands or decks, or determining the cards drawn by some player (including themselves) beforehand. On the other hand, dishonest players could try to take advantage of the resulting secrecy in order to: manipulate the order in which cards are drawn; place into play a card that is not in their hands; build a deck that violates some of the game rules.

Such cheating attempts are nevertheless easily detectable due to the commitment mechanism adopted in FairShuffle, meaning that players who are unable to violate the security properties of the hash function employed cannot cheat except for an arbitrarily low probability (which depends on the hash length adopted). Indeed, a cheating attempt of the first type mentioned above can be detected at the moment it occurs, in step 2 of the Card Drawing protocol. The second kind of cheating can also be detected right away, in step 2 of the Card Revealing protocol; Finally, third cheating category is detectable after the match ends, when all players reveal the secret information used to conceal their cards and the play decks can be fully analyzed.

6. Performance Analysis

In what follows, we evaluate the proposed solution considering the efficiency and flexibility requirements presented in section 3.

6.1. Computational cost

Most of the communication overhead introduced by FairShuffle is concentrated in its Initialization protocol, in which P_i broadcasts $d_i \times h_{len}$ bits corresponding to $Pile_i$ and h_{len} bits for $tail_i$. In comparison, the Card Drawing protocol requires every non-drawing player to broadcast h_{len} bits, corresponding to a single link from his/her own hash chain. Finally, the FairShuffle-specific information broadcast in the Card Revealing protocol corresponds only to the card itself (c_{len} bits). The first column of Table 1 summarizes the communication cost of FairShuffle.

In terms of processing, the Initialization protocol requires each player P_i to generate one random nonce — $mask_i$ — and compute one hash for each of the d_i cards c_i in his/her own deck. In addition, that player must also create a hash chain of length $\ell_i = 1 + \sum_{j \neq i}^{p} d_j$. Therefore, for a game in which a total of t_c cards exist, this protocol involves approximately t_c hash computations. During the actual match, every player needs to perform $p+1$ hash computations whenever a card is drawn for verifying the hash links informed and for computing pos. When P_a reveals a card, all other players need to perform one hash computation for verifying if the card revealed is indeed in that player's hand. The second column of Table 1 summarizes the processing costs of FairShuffle's protocols, assuming that all random numbers are generated using the pseudorandom function prf.

In terms of memory usage, and again for a game involving t_c cards, the total overhead for player P_i is composed by the following elements. From the Initialization protocol, that player must first store his/her own information — namely $d_i \times c_{len}$ bits for $Deck_i$ and the memory associated to his/her own hash chain — together with his/her opponents' information — $(t_c - d_i) \times h_{len}$ bits for the committed play decks $Pile_{\alpha \neq i}$ and $(p-1) \times h_{len}$ bits for the committed chain tails. It is worth noticing that the memory required by the hash chain itself can be arbitrarily reduced at the cost of extra processing: since any chain link $link_i^j$ can be computed from the anchor $link_i^0$ by the repeated application of the hash function, only $link_i^0$ actually needs to be stored; some intermediary chain links may be stored, though, in order to reduce the number of hash computations during the match. The third column of Table 1 shows the overall memory overheads involved in FairShuffle, summarizing the discussion above.

In order to give a more concrete insight on the total computational cost of FairShuffle, the algorithms involved were evaluated on a Motorola Milestone 2, a reasonably high-end mobile device equipped with a 1 GHz processor, 8 GiB internal flash memory, 512 MiB of RAM[4]. Specifically, we evaluate the SHA-256 hash function [NIST 2008], which provides a 128-bit security level and lead to $h_{len} = 256$ bits. As pseudorandom function, we use the HMAC-based construction described in [NIST 2009]. All implementations are taken from the publicly available RELIC cryptography library [Aranha and Gouvêa 2012]. The results are summarized in Table 2. This table considers that the base and play decks contain, respectively, $base = 100$ and $d_{=}60$ cards, which are reasonable values for *Magic: The Gathering* tournaments [Wizards of the Coast 2009]. It also assumes a format of ID that leads to $c_{len} = 200$ bytes.

Since each execution of SHA-256 takes on the order of 10 μs, it is easy to see that

[4]Source: `http://www.gsmarena.com/motorola_milestone_2-3495.php`

Table 1. Computational overhead per player in FairShuffle. We use the notation <cost for active player / cost for other players> when these costs differ.

	Communication (bits)	Processing	Memory (bits)
Initialization	$h_{len} \times$ $(d_i + 1)$	(d_i) prf (t_c) hash	$d_i \times c_{len}+$ $(p-1) \times h_{len}+$ $(t_c - d_i) \times h_{len}$
Card Drawing	<0 / $h_{len}>$	$(1+p) \times$ hash	0
Card Revealing	$<c_{len}$ / $0>$	<0 / $1> \times$ hash	0

Table 2. Benchmarking FairShuffle on the Motorola Milestone 2.

	Communication (bytes)	Processing (ms)	Memory (bytes)
Initialization	1952	6.2 ± 0.4	27648
Card Drawing	<0 / $32>$	$<0.08 \pm 0.01$ / $0>$	0
Card Revealing	<200 / $0>$	<0 / $0.012 \pm 0.001>$	0

the resulting FairShuffle is very lightweight.

6.2. Scalability and support for multiple players

As discussed in section 6.1, the total computational cost involved in the proposed solution grows linearly with the total number of cards and players involved in the match. The maximum number of players in a same match is, thus, limited only by the amount of resources available at the players' devices, while the protocol itself allows any $p \geqslant 2$. Considering that the turn-based nature of TCGs is likely to lead to a small number of players and cards per match (as discussed in section 2.3), even players participating in many simultaneous matches should be able to keep their resource utilization in a low level. For example, in a match with 10 players and 600 cards as the one described in section 6.1, both the amount of data stored or exchanged between players is expected to remain below 4 KiB for the whole match (see Table 2).

6.3. Tolerance to player dropout

If a player P_o suddenly leaves the match, either after being defeated or due to network problems, the impact over the proposed protocol is minimal. Namely, the only modification necessary applies to the Drawing protocol, during which the remaining players must not use nor wait for P_o's contribution when computing pos. Therefore, the remaining players can continue playing normally after they all detect P_o's dropout.

7. Related Work

Many solutions for the mental poker problem have been proposed since its first appearance in [Shamir et al. 1978] (for a survey, see [Roca 2005]), and some techniques for preventing cheating in such games can also be used in the context of online TCGs. Commitment mechanisms, for example, have been proposed in the past for preventing cheating in online gambling scenarios [Showers et al. 2000]. However, existing mental poker protocols usually rely on a trusted third part (TTP) [Fortune and Merritt 1984, Showers et al. 2000], make intense use of costly algorithms for shuffling and drawing cards [Barnett and Smart 2003, Crépeau. 1986, Goldwasser and Micali 1982, Kurosawa et al. 1997], or both [Chou and Yeh 2002, Oppliger and Nottaris 1997]. In

FairShuffle, we explore some important characteristics of TCGs that are not present in mental poker, such as the existence of separated decks rather than a single shared deck and the fact that cards are revealed during the game rather than at its end. This allows the construction of a cheating-detection solution more suitable for TCGs, which is more efficient than existing mental-poker proposals and also TTP-free.

In [Yeh 2008], the authors propose a TTP-free shuffling protocol which uses only symmetric operations (modular additions and permutations) for distributing secret shares among players; during the game, these shares are disclosed to the appropriate players, which can then recover the value of the cards. However, the proposal requires at least three players and is not collusion-resistant, since two colluding players may be able to discover all cards in the deck. Another P2P-oriented solution that relies on secret shares and lightweight operations is the protocol for P2P Scrabble described in [Wierzbicki and Kucharski 2004]. Interestingly, this solution employs a commitment mechanism in order to verify if a card played was actually drawn by the corresponding player. Notwithstanding, this scheme also displays low collusion resistance: the secret shares are built by distribution players, which may collude with other players in order to recover some or even all secrets; moreover, the commitment mechanism depends on an arbiter (another player), which may also collude with other players in order to influence the drawings. Indeed, the secret sharing mechanism employed for shuffling a deck before the match requires the distribution players to know this deck beforehand, which goes against one of the main requirements of TCG games: the confidentiality of cards in each player's play deck. FairShuffle, on the other hand, provides such mechanisms and includes a stronger method against collusion attempts, ensuring the fairness of the cards drawn and played as long as there is at least one honest player participating in the match.

To the best of our knowledge, the only cheating-detection proposal focused specifically on P2P online TCGs is the recently proposed Match+Guardian protocol [Pittman and GauthierDickey 2013]. Nonetheless, it also displays some important limitations. The most important is that Match+Guardian was designed for matches involving only two players and, hence, it does not take into account security against collusion or tolerance to player dropout. For example, Match+Guardian's shuffling mechanism consists basically in allowing opponents to shuffle each other's $Pile$ array, changing the index of the underlying cards, and then only revealing the next card in this shuffled deck when requested by the deck's owner or due to some card effect. If the same concept is applied to a match with three or more players, allowing every player to iteratively shuffle all play decks, two players could easily collude and discover (or even choose!) the order of the cards they will draw. This can be done by a simple collusion between the owner of the deck P_i (who knows how to unmask any card from $Pile_i$) and the last player shuffling the deck (who, ultimately, is the sole responsible for determining the order of the cards in $Pile_i$). In addition, if the final shuffler leaves the game, it is unclear how Match+Guardian would allow the match to continue. It might be possible to use the order of the cards from the previous shuffler, but that would create a mismatch between cards already drawn and those that are still in the deck. The shuffling mechanism of FairShuffle, on the other hand, involves commitments from all (remaining) players, thus preventing collusion attempts and providing tolerance to dropouts.

8. Conclusions

The deployment of effective cheating-detection mechanisms is an important requirement for the success of multiplayer online games, especially when a P2P-based architecture is adopted.

In this paper, we present and analyze FairShuffle, a proposal for detecting cheating attempts in a type of game that has strong synergy with the distributed nature of P2P scenarios: multiplayer online card games where the players use their own decks to play, such as Collectible Card Games (TCGs). The proposed solution employs commitment mechanisms based essentially on hash functions, exposing a players' dishonest behavior in commonplace and cheating-prone situations, such as drawing and revealing cards. This adoption of very lightweight cryptographic tools allows its adoption in a wide range of platforms, including resource-constrained ones. At the same time, FairShuffle provides a wide number of interesting features required in online TCGs, such as tolerance to player dropout, support for multiple players and independence of a trusted third party (TTP) during matches.

References

Aranha, D. and Gouvêa, C. (2012). RELIC is an efficient library for cryptography. http://code.google.com/p/relic-toolkit/.

Barnett, A. and Smart, N. (2003). Mental poker revisited. In *Proc. of Cryptography and Coding*, volume 2898 of *LNCS*, pages 370–383, Berlin / Heidelberg. Springer.

Chou, J.-S. and Yeh, Y.-Y. (2002). Mental poker game based on a bit commitment scheme through network. *Comput. Netw.*, 38(2):247–255.

Crépeau., C. (1985). A secure poker protocol that minimizes the effects of player coalitions. In *Advances in Cryptology: Proc. of Crypto'85*, volume 218 of *LNCS*, pages 73–86. Springer.

Crépeau., C. (1986). A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In *Advances in Cryptology: Proc. of Crypto'86*, pages 239–247.

D2D (2012). Online gaming impacts physical industry. *Digital2Disc Magazine*. `http://digital2disc.com/index.php/news/article/online-gaming-impacts-physical-industry`.

Fan, L., Trinder, P., and Taylor, H. (2010). Design issues for peer-to-peer massively multiplayer online games. *Int. J. Adv. Media Commun.*, 4(2):108–125.

Fortune, S. and Merritt, M. (1984). Poker protocols. In *Advances in Cryptology: Proc. of Crypto'84*, pages 454–464.

Goldwasser, S. and Micali, S. (1982). Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC'82: Proc. of the 14th annual ACM symposium on Theory of computing*, pages 365–377, NY, USA. ACM.

Hsu, C. and Lu, H. (2007). Consumer behavior in online game communities: A motivational factor perspective. *Computers in Human Behavior*, 23(3):1642–1659.

IETF (2008). The transport layer security (TLS) protocol version 1.2. Technical report, Internet Engineering Task Force. http://tools.ietf.org/html/rfc5246.

Jardine, J. and Zappala, D. (2008). A hybrid architecture for massively multiplayer online games. In *Proc. of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'08)*, pages 60–65, NY, USA. ACM.

Kurosawa, K., Katayama, Y., and Ogata, W. (1997). Reshufflable and laziness tolerant mental card game protocol. *IEICE Trans. Fundamentals*, E00-A(1).

Lamport, L. (1981). Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772.

Newzoo (2012). Mobile games trend report. `http://www.newzoo.com/trend-reports/mobile-games-trend-report/`.

NIST (2008). *Federal Information Processing Standard (FIPS 180-3) – Secure Hash Standard*. National Institute of Standards and Technology, U.S. Department of Commerce. `http://csrc.nist.gov/publications/nistpubs/800-107/NIST-SP-800-107.pdf`.

NIST (2009). *NIST Special Publication 800-108 – Recommendation for Key Derivation Using Pseudorandom Functions*. National Institute of Standards and Technology, U.S. Department of Commerce. `http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf`.

Oppliger, R. and Nottaris, J. (1997). Online casinos. In *Kommunikation in verteilten Systemen*, pages 2–16.

Pittman, D. and GauthierDickey, C. (2013). Match+guardian: a secure peer-to-peer trading card game protocol. *Multimedia Systems*, 19(3):303–314.

Roca, J. C. (2005). *Contributions to Mental Poker*. PhD thesis, Universitat Autònoma de Barcelona.

Salomon, M. and Soudoplatoff, S. (2010). Why virtual-world economies matter. *Journal of Virtual Worlds Research*, 2(4).

Shamir, A., Rivest, R., and Adleman, L. (1978). Mental poker. Technical report, MIT.

Showers, B., Prud'homme, G., Gindikin, D., and Oppenheim, K. (2000). Distributed secrets for validation of gaming transactions. Patent No US6.949.022 B1.

Wierzbicki, A. and Kucharski, T. (2004). P2P scrabble. Can P2P games commence? In *Proc. of the 4th International Conference on Peer-to-Peer Computing (P2P'04)*, pages 100–107, Washington, DC, USA. IEEE Computer Society.

Wizards of the Coast (2009). *Magic: The Gathering Tournament Rules*. `http://www.wizards.com/dci/downloads/MTG_MTR_1Jul09_EN.pdf`.

Yeh, C.-C. (2008). Secure and verifiable P2P card games. In *Proc. of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'08)*, pages 344–349, Washington, DC, USA. IEEE Computer Society.

yStats (2012). Global online gaming report 2012. Technical report, yStats.com.