Covert Channels for Collusion in Online Computer Games

Steven J. Murdoch and Piotr Zieliński

University of Cambridge, Computer Laboratory,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
http://www.cl.cam.ac.uk/users/{sjm217, pz215}/

Abstract. Collusion between partners in Contract Bridge is an oft-used example in cryptography papers and an interesting topic for the development of covert channels. In this paper, a different type of collusion is discussed, where the parties colluding are not part of one team, but instead are multiple independent players, acting together in order to achieve a result that none of them are capable of achieving by themselves. Potential advantages and defences against collusion are discussed. Techniques designed for low-probability-of-intercept spread spectrum radio and multilevel secure systems are also applied in developing covert channels suitable for use in games. An example is given where these techniques were successfully applied in practice, in order to win an online programming competition. Finally, suggestions for further work are explored, including exploiting similarities between competition design and the optimisation of voting systems.

1 Introduction

In many games, a player who is able to collude with other participants can gain a significant advantage. In this paper we explore how, in a tournament, a player may surreptitiously authenticate players who may be colluded with, what actions can be taken and what advantage this may gain him.

One of the games for which much research in collusion has been performed is Bridge. Here, systems for transmitting information between partners during the bidding stage are legal and can provide a great advantage to the team more adept in their usage. These schemes typically provide a means by which one player can encode information about his hand in the cards that he plays. His partner (who he is not allowed to communicate with through any other means) can then make a more precise contract.

One complication in Bridge is that while covert channels are permitted by the rules, if the partner of a player making a bid is asked what the meaning of a bid is, then he must answer truthfully [1, 2], so the information sent through the channel cannot be secret. However, the two members of a team do share a secret, e.g. if one player holds all the aces then he knows that his partner holds none, but the opposing team does not know this [3]. If this secret is used as a key, then it is legal for the recipient of the information to only tell what the bid

J. Fridrich (Ed.): IH 2004, LNCS 3200, pp. 355–369, 2004.

means in isolation. He does not need to tell his opponent what the bid means when combined with the knowledge of the player's own hand.

In Bridge, the collusion is between two members of a team, where communication, other than through bidding, is not permitted, however, in Section 2 we discuss the different situation, where the colluding parties are considered to be independent players. Here, communication is simply unexpected, since in a competition it is normal for each player to try to optimise his own performance, so there would be no need for communication with other opponents. In this paper, we examine the situation where several independent players cannot win the competition acting by themselves, but one of them can win if they collude. If the value of the prize can somehow be divided up between the winner and colluders, this option is attractive for all parties.

In order for collusion to work, there must be some means of communicating. If collusion is not expected, then it may be the case that communication is easy, but the case where it is banned is both plausible and more interesting. In Section 3, we discuss how communication can be established, and in particular we show how covert channels can be used for authentication. A number of possibilities are presented and compared, including a scheme which draws on techniques used in low-probability-of-intercept spread spectrum radio to increase the confidence that authentication has been performed correctly.

In Section 4, an example of where these techniques were successfully applied is given. This was a online programming competition where contestants were required to write a program to play *Connect-4* against the other programs entered. We found that it was in fact impossible to guarantee a win in any individual game, however by developing a collusion based system it was possible to win the contest subject to reasonable assumptions about other contestants.

Finally, in Section 5, defences against such types of collusion are discussed. These include prevention, detection, and modifying the competition so that the benefits of collusion are reduced. One option considered is to use the similarities between elections and competitions so as to design better tournament structures.

2 Competition Structures

The type of competition dictates how effective collusion can be and also how it can best be used. In this section, we introduce two simple but popular tournament arrangements (league and knockout) and show how collusion can be exploited. In Section 4, these two arrangements are combined to form the hybrid structure that the techniques described in this paper were designed to win.

2.1 League Tournaments

In a typical league, each of the n players competes against every other player, resulting in $n(n-1)/2$ matches. The structure of a game is not important, only that there are two participants and it may lead to three outcomes: win, lose, or

Table 1. Summary of winners in matches between Fox, Chicken and Optimal players ("—" denotes a draw)

	Fox	Chicken	Optimal
Fox	—	Fox	—
Chicken	Fox	—	—
Optimal	—	—	—

draw. It is almost universal for a win to gain a player more points than a draw and a draw to gain the player more points than a loss.

Without loss of generality, we can assume that the game is *fair*, that is, neither of the players has an advantage. This is because any game can be made fair by playing it twice with the roles of the players exchanged the second time. Fairness implies that a perfect player must draw against itself, therefore, no winning strategy exists for the player. Since the opponent has no winning strategy either, the player must have a strategy that guarantees at least a draw.

In order to calculate a lower bound for the benefit of collusion, we assume the worst case scenario — that non-colluding, independent opponents are optimal, i.e. they will win a match where possible and draw otherwise. Similarly, we make conservative assumptions for colluding players, namely that they will never lose, but also will never win against independent players. If every player was optimal, then each will gain the same number of points. However, this assumes that every player plays as well as possible all of the time. Where some colluding players (*Chickens*) aim to draw against all players except that they lose to colluding players (*Foxes*), then Foxes will get more points than would otherwise be possible.

In a competition, let us assume there are x Optimal players and c Chickens colluding with f Foxes whom the Chickens want to win. A match between an Optimal player and a Chicken, or between two Chickens, will result in a draw since the Chicken will play to draw. However, a match between a Fox and a Chicken will result in a win for the Fox, since the Chicken will recognise that it is playing a Fox. A win will gain the winner p_w points, a draw p_d points, and a loss p_l points (as noted above, $p_w > p_d > p_l$). We assume each player will also compete against himself and draw. This is summarised in Table 1.

In this competition, each of the x Optimal players will get $p_d x + p_d c + p_d f$ points, each Chicken will get $p_d x + p_d c + p_l f$ points, and each Fox will get $p_d x + p_w c + p_d f$. It can then be seen that under these assumptions a colluding player will score higher in the competition than the Optimal player since $c \geq 1$.

2.2 Knockout Tournaments

For knockout tournaments, the impact of collusion is much less than for league tournaments. The result of a match must be a win for one player so as to decide who will continue to the next round. Typically this will require some kind of tie-breaking system, such as the penalty shootout in soccer.

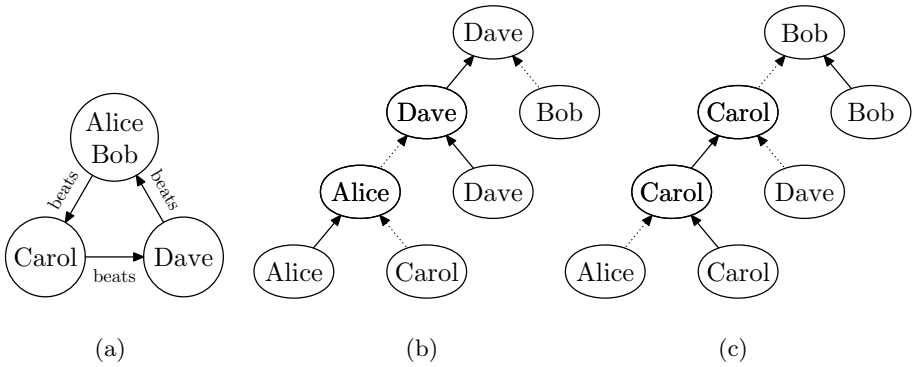


Fig. 1. Knockout tournament collusion example

The only way for a player to win in all arrangements of initial matches is if he can beat all other participants. Likewise if a player can beat all other players then he will win the competition regardless of the initial arrangement. However, it may be advantageous for a player to influence the arrangement of initial matches if there are cycles in the directed graph of game results, for example Figure 1(a). Here Alice and Bob are equivalent players, who both can beat Carol but will be beaten by Dave. Also Carol can beat Dave. In the scenario shown in Figure 1(b), if Alice plays as well as possible, then while Alice will win the first round she will be eliminated by Dave in the next round. Then Dave will eliminate Bob and go on to win the tournament. However, if Alice and Bob collude then the result can be as shown in Figure 1(c), allowing Bob to win. Alice can deliberately lose the first match and so Carol will go through. In the next round, Carol will eliminate Dave but in the final round Bob can beat Carol. This example shows that there are cases where, if a player is colluding with others in a knockout tournament, it may be in the best interest of the collusion group for one member to play less well than is possible.

Unlike the league tournament, it is clear that the result of a match between co-colluders does not contribute to the final result, if we assume that all players colluding with each other have equal abilities. However, in situations like those described above, it is useful for a colluder to lose against an opponent who possesses an ability that the colluders do not. We do not explore this further, and in the rest of this paper we concentrate on league-like tournaments.

3 Authentication Mechanisms

To manipulate a knockout tournament it is necessary for the abilities of the opponents to be known in advance, however, in a league all that is necessary is for colluding players perform normally against independent opponents, but selectively play poorly against other colluding players.

In order to identify a colluding player when the order of games is not known, there must be some form of authentication that happens before or during each game. This should be reliable and must identify the case where one player must lose before the result of the game is decided.

It may be the case that communication is easy, for example in a face-to-face game the players may recognise each other or be allowed to speak to each other. If the players are computer programs (the case which the rest of this paper will concentrate on), a standard program-to-program authentication can be accomplished.

However, there may be times when an overt channel is either not possible because of the constraints of the competition or not permitted by the competition rules. In these situations, a covert channel can be used. There are a variety of techniques developed for such communication channels, however, the majority of them are described in the literature for the analysis of multi-level secure computer systems (many of which are summarised in the "Light pink book" [4]), so while not directly relevant, they can be modified for use within games.

3.1 Timing

In the literature on multi-level secure systems, one frequent way to create a covert channel is for a program to signal to another by varying some kind of system-wide property. For example, this could be modifying the CPU load [5], hence changing scheduling patterns, or it could be modifying timing of acknowledgements of messages which may flow in only one way [6]. These techniques could be used directly, but there are also timing based covert channels that are specific to games.

One such channel would be to use the timing of moves to carry information by causing the sender to delay making a move and the recipient to measure this delay. Such schemes are easy to create and can have a relatively high bandwidth. However, if the transport mechanism is affected by latency and/or jitter, then this covert channel may be unreliable or even eliminated completely.

Where the latency is fixed, this can be easily cancelled out, but jitter is more problematic. If the jitter is sufficiently small, then it can be removed, at the cost of reducing bandwidth. However, the rules are likely to place an upper bound on the maximum time to make a move, and so fix the maximum possible delay. If the jitter is of similar magnitude to this limit, then the bandwidth of the channel will be very small. If the CPU time to make a move is limited by the competition rules rather than wall clock time (the amount of time to have passed in the real world), then the maximum delay can be fairly large, since in most operating systems the time that a program is paused is not counted towards the CPU time.

One form of jitter specific to a competition is if the time for a move to be sent is fixed to a value greater than the maximum allowable time for the delay. This may occur if the competition is to be shown live and the organisers wish to slow the competition to a speed that humans can watch. If this is done, then the move timing covert channel would be eliminated.

3.2 Choice of Equivalent Moves

The timing based mechanisms mentioned above are possibly unreliable in the presence of jitter. An alternative to this is to encode the authentication data in the moves themselves. In person-to-person games, this could be, for example, the way the pieces of a board game are held, or the place in which a card is put down in a card game (this is why there are complex physical arrangements in Bridge tournaments to prevent such communication). In contrast, for the case of an online competition the move will likely be expressed in an unambiguous form hence will allow no extra information to be carried in a side channel.

At a stage in the game, if there is more than one move which can be shown to not change the outcome of the game when compared to the best move, then this fact can be used to transmit information. One possible way for this to be achieved is by ordering the n equivalent moves. The order chosen can be arbitrary, but often there is an obvious solution, for example in the Connect-4 situation described in Section 4, ordering moves by column number would be sensible. In order to send $r \in \{1, \ldots, n\}$ then the rth move is chosen. After receiving a move from its opponent, a player can identify which move, out of the opponents possible moves, was chosen and hence identify r.

3.3 Analysis of Authentication Mechanisms

In order for a collusion strategy to succeed, a reliable covert channel must be established to allow a Chicken to identify when it is playing a Fox and thus should deliberately lose.

For the simple case where a Chicken needs to identify whether its opponent is a Fox or not (Section 2.1), the goal of the channel can be viewed as being able to transmit a single bit while the result of the game is still undetermined. While the required capacity of the channel is low, the reliability requirements are high, since a false positive will result in a Chicken losing to an independent opponent and so reduce the chance of the Fox winning.

Much research on bandwidth estimation of covert channels, for example [7], has concentrated on finding upper bounds for the data rate of the channels. These techniques can be used to design a coding system which approaches these upper bounds.

In the case where the timing information is used for authentication, it is possible that the communications channel will modify the meaning of the information being sent. However, where the move itself carries the information it is reasonable to expect that the signal will be received intact. For this reason a message sent using this covert channel will always be received correctly. This is in contrast to the timing channels, where interference from other processes on the machine could corrupt the signals.

However, this does not mean that the channel is noiseless, since the receiver cannot differentiate between the case where information is being sent, and the case where the moves carry no meaning (this is also the case for timing channels).

The moves of independent players are analogous to noise in communications theory. The situation is similar to low-probability-of-intercept spread-spectrum radio in that the "amplitude" of the signal cannot be any more than the noise (a particular move is either made or not, there is no concept of "magnitude").

In order to reliably transmit a single bit of information, a technique based on frequency-hopping can be used. For each move, the number sent is chosen according to a keyed generator. The receiver shares the key and so knows what move to expect from a colluding player. If, after a number of moves, the receiver has found that the opponent has made every move as expected, then it can assume that the opponent is colluding with it and act accordingly. The confidence level of the decision being correct can be increased by increasing the number of possibilities at each move or by increasing number of moves before a decision is made. While waiting longer before making a decision is preferable, if the player waits too long, then by the time a decision is made, it is no longer possible to change the game result.

3.4 Authentication Key

The goal of the generator is to distinguish itself from the "background noise" of other players. Where little or nothing is known about the game strategies of independent players, it is difficult to make any assertions about the characteristics of the noise. For this reason, it may be safe to assume that at each turn every move is equally likely — analogous to white noise. This assumption is particularly useful since it greatly simplifies the design of the generator, and allows a fast implementation so as to reduce CPU usage (which may be a factor in deciding a winner).

For spread-spectrum radio, typically a cryptographically secure pseudorandom number generator, such as a stream cipher, is used. In the case of spread-spectrum radio the transmission is effectively public but in a game the moves are typically only seen by the opponent. One threat in spread-spectrum radio is an adaptive adversary, whereas in a game the opponents may not be changed during the competition. When coupled with the fact that other opponents are probably not aware of the collusion strategy, it is reasonable to assume that cryptanalytic attacks are unlikely. Again, this assumption simplifies the design of the generator and so reduces processor time requirements.

The only goal of the generator is to appear different from a white noise source so a repeating constant could be used, such as always picking the first move. However, it is feasible that an opponent could accidentally pick the same strategy. A small change can be made where the move chosen depends on the stage in the game. For example r could simply be the result of a pseudorandom number generator (PRNG) seeded by a shared secret. This simple authentication system could also be used with the timing based covert channels. A linear congruential PRNG is very fast and simple, and with well chosen parameters [8, Section 3.2.1] meets all the requirements (assuming no cryptanalytic attacks).

4 Real World Example

The above techniques were developed for and used with the Cambridge University Computing Society (CUCS) Winter Competition [9]. This was a programming competition where entrants submitted one or more programs which played a variant of Connect-4. These programs then played against each other and a winner was decided.

4.1 Rules of the Game

As with normal Connect-4, the game is played on a 7×6 board. Each player takes turn to choose a column and places his token at the lowest free square. The first player to have four tokens in a row, either horizontally, vertically or at a 45° diagonal, wins the game. In the standard game, a player must place exactly one token at each turn, but in the variant used in the competition, the player also has the option to pass. This change was made so that standard Connect-4 strategies would not work and thus force entrants to come up with their own techniques. However, an unforeseen result of the modification to the rules was that the possibility of a guaranteed winning strategy was eliminated, regardless of whether the player makes the first move, since a move cannot be forced.

The competition was split into two stages, a league followed by a knockout tournament. The league proceeds by every entered program being played against every other entered program. Each match consisted of six games, with each player alternately starting first. The winner of the match was the player with the most number of wins and was awarded two points. If both players had an equal number of wins in the match, then each player is awarded one point.

The five programs with the highest scores in the league were selected for the knockout tournament. Firstly, the fourth and fifth programs were played in a match of six games as in the league. However, if this match was a draw, then the winning program would be the one with the least CPU usage, and if that was equal, then memory usage and finally code size were considered. Then, the remaining four programs were played in a standard knockout tournament, with each match following the rules for the fourth/fifth playoff, i.e. fourth/fifth vs. first, second vs. third, and finally the winners of the previous two matches.

4.2 Collusion Strategy Chosen

In this competition, overt communication was not permitted in order to prevent programs communicating with more able humans or more powerful computers. Also, the only information that a program received from its opponent was the move number, in ASCII, so there was no redundancy in the encoding. However, the rules did not explicitly prohibit collusion between opponents. For these reasons a covert channel was required for communication, but it would not break the rules. There were plans for the final stages of the competition to be run live so there was a possibility of jittering timing information, even unintentionally.

Table 2. Summary of winners in matches between Fox, Chicken, Rooster, Rabbit and Optimal players ("—" denotes a draw)

	Fox	Rooster	Chicken	Rabbit	Optimal
Fox	—	Fox	Fox	—	—
Rooster	Fox	—	Rooster	—	—
Chicken	Fox	Rooster	—	—	—
Rabbit	—	—	—	—	Optimal
Optimal	—	—	—	Optimal	—

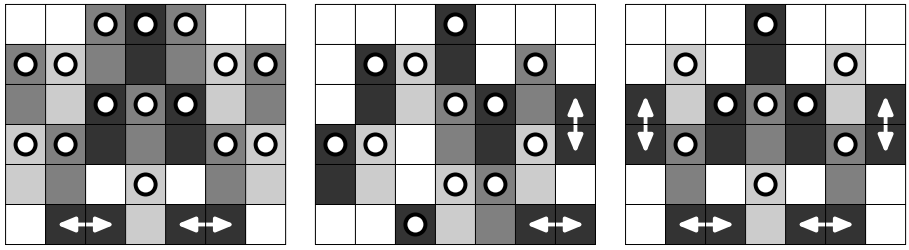
Because of the advantages in reliability and simplicity of the *Choice of Move* covert channel described in Section 3.2, this was used for communication.

One refinement to the authentication method described in Section 3.4 was rather than having only two types of colluding player (the Fox and the Chicken, where a Fox always wins against a Chicken), three were used. The additional category, *Rooster* would beat a Chicken but would be beaten by a Fox (see Table 2). This was because collusion is ineffective in the knockout stage, so the only way to win was for all five participants to be our colluding players. This could be achieved by having five Foxes and the rest Chickens, but there remained the risk that another independent player would get into this stage (due to *Rabbits*, the category which will be introduced in Section 4.6). Since, by applying the strategy described in Section 4.3, our players will never lose, CPU usage would be the decider and so this should be optimised. Hand optimising a program is time consuming so it is preferable to minimise the number of programs that this needs to be done on. If only one of the five Foxes was optimised, then there is the risk that another will knock it out of the tournament before it has a chance to play the independent player. To mitigate this risk, two optimised Foxes were entered, along with four Roosters, so the optimised Foxes would be guaranteed to play any remaining independent players. Two Foxes were entered to reduce the impact of any programming errors. This reduced the number of points given to the Roosters and Fox slightly, but it was decided to be worthwhile.

4.3 Game Strategy

In order for collusion to be feasible, it was necessary to have a strategy which guaranteed a draw in every game. It was also desirable to design the strategy such that the all outcomes of the game remain possible for as long as feasible, so that the decision as to whether to lose or not can be delayed. Finally, so as to optimise the bandwidth of the covert channel, the number of possible moves at each turn should be maximised.

We developed a very efficient strategy which allowed a draw to be forced, regardless of who made the first move. This was in contrast to the non-pass version of Connect-4 where a strategy [10] exists which guarantees a win if used by the player who starts and almost never loses when if he plays second.



(a) Simple pattern (b) First player (c) Second player

Fig. 2. Possible board patterns used for the game strategy

Our strategy relies on finding a subset of the squares on the board, such that every winning line must pass through at least one of these, and preventing the opponent from occupying any of them. We achieve this by designing a pattern of non-overlapping rectangles on the board as shown in Figure 2(a).

If the opponent plays on the bottom square, then our player plays on the top square. Our player never plays on the bottom square. Therefore, the opponent can never occupy the top square.

If the opponent plays on one of the squares, then our player plays on the other. Therefore, the opponent can never occupy both squares.

If our player moves first, then it plays on this square, thereby preventing the opponent from occupying it.

Three possible patterns are shown in Figure 2. The different shades of grey have no semantic meaning; they are used only to differentiate the rectangles from each other. Since the rectangles do not overlap, the strategy forces our player to play on at most one square per move, thereby guaranteeing at least a draw.

4.4 Implementation

The competition allowed ten entries per person and three people entered from our research group. While the rules explicitly stated that it was permitted to implement an algorithm developed by someone else, using someone else's code was not allowed. For this reason each member of the group entered a program written independently in a different language.

As intended, no players lost other than times when it was designed to lose against another colluding player. While there was some risk that this (false positive) could have happened by accident, the design of the covert channel reduced this to an acceptable level. As shown in Figure 3, after ten moves (the

point at which a decision was made) the number of possible move sequences ranged between 960 and 5760. Therefore, even if an opponent happened to choose an identical game strategy, the probability of a false positive was at least 1 in 960 (subject to previous assumptions). In contrast, the risk of a false negative (that one colluding player who should lose to its colluding opponent, fails to identify in time) can be reduced to the risk of programming error. This is because the covert channel used can be assumed to introduce no noise. Furthermore, for deterministic players, all possible games between colluding opponents can be exhaustively tested in a reasonable time, before entry to the competition.

4.5 Optimisation

The final stage of the competition would take CPU usage into account so there was a potential advantage to optimise the Foxes. Aside from standard code efficiency improvements, one domain specific optimisation was to remove all detection code from the Foxes. The simplification was feasible since it was not necessary for a Fox to identify that it is playing a colluding player, as the responsibility for the match result can be given to the losing player. To achieve this a player who has identified that it must lose continually passes until the game has ended. Additionally no evidence of collusion can then be found by inspecting the source code of the Foxes.

To ensure the game will result in a win for the Fox when the Chicken passes the game strategy must be changed slightly. Firstly, the Chicken must start losing sufficiently early in the game such that it is still possible to lose. Secondly, a different pattern must be used for the player starting first and the player starting second. This is because both players having the same pattern would result in them drawing the game by default after playing four passes before the authentication could be completed. Thirdly, more flexible patterns (Figure 2(b) and Figure 2(c)) give the players more equivalent moves, thereby increasing the reliability of the authentication procedure.

4.6 Rabbits

In the simple example of Optimal players and colluding players, it was seen that only one Chicken was necessary for the Fox to win, however, the situation is not so simple when not all independent players are Optimal. That additional worst-case category of players (so as to find a lower bound) encountered in practice is a *Rabbit*, which will play poorly, so lose to Optimal players, but draw with everyone else. From Table 2 it can be seen that an Optimal player will act as if it is colluding with any Rabbits in the tournament. Therefore the only way to win the tournament is to have greater number of Chickens than there are Rabbits, no matter how many Optimal players exist. While it was likely that several approximately Optimal players would be entered, it was hoped that there would be a small number of people who would enter a player that would play so badly that the chances of winning would be low.

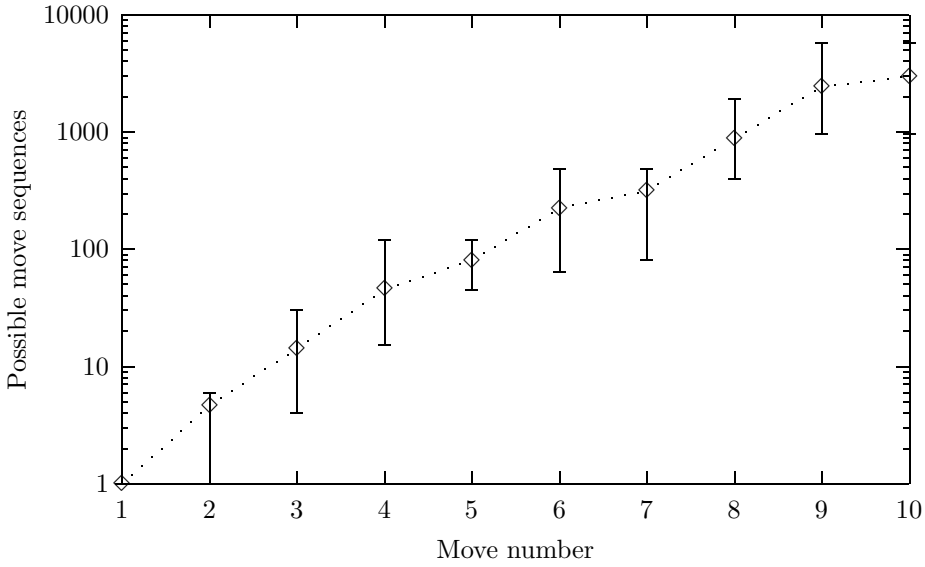


Fig. 3. Number of possible move sequences after a given number of moves. Three classes of colluding players were used so for each move number, the lower limit, mean and upper limit of the nine possible matches is plotted

4.7 Results

A summary of the final league table is shown in Table 3.

Since the algorithm used by the Fox, Rooster, and Chicken would only win in exceptional circumstances, the actual results for colluding players in the competition were very similar to the worst case scenario estimates. Some players appeared to play randomly, so when played against programs using a tree-searching algorithm the tree-searching algorithm won. This behaviour approximates the expected results from ideal Rabbits and Optimal players, so the random players are classed as Semi-Rabbits and the tree-searching players are classed as Semi-Optimal. However, as expected only six Semi-Rabbits were entered by other participants and 28 Chicken/Roosters were entered by our group, so we won the competition with a safe margin of 30 points.

5 Further Work

The above example dealt with the case where neither non-colluding participants nor the competition management expected collusion to be used. In the case where collusion is expected and not desired, there are interesting possibilities for preventing collusion from being effective.

Table 3. Summary of results at end of league stage. Players are ordered in descending order of points

No	Category	Won	Drew	Lost	Points
1	Fox	58	26	0	142
2	Fox	58	26	0	142
3	Rooster	51	29	4	131
4	Rooster	49	31	4	129
5	Rooster	49	31	4	129
		cut-off point			
6	Rooster	48	32	4	128
7	Semi-Optimal	16	67	0	99
⋮	⋮	⋮	⋮	⋮	⋮
13	Semi-Optimal	12	64	8	88
14	Chicken	3	69	12	75
⋮	⋮	⋮	⋮	⋮	⋮
37	Chicken	0	72	12	72
38	Semi-Rabbit	4	63	17	71
⋮	⋮	⋮	⋮	⋮	⋮
43	Semi-Rabbit	1	52	31	54

5.1 Collusion Resistant Competitions

In order to prevent collusion, the competition could be designed such that collusion provides no advantage. During discussion of the problem one observation made was that the problem of deciding a winner in the competition is similar to the problem of electing a candidate in an election. While there are some differences, for instance, that the number of candidates is identical to the number of voters, there are also many similarities.

One possibility investigated was of a game tournament similar to the Single Transferable Vote (STV) system. Here, every player plays every other player, in a similar fashion to a league tournament. However, the winner evaluation is more complex. At each stage, the normal league rules are applied and an ordering established, but then the players with the lowest score are eliminated, along with their contribution to all other players' scores. The process is repeated until no more players can be eliminated.

This system has the advantage that Chickens will be eliminated before Foxes, so the Chickens' scores can have no effect on the final result, however, they can control the order in which players are eliminated so it is not clear that this system is free from manipulation. Additionally, the number of "voters" is identical to the number of "candidates" so the final stage will likely result in more than one winner. This was confirmed by running the results of the above example competition through this algorithm. As expected, all the Chickens were

eliminated but the final result included the Foxes and all the Semi-Optimal players. Since all these players will draw against each other, deciding a winner is difficult.

Not only should competitions be resistant to collusion but they should be fair and this is a very difficult quantity to measure. There are a variety of proofs which state, given certain assumptions, that it is not possible to design an *ideal* election. These include Arrow's theorem [11], Gibbard-Satterthwaite [12, 13] and Gärdenfors' extension [14]. These primarily deal with manipulation by voters, but there has been some work on manipulation by candidates, such as a general result in [15] and an analysis of the particular case where the election is made out of a series of pair-wise comparisons in [16]. These state that, given certain assumptions, non-dictatorial elections are manipulable by candidates deciding whether or not to participate in the election. This result is not directly applicable since it assumes that each candidate who votes will vote himself the highest, and the stronger version of the result also assumes that no candidates vote. However it may still be partially applicable. Whether these theories imply that an *ideal* competition is impossible depends on a formal definition of fairness and collusion resistance, which is outside the scope of this paper.

5.2 Detecting Collusion

In some games, it may not be desirable or possible to re-arrange the competition to make collusion infeasible. In these cases, the only alternative may be to detect collusion and eliminate players if caught. For example, an expert could examine the match results [17], and in a similar way that a Bridge expert would look for players being exceptionally lucky in a tournament, an expert suspecting collusion would look for players being exceptionally unlucky. The expert could also monitor the games in progress looking for an suspicious changes in apparent skill. If a player is aware of such monitoring, then countermeasures to both techniques could be taken.

6 Conclusion

In this paper, we show that collusion can offer significant advantages in tournaments which are based around leagues. We present a simple algorithm for acting on the basis of authentication information which will guarantee winning a competition, assuming only one team is using a collusion strategy and the standard of players is good. We also introduce a covert channel built using only redundancy in the moves of a game and show how this can be used to authenticate colluding players. We demonstrate these techniques being successfully applied in order to win a real world competition. Finally, options for resisting and detecting collusion are explored, including drawing parallels between the design of competitions and the design of elections.

7 Acknowledgements

Thanks are due to Phil Cowans, John Fremlin, Ian Jackson, Matt Johnson, Stephen Lewis, Andrei Serjantov, and Hanna Wallach for their helpful contributions, and to Microsoft for donating an X-Box as the prize for the competition. We also would like to thank the anonymous reviewers for their suggestions.

References

[1] American Contract Bridge League: Law 20. Review and Explanation of Calls. (1997) in Laws of Duplicate Contract Bridge (American Edition).
[2] American Contract Bridge League: Laws of Contract Bridge (Rubber Bridge Laws, American Edition). (1993)
[3] Winkler, P.: The advent of cryptology in the game of Bridge. Cryptologia **7** (1983) 327–332
[4] Gligor, V.D.: DoD NCSC-TG-030 A Guide to Understanding Covert Channel Analysis of Trusted Systems (Light-Pink Book). National Computer Security Center (1993)
[5] Huskamp, J.C.: Covert Communication Channels in Timesharing System. PhD thesis, University of California, Berkeley, California (1978) Technical Report UCB-CS-78-02.
[6] Kang, M.H., Moskowitz, I.S.: A pump for rapid, reliable, secure communication. In: 1st ACM Conf. on Computer and Communications Security, Fairfax, VA, Center for High Assurance Computer Systems (1993) 119–129
[7] Millen, J.K.: Finite-state noiseless covert channels. In: Proceedings of the Computer Security Foundations Workshop, Franconia, New Hampshire (1989) 81–85
[8] Knuth, D.E.: The Art of Computer Programming. Third edn. Volume 2, Seminumerical Algorithms. Addison-Wesley (1998)
[9] Cambridge University Computing Society: Winter programming competition (2002) http://www.cucs.ucam.org/competition.html.
[10] Allis, L.V.: A knowledge-based approach of connect-four. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands (1988) ftp://ftp.cs.vu.nl/pub/victor/connect4.ps.Z.
[11] Arrow, K.J.: Social Choice and Individual Values. Second edn. Yale Univ Press (1970)
[12] Satterthwaite, M.: Strategy-proofness and Arrow's condition: Existence and correspondence theorems for voting procedures and social welfare functions. Journal of Economic Theory **10** (1975) 187–217
[13] Gibbard, A.: Manipulation of voting schemes: a general result. Econometrica **41** (1973) 587–601
[14] Gärdenfors, P.: Manipulations of social choice functions. Journal of Economic Theory **13** (1976) 217–228
[15] Dutta, B., Jackson, M.O., Breton, M.L.: Strategic candidacy and voting procedures. Econometrica **69** (2001) 1013–1038
[16] Dutta, B., Jackson, M.O., Breton, M.L.: Voting by successive elimination and strategic candidacy. Journal of Economic Theory **103** (2002) 190–218
[17] Yan, J.: Security design in online games. In: 19th Annual Computer Security Applications Conference, Acteve (1993)