Cryptographic Blackjack

Alvin AuYoung and Christopher Tuttle

CSE 207: Final Project, Spring 2004
University of California, San Diego
{*alvina*, *ctuttle*}@cs.ucsd.edu

Abstract

Internet casinos have become a billion dollar industry [21]. The increasing popularity of online gaming is surprising given its weak guarantees of fairness compared to those offered by physical casinos. We apply a bit commitment protocol to an online blackjack game that provides strong fairness guarantees between the player and casino without compromising the play of the game. We introduce a set of experiments that capture the fairness guarantees of the protocol, and describe how this protocol can be extended to other online games.

1 Background

Casino games have outcomes with well-known probability distributions. Most people who partake in these games are somewhat familiar with these statistics and play based on this knowledge of expected outcomes. However, these expected outcomes are based on the assumption that there exists a source of true randomness determining the outcome of the games. For example dice are expected to be equally weighted on all sides so that there is an equal probability of landing any of its sides, and card games are expected to have a randomized ordering of cards (after a shuffle) such that the probability of drawing any particular card is uniformly distributed between all cards in the deck. There are several issues that make this randomness difficult to confirm in casinos. Manipulation of this randomness more difficult to detect in Internet casinos.

1.1 Fair Gaming

All casino games are based on some sort of randomness. A casino game is fair if the randomness is unbiased. One can expect fair game playing when gambling in a casino, and therefore, a well-known distribution of outcomes over a large number of trials (game plays). However, it is likely that during play, there will be seemingly improbable events over a small sample of trials. In order to believe that the game is fair, there are two existing ways in which this trust can be established: physical auditing and the use of a third-party.

1.1.1 Physical Auditing

A player is more likely to be able to differentiate between a streak of luck (improbable outcomes) and unfair play in a casino game if she can inspect the physical actions that determine the outcome. For example, if the rolling of a pair of dice consistently lands a pair of 7s, she might begin to suspect unfair play. However, her physical presence at the casino would allow her to inspect the dice and watch the rolling of the dice to allow her to differentiate between *luck*, and unfair play. Similarly in a card game, observing the shuffling of the deck in a card game provides some confidence that the proper protocol for fair game playing is being followed.

1.1.2 Third-Party Auditing

There exist third-parties that verify the fairness in casino games. For example, an excerpt from the mission statement of the National Indian Gaming Commission (NIGC) [6] states: *"Our Mission...to assure that gaming is conducted fairly and honestly by both operators and players..."*. The NIGC performs audits of casinos under its jurisdiction and has the authority to pursue legal actions against such casinos. Similarly, the Nevada Gaming Commission (NGC) [7] performs audits and enforces fair gaming laws on all casinos operating in Nevada. The NIGC and NGC are examples of well-known third-parties whom casino game players can trust to keep the casinos honest.

1.2 Internet Casinos

Internet Casinos do not allow the possibility of physical auditing, and therefore rely on third-party auditing. Due to lack of legislation of online gaming [21], these casinos do not fall under the jurisdiction of any body of gaming commissions. Because of this, there are no well-known third-parties like the NIGC or NGC that can vouch for the fairness of Internet gaming. At best, online casinos have their source code audited and verified by independent third-parties in order to prove that the game is fair. Organizations like Fairbet [9] provide reviews and audits of online casino sites. However, most online casinos simply make claims about having fair and tested source code and random-number generators [4], [15]. The most common form of guarantees that Internet casinos provide is in using SSL to provide privacy for its players [11].

1.3 Problem Statement

The lack of physical auditing and trusted third-party auditing in Internet casinos provide few assurances to the end-user about the fairness of online games. In particular, games with a relatively small casino advantage provide an obvious opportunity to bias the outcome of a game. For example, consider the statistical casino advantage of a blackjack game. In the case of a single-deck game, the advantage of the house (the probability that the house wins) is .01[14]. To double its earnings, the casino would merely have to cheat a tenth of a percent of the time, which it could conceivably do very easily without being detected. With no possibility of physical auditing, the player cannot see the deck being stacked a tenth of a percent of the time. It is also very difficult for an end-user to establish any trust with a third-party that audits an online game's source code. We argue that it is possible, with the use of cryptography, to provide very strong fairness guarantees to particular online games that are perhaps even stronger than those in physical casinos.

2 Bit Commitment

Bit commitment is a well-known protocol where one party can commit to some bit value without revealing it to another party, and later confirm what the original bit was without changing her mind. For example, consider the case where Alice and Bob would like to decide on a number, 1 or 0. Alice and Bob each decide on a bit, and the number is the XOR of the two bits. The hope is that neither Bob nor Alice can control the outcome of the bit. However, if Bob were to tell Alice his bit first, then Alice could change the bit that she had chosen to determine the value of the resulting bit. The use of bit commitment allows Bob to commit to a bit, without revealing it to Alice, and then unveil it later to determine the resulting bit. For example, if Bob were to write the value of the bit in a sealed envelope, and hand it to Alice, she can openly reveal her bit, and they can open the sealed envelope afterwards to determine the resulting bit. In this section, we discuss two forms and applications of the bit commitment protocol.

2.1 Fair Coin Flipping

Variations of the protocol have been extended to protocols for fair coin flipping and mental poker [18]. One-way functions are often used to create coin-flipping protocols. For example, if Alice and Bob can agree on a one-way function, the protocol is as follows [19]:

1. Alice choose a random number, x. She computes $y = f(x)$, where f is the one-way function.
2. Alice sends y to Bob.
3. Bob guesses whether x is even or odd and sends this guess to Alice.
4. If Bob is correct, the coin flip is *heads*, otherwise, it is *tails*.
5. Alice announces the result of the coin flip and sends x to Bob.
6. Bob confirms that $y = f(x)$.

The security of the protocol rests on a few properties of the one-way function. For example if Alice were able to find x and x' such that $f(x) = f(x')$, and x is even and x' is odd, she can always determine the outcome of the coin flip by sending back either x or x' to Bob in the penultimate step of the protocol. Also, if the function f is known to have some distribution of outputs, such as $f(x) = 2x$, then Bob can determine the outcome of the coin flip. Finally, if the function f is invertible, meaning from y, Bob can guess likely candidates for x, he has an advantage in determining the output of the coin flip. We discuss the fairness properties of our variation of this protocol later.

2.2 Mental Poker

A variation of fair coin flipping protocol uses public-key cryptography. We describe its use in allowing Alice and Bob to play a game of Poker [18]. The protocol works as follows:

1. Alice produces 52 messages representing each card in the deck, M_i for $1 \leq i \leq 52$, encrypts them using her public key, and sends them to Bob.
2. Bob chooses five messages at random, encrypts them with his public key and sends them back to Alice.
3. Alice decrypts the five chosen messages and sends them back to Bob, who decrypts them to determine his hand.
4. Bob chooses five more messages at random (different from his), and sends them to Alice, and those will determine her hand.

During game play, additional cards can be dealt similarly. After the game play, both Alice and Bob can reveal their key-pairs to confirm that neither side cheated. It has been shown that this particular protocol can leak information if the RSA public-key algorithm is used. [10], [8]. There have been other protocols for online poker, but these more secure protocols can also require too much overhead to implement in a practical setting.

3 Algorithm and Analysis

In this section we describe our protocol and introduce a set of experiments that capture the fairness guarantees of our protocol. The algorithm used in the protocol is a variation of the bit commitment protocol described in the fair coin flipping application described above. We chose this protocol because it requires less communication and computation overhead in the setting of an interactive online game, and doesn't require public and private keys for the participants.

The algorithm is specifically used for online blackjack (see Appendix for description of the rules of the game). We discuss its applicability to other online games in Future Work.

3.1 Algorithm

Server		**Client**
$ss \xleftarrow{\$} \{0,1\}^{128}$		
$commit \leftarrow \text{SHA1}(ss)$		
send($commit$)	$\Rightarrow$	recv($commit$)
		$cs \xleftarrow{\$} \{0,1\}^{128}$
recv(cs)	$\Leftarrow$	send(cs)
$seed \leftarrow cs \oplus ss$		
$coins \leftarrow seed$		
Play the blackjack game, drawing cards as follows: When a card is to be dealt, pick the next 6 bits from *coins*, and if that card has not been dealt then deal it. Otherwise, pick again. In this particular game, *coins* is of adequate length, but for other games, $\|seed\|$ can be increased from 128 to the appropriate length. Play until the game terminates.		
send(ss)	$\Rightarrow$	recv(ss)
		$seed \leftarrow cs \oplus ss$
		If $commit \neq \text{SHA1}(ss)$ then output "Server Cheated."
		$coins \leftarrow seed$
		Deal the same number of cards as were dealt in the game in the same manner as above. If the cards have an identical sequence to those drawn in the game, output "Game Verified." Otherwise, output "Server Cheated."

3.2 Analysis

We would like to capture the notion that neither the client nor the server can get an advantage over the other by affecting the random output of the blackjack game, BJ. To do this, we must show that (1) the server cannot affect the output from its choice in seed, (2) the client cannot affect the output, even if it has the server's commitment, and (3) the server cannot change its mind about the seed (and thus affect the output) after the client has sent in its seed. We shall quantify these as three experiments and then show how our algorithm admits small advantages in these experiments.

3.2.1 Server Seed Experiment

Definition 3.1 *We define the* ***Server Seed*** *experiment and its associated advantage function as follows:*

Experiment $\mathbf{Exp}_{BJ}^{\text{srv-seed}}(A)$:

$target \xleftarrow{\$} \{0,1\}^{128}$

$ss \xleftarrow{\$} A(target)$

$cs \xleftarrow{\$} \{0,1\}^{128}$

$seed \xleftarrow{\$} cs \oplus ss$

If $(seed = target)$ then return 1

Otherwise, return 0.

The **srv-seed** *advantage of A is defined as*

$$\mathbf{Adv}_{BJ}^{\text{srv}-\text{seed}}(A) \quad = \quad \Pr[\ \mathbf{Exp}_{BJ}^{\text{srv}-\text{seed}}(A) = 1\ ]$$

■

It is easy to see that the **srv-seed** advantage for the given algorithm is very low. Experiment 1 corresponds to the server's initial selection of a random seed. The client seed is chosen randomly, so the final seed, as the bitwise XOR of the two seeds, will be uniformly distributed over $\{0,1\}^{128}$.

Lemma 3.2 *Let A be a boolean-valued random variable with arbitrary distribution. Let B be a boolean-valued random variable with a uniform distribution. Let X be the exclusive-or of A and B. Then the distribution of X is uniform.*

Proof:

$$\begin{aligned}
\Pr[\ X = 1\ ] &= \Pr[\ A = 0 \wedge B = 1\ ] + \Pr[\ A = 1 \wedge B = 0\ ] \\
&= \Pr[\ A = 0\ ]\Pr[\ B = 1\ ] + \Pr[\ A = 1\ ]\Pr[\ B = 0\ ] \\
&= \Pr[\ A = 0\ ](\frac{1}{2}) + \Pr[\ A = 1\ ](\frac{1}{2}) \\
&= \frac{1}{2}(\Pr[\ A = 0\ ] + \Pr[\ A = 1\ ]) \\
&= \frac{1}{2} \\
\Pr[\ X = 0\ ] &= 1 - \Pr[\ X = 1\ ] \\
&= \frac{1}{2}
\end{aligned}$$

■

Proposition 3.3 *For any adversary A, the* **srv-seed** *advantage of A is* $\leq \frac{1}{2^{128}}$.

Proof: When the server's random seed is XOR-ed with the client's seed, which is a random string, each output bit will have a uniform distribution, so the entire string will be uniformly distributed over $\{0,1\}^{128}$.

$$\Pr[\ seed = target\ ] \quad \leq \quad \frac{1}{2^{128}}$$

■

3.2.2 Client Seed Experiment

Definition 3.4 *We define the* **Client Seed** *experiment and its associated advantage function as follows:*

Experiment $\mathbf{Exp}_{BJ}^{\text{cli}-\text{seed}}(A)$:

$target \xleftarrow{\$} \{0,1\}^{128}$

$ss \xleftarrow{\$} \{0,1\}^{128}$

$commit \leftarrow \text{SHA1}(ss)$

$cs \xleftarrow{\$} A(target, commit)$

$seed \leftarrow cs \oplus ss$

If $(seed = target)$ then return 1

Otherwise, return 0.

The **cli-seed** *advantage of A is defined as*

$$\mathbf{Adv}_{BJ}^{\text{cli}-\text{seed}}(A) \quad = \quad \Pr[\ \mathbf{Exp}_{BJ}^{\text{cli}-\text{seed}}(A) = 1\ ]$$

■

Experiment 2 represents the step when the client selects its random seed. In this case the client has access to the server's commitment, and we would like the commitment to be of no use to the client. The client adversary has two options: either it can use the server commitment or it can ignore it. If it is ignored, then this is the same as the previous case, so the **cli-seed** advantage of the client is $\frac{1}{2^{128}}$. If the client uses the commitment then we want it to have no more than a small advantage. To gain any advantage the client must attack the one-wayness of SHA1. Since we believe SHA1 to be one-way, then we conclude that

$$\mathbf{Adv}_{BJ}^{\text{cli}-\text{seed}}(A)$$

is small.

Proposition 3.5 *For any adversary A there exists an adversary B such that*

$$\mathbf{Adv}_{BJ}^{\text{cli}-\text{seed}}(A) \quad \leq \quad \mathbf{Adv}_{SHA1}^{\text{ow}-\text{kk}}(B)$$

Furthermore the running time of B is the same as that of A.

Proof: The following adversary has the above properties:

Adversary $B(ciphertext)$:

$target \xleftarrow{\$} \{0,1\}^{128}$

$cs \xleftarrow{\$} A(target, ciphertext)$

$ss \leftarrow cs \oplus target$

Return ss

B uses A to invert a random ciphertext in SHA1. When A is successful in the context of $\mathbf{Exp}_{BJ}^{\text{cli}-\text{seed}}(A)$, it generates the plaintext ss such that $\text{SHA1}(ss) = ciphertext$. Thus

$$\mathbf{Adv}_{BJ}^{\text{cli}-\text{seed}}(A) \leq \mathbf{Adv}_{\text{SHA1}}^{\text{ow}-\text{kk}}(B).$$

We note that it is not feasible for the client to pre-compute all possible $\{0,1\}^{128}$ outputs from all possible 128-bit server seeds, since storing all these value would require more storage space than is possible on a computer today.

■

3.2.3 Verification Experiment

Definition 3.6 *We define the Verification experiment and its associated advantage function as follows:*

Experiment $\mathbf{Exp}_{BJ}^{\text{verify}}(A)$:

$(ss_1, state) \xleftarrow{\$} A()$

$cs \xleftarrow{\$} \{0,1\}^{128}$

$ss_2 \xleftarrow{\$} A(cs, state)$

If $ss_1 = ss_2$ then Return 0.

$c_1 \xleftarrow{\$} \text{SHA1}(ss_1)$

$c_2 \xleftarrow{\$} \text{SHA1}(ss_2)$

If $c_1 = c_2$ then Return 1, else Return 0

The **verify** *advantage of A is defined as*

$$\mathbf{Adv}^{\text{verify}}_{BJ}(A) \quad = \quad \Pr[\ \mathbf{Exp}^{\text{verify}}_{BJ}(A) = 1\ ]$$

■

Experiment 3 represents the verification process. Informally we want to ensure that the server used its originally chosen random seed to deal the cards, and that it dealt them properly. Verification, then takes on two parts. First the client verifies that the given seed matches the committed seed. Second, the client verifies that the server dealt the cards according to the random seed. We bound the former by the server's ability to find a collision in the hash function. Therefore, if we believe SHA1to be collision-resistant, then we believe that the advantage of any adversary in the Verification Experiment to be small.

Proposition 3.7 *For any adversary A there exists an adversary B such that*

$$\mathbf{Adv}^{\text{verify}}_{BJ}(A) \quad \leq \quad \mathbf{Adv}^{\text{cr1-kk}}_{SHA1}(B)$$

Furthermore the running time of B is the same as that of A.

Proof: The following adversary has the above properties:

Adversary B:

$(ss_1, st) \xleftarrow{\$} A()$

$cs \xleftarrow{\$} \{0,1\}^{128}$

$ss_2 \xleftarrow{\$} A(cs, st)$

Return (ss_1, ss_2)

B uses A to find a collision in SHA1. When A is successful in the context of $\mathbf{Exp}^{\text{verify}}_{BJ}(A)$, it generates a collision in SHA1. This is evident from the definition of the $cr1 - kk$ experiment. Thus $\mathbf{Adv}^{\text{verify}}_{BJ}(A) = \mathbf{Adv}^{\text{cr1-kk}}_{\text{SHA1}}(B)$. ■

SHA1is believed to be $cr2 - kk$ secure, so by Proposition 5.2 of the course notes on hash functions, it is also $cr1-kk$ secure. Thus we believe that our algorithm is secure with regards to the verification experiment.

3.3 Implementation

To demonstrate the feasibility of this algorithm, we implemented a pair of client and server programs to run the algorithm. The server program starts and listens on a network socket until a client program connects to it. The server and client play hands of blackjack according to the above algorithm until the client disconnects. The complete algorithm is executed for every hand, so every hand is verified independently.

The programs are written in $C++$ and communicate over network sockets. So that we could concentrate on the central issues to this project, we used SHA1 implementations from the OpenSSL 0.9.7*d* project, which provide correct and fast implementations of both algorithms.

In order to accurately describe how our random seeds are chosen, we must give some background on secure random number generation on typical Linux and BSD workstations.

Random numbers to seed the generators were selected by reading from `/dev/urandom` on Linux and FreeBSD machines. The `/dev/random` and `/dev/urandom` character devices are special files maintained by the kernel to provide random input for cryptographic algorithms. As the computer runs, it collects random input by timing the intervals between hardware interrupts, network packet arrival times, and disk request response times. It uses a few of the least-significant bits of these timings as random input to an "entropy pool", maintaining an estimate of how many random bits are in the pool at a given time. When `/dev/random` is read, a SHA hash of the entropy pool is returned, and the internal estimate of randomness is updated. The interested reader is directed at the source code to Linux [5], file `drivers/char/random.cc`; it contains a very helpful explanation of the algorithm, as well as the code that implements it.

The only difference between the two devices is that reading `/dev/random` can cause the caller to wait until there are enough bits of entropy in the system to service the request. When reading `/dev/urandom`, the entire request is serviced with however much entropy is available at the time. Though the former is important for very secure operations, such as key generation, we believe the latter is sufficient in our case.

4 Future Work

4.1 Applications to other games

The key point of this algorithm is to exploit cryptographic primitives to generate fair random numbers. The use of the random numbers was to generate cards for blackjack, but the technique can be extended and applied to many other games. Like blackjack, most casino games are based on some physical random source, whether it be the shuffling of a deck of cards, the tossing of some dice, or the selection of a numbered ball from a spinning cage. The casino's long-term monetary advantage comes from the interpretation of the random source – the rules and payoffs of the game – not from a fundamental bias.

In roulette, craps, and Keno, a stream of random bits generated by this algorithm could be used to select which slot the ball falls in (6 bits), what faces are shown on a pair of dice ($2 * 3$ bits), or what squares are selected on the Keno board ($20 * 7$ bits).

These algorithms can also be extended to handle multiple player scenarios where the players do not trust each other or the casino. Instead of having the single player send his or her seed to the server, the seed is broadcasted to all participants. The server broadcasts its commitment as well. The selected seed becomes the exclusive-or of all the seeds at the table. As long as the player selects a random seed for him or herself then the outcome will be random.

There are some multiple player examples to which this algorithm could not be applied directly, particularly those where the players compete between themselves and the casino. In these cases, the players may use some long-term strategy of deception and wagering to trick other players into believing the bluffers have a superior hand. The long-term success of bluffing depends on concealing the contents of the bluffer's hand, even after the game has finished. The random seed, and hence the entire deck of cards, is available at the end of our verification algorithm. An interesting area of future work would be to adapt our algorithm to such cases.

4.2 Disconnections and Fair Exchange

Though the above algorithm solves the problem of a server selecting non-random numbers when playing a game, there remain feasible ways for either side to cheat. The casino and the player communicate over the Internet, which is an unreliable medium. They can use this to cheat the other player.

Once the casino has received the random seed from the client, it can run the random number generator and look at the resulting cards. If the casino sees that the upcoming hand will not go well, the casino could disconnect from the client, blaming it on a crash or other failure beyond the casino's control. To increase the

fairness of the game, we would also like to ensure that neither party can benefit from cheating and blaming the communication medium.

4.2.1 Protecting Against Message Loss

One attack which could be used by the client would be to blame the communication medium for errors in its decisions to *hit* or *stay* while running a hand. It could send a *hit* message, wait to see what the card was, and then claim to the server that it never sent the message. Similarly, arguments could ensue over whether or not the server sent the verification message at the end of the exchange.

Some algorithms have been proposed which use a trusted third party to ensure non-repudiation of message transfers [22, 17]. These algorithms provide *non-repudiation of origin* and *non-repudiation of receipt*. The former guarantees that the sender cannot deny they sent the message, and the latter guarantees that the receiver cannot deny they received the message. At a high level, the algorithms in the papers above have both the sender and receiver sign an encrypted version of the message, and it uses a trusted third party to publish the key for the message.

4.2.2 Protecting Against Disconnections

To protect the fairness of the game, we want neither party to be able to benefit by disconnecting from the game when they feel as though it is going badly. If a disconnection occurs during the first few steps of the algorithm, before the server receives the client's random seed, then a disconnection hurts neither side, and it can be ignored. After the server receives the seed, though, the hand must complete. This type of atomic guarantee can be provided by fair exchange protocols.

Fair exchange protocols have been designed that force transaction atomicity via the use of a trusted third party. They are intended for business transactions where one wants both the transfer of a good and the payment for that good to happen atomically. In these algorithms, the trusted third party interjects itself to ensure success.

An adaption of these algorithms is the class of Optimistic Fair Exchange protocols [2, 12, 3]. Targeted for the same applications as other fair exchanges, the optimistic fair exchange protocols optimize the case where both parties act in good faith. During a typical exchange, the parties transfer the goods and money normally, and the third party is oblivious to the transaction. Either side can request arbitration from the third party. An unsatisfied participant gathers all of its own messages and state, and the messages gathered from the remote participant, and passes them to the third party. The third party uses this data, and encrypted annotations added to the clients communication, to determine the outcome.

In either of the above protocols, the trusted third party is presumed to have some out-of-band executive authority to enforce its decision.

The protocols above attempt to establish an atomic set of actions. In the case of blackjack we wish for a hand, once it has started, to run to completion. A client or server should not benefit by crashing when things are going badly for it.

In a casino game where the casino's actions are non-deterministic for a given hand, integrating a fair exchange protocol may be tricky, as the third party may not have sufficient knowledge to make correct decisions on behalf of the casino. Blackjack's rules define the action of the casino on any hand, so the only participant which must make decisions is the client.

We believe that optimistic fair exchange protocols can be applied to our blackjack system to force operation (hand) atomicity. There are three cases of disconnections (or network partitions):

1. **Server disconnection** The client finishes the game with the third party. Third party signs the sequence of client moves and logs the transaction outcome. The signed sequence is given to the client. When the client reconnects to the server, it presents the sequence as proof that the game was played.

2. **Client disconnection** The server contacts the third party to determine if the client has already finished the hand with the third party. If it has, the server uses whatever outcome was returned by the trusted third party. Otherwise, it finishes the game with the client. Since the client cannot proceed

with more hands until it finishes the current one (either with the trusted third party or with the server), the client gains no advantage for disconnecting. Indeed, if the client must use the server to exchange its virtual casino chips for physical money, then the client will be forced to finish the hand in order to collect the money.

3. **Both disconnection** The client attempts to contact the trusted third party. If it succeeds then the case for server disconnection is followed. Otherwise, the client waits until it can connect to the server, and it finishes its game there.

To apply optimistic fair exchange to blackjack, we must provide the additional evidence needed by the trusted third party to determine the outcome of the game. To recreate the entire game, the third party must have the client and server's random seeds, and the sequence of client operations (Hit, Stay, Split, and Double Down). Define pk as the public key of a trusted third-party, and $\mathcal{E}_{pk}$ as an encryption operation. For the digital signatures below, let $Sign_S$ be a signature with the private key of the server, and let $Sign_C$ be a signature with the private key of the client.

That data is captured by the following additional messages:

1. When the server sends its commitment, it also sends the package $\mathcal{E}_{pk}(seed)$ and $Sign_S(commit)$ to the client.
2. When the client sends its seed, it also sends $Sign_C(seed)$ to the server.
3. The client keeps a counter, starting at one, of the number of actions it has taken. Whenever it sends a "hit", "stay", "split" or "double down" message to the server, it sends $Sign_C(counter||action)$ as well, and it increments the counter.

With this data, either side has enough data to provide the third party so it can know the entire game state.

One may be concerned that accidental disconnections (rather than malicious ones) would be penalized as malicious. Liu *et. al.* give several algorithms for both on-line and off-line fair data exchange which are constructed to prevent loss of fairness, even under benign disconnections [13].

4.2.3 Protecting Against Improper Messages

As this algorithm is applied to other games, it may become necessary for the client and server to encrypt secret data, beyond the seeds, for use by the trusted third party. In the example above, the client receives both the encryption of the seed as well as a signature of the hash of the two. The third party can decrypt the seed and verify them against the commitment and the signature. This prevents the server from giving a bogus seed to the client for dispute resolution.

In general, however, such approaches may not work. During our research on the matter, we discovered some algorithms which may be suited for these problems [16, 1]. In these protocols, a client buys some electronic media from an online reseller. The protocols ensure that the client receives the content they were paying for, and that the transfer of money from the client to the seller and the transfer of the good from the seller to the client are atomic. The algorithms in the above work make use of the commutative property of RSA to allow the client to verify the content of a message without being able to decrypt the message. We believe that such capability would be very useful in ensuring that the messages to the third party are correct.

5 Related Work

When we were concluding our work, we found that `secureplay.com` has advertised a similar means of guaranteeing fairness in online gaming [20]. The high-level description of the protocol seems similar to our bit commitment protocol, but they do not provide theoretical analysis nor discuss issues in disconnections, fair exchange, or application to specific online games. An analysis an comparison of their source code and our might make for interesting future work, however, due to (shipping) fees, we were unable to obtain a copy of the source code.

6 Conclusions

The random number generation algorithm described above can form the basis for a secure online casino by providing uniform random number generators which are trusted and verifiable by both the clients and the servers. Indeed, as monitoring the fairness of a physical casino is sometimes imperfect, this algorithm can provide stronger randomness guarantees than one would find in a real casino.

We have analyzed the protocol and shown reductions to problems which are believed to hard, such as attacking the one-way property of SHA1. To show feasibility, we have also implemented the protocol in code, producing both client and server programs who can connect and play blackjack over the Internet.

Lastly, we have summarized some other attacks which could be raised on an Internet casino, especially the problem of disconnections, and we have speculated on how one would apply algorithms and protocols from electronic commerce and business to ensure fairness under these attacks.

A Appendix

We describe the basic rules of blackjack play. Variations of the game can include multiple players as well as multiple decks. The description below is sufficient to understand the issues in online gaming and the guarantees of our protocol.

A.1 Objective

The goal of the game is to obtain a hand of cards valued as close to 21 without going over 21. In a typical casino setting, there is the house (who deals the cards and is associated with the casino), and at least one player competing only against the house (not each other if there are multiple players). We will describe the case of one player against the house. The case of multiple players competing against the house is very similar.

A.2 Scoring

Each face card is worth 10 points, an ace can be valued at 1 or 11, and this value can change to the advantage of the card-holder as cards are added to the hand. Each numbered card is worth the same value as its face value. The term *blackjack* is synonymous with scoring a 21 with the first two cards in a hand.

A.3 Play

The game begins with the player placing a wager on the table. This occurs before any cards are dealt.

The house starts out by dealing four cards. The first and third cards are dealt face-up to the player, and the second and fourth cards are dealt to the house, with only the second card being face up. If at any time the player has a hand valued at 21, its play is over and the house plays its cards. If at any time the player has a hand that's minimum value (i.e. even if any and all aces are valued at the minimum value of 1), the player loses that particular hand. Otherwise, the client has 4 options: hit, stay, double-down, or split.

A.3.1 Player Moves

A player *hits* when it wants another card from the house. It can continue to hit as long as it would like, as long as the value of the hand stays below 22. A player *stays* when it is satisfied with its hand. The turn then goes to the server to play its cards.

A player *splits* when it has two cards in its hand with the same face value (not numeric value). For example, a jack and a queen have the same numeric value, but different face value, whereas two queens have the same face value. A split gives the player two hands, with each hand being bound to a wager equal to the amount of its original wager. The player then proceeds to play each hand separately, with the house automatically dealing a second card at the beginning of play for each hand.

A player can *double down* at any time. This means that the player requests exactly one more card (i.e. *hits* once) and doubles its wager. This may be advantageous for example, if the player has a 6 and a 5. Assuming only a single player and a full deck, this means the most probably numeric value for the next dealt card is a 10, and therefore it is likely that the player will have a 21.

A.3.2 House Moves

In casinos, the play of the house is deterministic, and known beforehand to the player. The house can only hit or stay. It must continue to hit until it reaches a *hard* or *soft* 17. A hard hand refers to a hand without an ace, or with an ace valued at 1. A soft hand refers to a hand valued at 11. Whether it must hit until a *hard* or *soft* 17 depends on the policies of the particular casino.

A.4 Winner Determination

In the case of a tie, the game is termed a *push*, and no money exchanges hands. However, a *blackjack* beats a hand valued at 21 if the hand has more than two cards (i.e. is not a blackjack).

References

[1] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *ACM Conference on Computer and Communications Security*, pages 7–17, 1997.

[2] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange, 1998.

[3] Feng Bao, R.H. Deng, and Wenbo Mao. Efficient and practical fair exchange with off-line ttp. In *1998 IEEE Symposium on Security and Privacy*, pages 77–81, May 1998.

[4] Peach Casino. Peach casino - fair gaming. `http://www.peachcasino.com/fair_gaming.html`.

[5] Linux Kernel Source Code. Kernel.org: `http://www.kernel.org/`.

[6] National Indian Gaming Commission. National indian gaming commission. `http://www.nigc.gov/nigc/about/mission.jsp`.

[7] Nevada Gaming Commission. Gaming control board home page. `http://gaming.nv.gov/`.

[8] D. Coppersmith. Cheating at mental poker. In *Advances in Cryptology: Proceedings of CRYPTO 85*, pages 104–107, 1987.

[9] fairbet.org. Fairbet::reviews, online gambling and online casino ratings!. `http://www.fairbet.org`.

[10] S. Fortune and M. Merritt. Poker protocols. In *Advances in Cryptology: Proceedings of CRYPTO 84*, pages 454–464, 1985.

[11] GameDay. Gameday security. `http://info.betgameday.com/security.html`.

[12] Sigrid Gurgens, Carsten Rudolph, and Holget Vogt. On the security of fair non-repudiation protocols, October 2003.

[13] Peng Liu, Peng Ning, and Sushil Jajodia. Avoiding loss of fairness owing to process crashes in fair data exchange protocols. pages 31(3):337–350, 2001.

[14] Players Network. Players network - blackjack. `http://www.playersnetwork.com/blackjacksinglevsmultideck.html`.

[15] Ian Portsmouth. The house always wins. `http://www.cryptologic.com/news/profit100winner.pdf`.

[16] Indrajit Ray, Indrakshi Ray, and Natarajan Narasimhamurthi. A fair exchange e-commerce protocol with automated dispute resolution. In *Proceedings of the Fourteenth Annual IFIP WG 11.3 Working Conference on Database Security*, Schoorl, The Netherlands, August 2000.

[17] Steve Schneider. Formal analysis of a non-repudiation protocol. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop*, pages 54–65. IEEE Computer Society Press, 1998.

[18] Bruce Schneier. Applied cryptography. pages 92–95.

[19] Bruce Schneier. Applied cryptography. pages 89–90.

[20] secureplay. secureplay: how it works. `http://www.secureplay.com/products/howitworks.htm`.

[21] startcasino.com. start casino: Internet casino legal issues. `http://www.startcasino.com/legal.php3`.

[22] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 55–61, Oakland, CA, 1996. IEEE Computer Society Press.