Remote Electronic Gambling

Chris Hall Bruce Schneier
Counterpane Systems
101 East Minnehaha Parkway
Minneapolis, MN 55419
{hall,schneier}@counterpane.com

Abstract

We examine the problem of putting a casino on the Internet. We discuss fairly generating random bits and permutations for use in casino games, protecting against player/player and player/dealer collusions, and ensuring a secure audit trail that both the player and dealer can use to ensure payment of debts. We conclude with a series of open problems.

1. Introduction

Over the past several years commercial interest in the Internet has grown. Naturally, casinos are eager to get online in order to try and make money just like everyone else. One problem with electronic casinos — especially those housed in obscure countries with lax gambling regulations — is that they cannot necessarily be trusted: the players have no way of being sure that the games are not rigged, that the dealer is not cheating, or that the casino will pay out as promised. This paper presents some solutions to these very real trust problems, and can potentially be applied to any electronic commerce situtation where trust is required.

Casinos exist to make money. People go to casinos in hopes of making money. Therefore, two of the biggest issues in designing gambling protocols are betting and debt payoff. In section 2 we briefly describe some of the issues involved with betting and paying off debts, and in section 3 we discuss one way that money can be represented in electronic casinos.

Most electronic gambling games boil down to making a prediction, generating random numbers, and calculating a win/loss result. If the numbers are truly random, then electronic games are mathematically indistinguishable from their real-world counterparts. In section 4 we explore protocols which allow two or more people to work together to choose a random number which none can influence directly. A similar problem to choosing random numbers is how two or more players can shuffle a deck of cards for a game; in section 5 we discuss protocols for card games, including how to shuffle and deal cards.

When designing cryptographic protocols for gambling, several issues must be kept in mind. While the ideal protocol will not allow any player to cheat, this ideal can be difficult, if not impossible, to achieve. Therefore when designing our protocols we focus more on making cheating detectible by players and casinos. This means that we work to establish a good audit trail so that someone who suspects that a player is cheating can use the audit trail to see if they are right. However, this is not good enough. Using the audit trail, the casino must be able demonstrate to a judge or another authority figure that the first player cheated (if that is indeed the case). In section 6 we discuss how the various protocols described in the previous sections are tied together with hash chains in order to form an auditable chain to form a body of evidence strong enough to convince a judge.

1.1. Notation

The following notation is used throughout the paper.

PK_A Denotes the public key of user A. This can be a RSA [30] public key for encryption or digital signatures, a DSS [25] public key, or a key for some other public key scheme.

SK_A Denotes the secret key corresponding to the public key PK_A. Again, this key can be used for either encryption or digital signatures.

$E_{PK}(M)$ Denotes the encryption of the plaintext message M with the public encryption key PK.

$D_{SK}(C)$ Denotes the decryption of the ciphertext message C with the secret decryption key SK.

$E_K(M)$ Denotes the encryption of message M with a symmetric encryption algorithm, such as DES [23], IDEA [19], or Blowfish [31], and key K.

$D_K(C)$ Denotes the decryption of the ciphertext message C with a symmetric encryption algorithm and key K.

$S_{SK}(M)$ Denotes the signature of message M with secret signature key SK.

$H(M)$ Denotes the hash of the message M with a cryptographic hash function like SHA-1 [24] or RIPE-MD [11].

A, B Denotes the concatenation of A and B. This is commonly used when describing messages.

Descriptions of these algorithm and methods can be found in [37, 32, 22].

2. Bets and Payoffs

Two of the core issues in gambling are bets and enforcing their payoff. It is important that one party (gambler or casino) cannot make a bet on another person's behalf, and that a party can prove that someone else owes them money as a result of a bet. This means that they must be able to prove not only that they made a bet with the other party, but that the outcome of the bet was in their favor. However, the other party should also have a way of proving that they paid the debt once they do so.

There are several different betting situations that we have considered in designing our protocols. For some games, like roulette, the bets are between a gambler and the casino. When the gambler wins, the casino must pay their debt and vice-versa. For other games, like poker, several players play against each other and the casino merely arbitrates. Typically the casino takes the pot that results from a game, keeps a percentage for itself (a "rake"), and distributes the money among the winner(s). Hence, while the gambling debts are really between players, they can all be transformed into debts between the casino and the players. By doing this we can also enforce player anonymity as the only iteraction two players have occurs while they play the game, and in order to play a game it is not necessary to know the identity of the other person. Once the game is over the players interact with the casino in order to pay/collect their money.

The protocols in the following sections have been designed so that one player can prove that another participated in a game and so that one player can prove that the outcome of the game was in their favor if necessary, i.e. they need to collect money after the protocol has ended. However, the protocols presented do not enforce payoff. We are assuming that a trusted third party exists for various services including settling payoff disputes.

3. Gambling Credit

In order for a player to effectively gamble in a casino they must have money. One simple way to represent money in a casino is in the form of gambling credit. That is, each player has a credit with the casino. When they lose a game the credit decreases appropriately and when they win a game it increases. Once a gambler's credit with a casino reaches zero they cannot gamble anymore unless they purchase more credit. On the other hand, a gambler should have the option of leaving the casino and collecting the appropriate amount of money represented by their credit (provided, of course, that they are not in the middle of a game). The casino should be careful that a player cannot participate in more than one game at a time. Otherwise, the player may place bets on two different games for the full value of their credit. If they lose both games, then they lose more money than they have available to them in credit, and hence they owe the casino money. It may be dificult (or impossible) for the casino to collect that money.

There are many electronic commerce protocols [6, 28, 29, 15, 3, 4, 21, 7, 13, 20, 27, 26, 1, 36, 38, 33], and most can satisfy these requirements. However, as we mentioned in the introduction, the casino cannot be trusted. Hence, player's cannot necessarily trust that the casino will keep track of their gambling credit accurately or pay the appropriate amount of money when they try and collect. We have designed our protocols so that the player and casino work together to create an auditable hash chain. The chain starts when the gambler enters the casino and purchases some credit, and ends when the gambler collects on their credit and leaves the casino. Every game that the gambler participates in becomes part of the hash chain. The chain is only valid once all games used to create it are accounted for, and hence the casino and player can each prove exactly how much money exchanged hands. That way the casino and player can each prove exactly how much credit the player should have when they go to leave the casino. This provides the sort of audit trail that we mentioned in the introduction which either player or casino can use to catch a cheater.

We should point out that while the casino and the player work together to create the hash chain, we have taken care to design the protocols so that neither one can cheat the other. Thus, we have designed the hash chain so that either player or casino can use a partially formed hash chain to prove the other's actions up to the point of cheating. Since the information contained in a valid, partially-formed hash chain binds both the player and the casino, the longest partially-formed hash chain can be used the event of a dispute. The person settling the dispute can then decide whether one of the two parties cheated and how the dispute should be resolved.

4. Fairness of Pure Luck Games

Most casino games are solely games of chance: one or more players make a bet on what the outcome of the game will be and the dealer (or machine) commences with the game. This usually consists of rolling some dice, spinning a wheel, or even spinning the dials of a slot machine.

In order for these kinds of games to be fair, the players should have some sort of assurance that the dealer is not cheating, either by picking "bad" random numbers or lying about the outcome. If the players and dealer agree on how the game should be played, they can work together to generate a random number whose value will be used to determine the outcome of the game.[1] Once the outcome is known, the players who won gain an appropriate credit with the casino, and the players who lost lose an appropriate amount of credit.

Consider a two person protocol for generating a random number. Suppose that Alice is the player and Bob is the dealer. There are several protocols in which two people use a one-way hash function, H, to generate random values. They commit to randomly chosen values and exchange them in order to generate a random number which neither has an influence over. We outline one protocol below among those found in [2, 5, 32].

1. Alice generates a random number R_0 and sends $H(R_0)$ to Bob (this protocol assumes that R_0 is large enough that Bob cannot do a brute force search for R_0 given $H(R_0)$).
2. Bob generates R_1 and sends it to Alice.
3. Alice sends R_0 to Bob and computes
$$R = R_0 \text{ XOR } R_1.$$
4. Bob verifies that the expected value of the hash matches the actual value. If they do, he computes
$$R = R_0 \text{ XOR } R_1.$$

This protocol works, but there is a problem when applying it to electronic gambling. Recall that the random value R will be used to determine the outcome of the game. If Alice determines that the computed value of R will cause her to lose the game, then she can refuse to send R_0 to Bob by cutting the connection. Unfortunately Bob has no way of proving that Alice cut the connection maliciously and he also has no way of knowing whether or not he won the game. He may suspect that he won, but he needs more evidence if he wants to take Alice to court.

Nearly every protocol for generating random numbers in this fashion suffers from the same weakness: Alice learns the outcome before Bob. Because we want to apply this protocol to gambling, one solution to this problem is to make it so that Alice does not owe any money to Bob regardless of the outcome of the protocol. She can do this by paying Bob as if she had lost. If she wins, then Bob not only owes her the money he gambled, but he owes her the money she payed him before carrying out the protocol. If she loses, then Alice cannot change the fact that Bob gets to keep her money so she has no incentive to halt the protocol.

This protocol also applies to gambling games like slot machines. Alice pays for a chance on the slot machine before she plays and tries to collect her money if she wins. This is fairly similar to how real slot machines work as you must put money or tokens into the machine before pulling the handle. The protocols for paying and playing should be intertwined in order to prevent a gambler from paying for one turn and "reusing" that turn with a replay attack until they win.

One more modification needs to be made to this protocol before it can be used. As stated, there is no way for Alice to prove that Bob actually performed the protocol with her. Thus, if Alice wins the game and Bob decides not to pay her, then Alice cannot prove to a trusted third party that Bob owes her money. We can remedy this situation by using digital signatures. The following protocol uses signatures and it has also been generalized for situations in which two or more players wish to participate. Alice is the dealer and Bob, Carol, and Dave are the three players.

1. Alice generates a random number, R_A, and sends everyone:
$$M_0 = H(R_A), S_{SK_A}(H(R_A)).$$
2. Bob generates a random number, R_B, and sends everyone:
$$M_1 = H(R_B), S_{SK_B}(M_0, H(R_B)).$$
3. Carol generates a random number, R_C, and sends everyone:
$$M_2 = H(R_C), S_{SK_C}(M_1, H(R_C)).$$
4. Dave generates a random number, R_D, and sends everyone:
$$M_3 = H(R_D), S_{SK_D}(M_2, H(R_D)).$$
5. Once Alice receives everyone else's hash, she sends everyone:
$$M_4 = R_A, S_{SK_A}(M_3, R_A).$$

[1] Protocols which help the players agree on the rules of the game are beyond the scope of this paper. However, if should be noted that while a physical casino is limited by the knowledge and training of the dealers, an electronic casino can easly play with hundreds of different "house rules" simultaneously.

6. Once Bob receives M_4, he sends everyone:

$$M_5 = R_B, S_{SK_B}(M_4, R_B).$$

7. Once Carol receives M_5, she sends everyone:

$$M_6 = R_C, S_{SK_C}(M_5, R_C).$$

8. Once Dave receives M_6, he sends everyone:

$$M_7 = R_D, S_{SK_D}(M_6, R_D).$$

9. Everyone computes

$$R = R_A \text{ XOR } R_B \text{ XOR } R_C \text{ XOR } R_D.$$

A comment on this protocol first. Each person must send their hashes before sending their actual values. Otherwise the last person who announces their random value could choose a random value which would allow them to win.

There is a difficulty with this protocol. Suppose that Dave represents the casino. Then the casino could try to collude with Carol. Once the protocol reaches step 7, Carol could reveal her value R_C to Dave without telling it to everyone else. Then Dave would know the value of R and thus be able to predict the outcome of the protocol. Dave can request that Carol abort the protocol if necessary. Since no one else knows that Dave and Carol are colluding they cannot blame the aborted protocol on the casino. Similarly, Dave could reveal R_D to Carol and Carol could tell Dave to abort the protocol. Hence, if we had the reverse situation in which Carol represents the casino and Dave colludes with Carol, then Dave could abort the protocol and shift the blame from the casino.

In general, we do not believe that it is possible to design a protocol for more than two people which is safe from collusion. The problem we were originally trying to solve was to decide what would happen if one player aborts the protocol before completion because they know the outcome. If two players are working with the casino to generate a random number, then any way of deciding what to do in the event that one player aborts the protocol is unfair to someone. If the rule is that the game is considered void for the player, then two players could collude to abort the protocol whenever one of them stands to lose a lot of money (Alice and Dave could be the two players in the above protocol). The casino and a player could also collude and abort the protocol whenever the other player stands to win big. It may not be possible to rule that the game is still valid for the other players because they may not know the outcome of the protocol and hence won't be able to prove to a trusted third party that they actually won.

5. Fairness of Shuffling and Dealing

Some casino games are based on one or more decks of cards. Part of a card game is shuffling and dealing the cards in a fair manner. As with the pure chance games we described above, the player wants some way of being sure that the dealer hasn't stacked the deck. There have been several protocols described in the literature for playing poker [35, 10, 14, 39]. These use public-key cryptography and require that players generate new key pairs for each game they play, and this may be too computation-intensive for networked play. Additionally, most of these protocols leak partial information about the cards themselves. Still other protocols [16, 8, 9] have no information leakage, but they are not at all practical. They use zero-knowledge proofs; one implementation on three Sparc workstations took eight hours to shuffle a deck [12].

The following protocol works if the dealer is trusted.[2] The dealer does not know how the cards are shuffled before they are dealt; however, the dealer does know what cards each player gets. Let Trent be the trusted dealer and Alice and Bob be the two players. The first part of the protocol effectively "shuffles" the cards:

1. Trent generates a permutation P_T of the 52 cards and sends to Alice and Bob:

   $$H(P_T), S_{SK_T}(H(P_T)).$$

   Note, Trent (as well as Alice and Bob) should append a random salt to the permutation before hashing it. He should not reveal the salt until the end of the protocol. This will keep Alice or Bob from guessing Trent's full permutation when only a few entries remain unknown (e.g. when almost all the cards have been dealt).

2. Alice generates a permutation P_A of the 52 cards and sends to Trent and Bob:

   $$H(P_A), S_{SK_A}(H(P_T), H(P_A)).$$

3. Bob generates a permutation P_B of the 52 cards and sends to Trent and Alice:

   $$H(P_B), S_{SK_B}(H(P_T), H(P_B)).$$

4. Everyone verifies that the hashes they received are different than the one they generated, i.e. that they all generated different permutations. If they did not, then they redo the protocol.

[2] Recall that in some games the dealer wins a rake regardless of who wins the card game. In these games the dealer can be trusted.

5. Everyone agrees on the order to compose the permutations without Trent's permutation (Alice before Bob or Bob before Alice). Assume for discussion purposes that Alice's permutation will be applied before Bob's.

Although nobody knows how the cards are shuffled yet, the information exists to determine the full arrangment of the cards. Whenever a player needs a card the following protocol can be carried out. Assume that $n-1$ cards have been dealt already and Alice needs a card:

1. Alice determines $P_A[n]$, the nth entry of her permutation, and sends to everyone:

$$M_0 = P_A[n], S_{SK_A}(H(P_A), H(P_B), H(P_T), n, P_A[n]).$$

   Note, including the hash of her permutation as part of the signature prevents someone else from replaying Alice's signature from another iteration of the protocol.

2. Bob determines $P_B[P_A[n]]$, the nth entry of the permutation formed by composing Alice's and his permutations, chooses a random salt R_1, and sends to everyone:

$$M_1 = R_1, P_B[P_A[n]], S_{SK_B}(M_0, R_1, P_B[P_A[n]]).$$

3. Trent verifies Bob's signature and determines $C = P_T[P_B[P_A[n]]]$. He then chooses a random salt R_2 and sends to Alice:

$$E_{PK_A}(R_2, C, S_{SK_T}(M_1, R_2, C)).$$

4. Alice decrypts the message, verifies the signature, and adds the card to her hand.

After the game is over everyone reveals their permutations in order to make sure nobody was cheating. This brings up a problem with this protocol (as well as the next): everyone knows everyone else's hand when the game is over. This is bad in games like poker when someone may not want to reveal if they were bluffing.

The following protocol also works if the dealer is trusted. Let Trent be the dealer and Alice and Bob be the two players. The first part of the protocol effectively "shuffles" the cards:

1. Trent generates a permutation P_T of the 52 cards and sends to Alice and Bob:

$$H(P_T), S_{SK_T}(H(P_T)).$$

2. Alice generates a permutation P_A of the 52 cards and sends to Trent:

$$E_{PK_T}(P_A, S_{SK_A}(H(P_T), P_A)).$$

3. Bob generates a permutation P_B of the 52 cards and sends to Trent:

$$E_{PK_T}(P_B, S_{SK_B}(H(P_T), P_B)).$$

4. Trent composes the three permutations to shuffle the cards. He should compose them in an agreed upon order.

Now whenever someone needs to be dealt a card they ask Trent for a card. He pulls the next card off of the top of the deck, encrypts it, and sends it to the player. He should really include a random salt R so that any of the other players cannot encrypt all 52 possible values for the card to see which one the first player received. To summarize, suppose Alice needs a card and n cards have been dealt already. Then she carries out the following protocol with Trent:

1. Alice picks a random number R_0, generates a request M for a card, and sends to Trent:

$$M_0 = R_0, M, S_{SK_A}(R_0, M).$$

   Note, M should contain an indentifier unique to the game and an indication that this is the nth card to be dealt. That way replay attacks are prevented.

2. Trent verifies the signature, picks the nth card out, generates a random salt R_1, and sends to Alice:

$$M_1 = E_{PK_A}(R_1, M, C_n, S_{SK_T}(M_0, R_1, M, C_n)),$$

   where C_n denotes the nth card.

3. Alice decrypts the message, verifies the signature, and adds the card to her hand.

6. Chaining it all Together

Many gambling games require actions like placing bets and asking for more cards. By using hash chains [17, 18] each event can linked into the game. In order for the hash chain to verify correctly, every event must be accounted for. Hence it is impossible for one player to prove that the game was in their favor unless they show that everything which happened in the game lead to the supposed outcome.

If a dispute arises in which the player claims that the casino did not give them all of their money, then the player must be able to prove this claim to a third party. The player will have to prove that the amount of credit that they purchased along with their net winnings is unequal to what the casino paid them. This means that they will have to prove the outcome of every game, including the ones they lost. On the other hand, the casino should be able to prove the amount of money the player initially had, the net winnings of the casino, and how much money they finally payed the player.

As we said, we have designed our protocols to help the winner of a game prove that they won. We also stated that protocols for helping one player to prove that they paid their debt to another are beyond the scope of this paper. The last supporting element we need is proof that the player actually participated in all of the games that they claim and that they didn't participate in any other games (otherwise a player could try and claim that the casino owes them more money by conveniently "forgetting" a game in which they lost money). One way to implement this protocol is to ensure that the token purchasing protocol, every protocol used in the games, and the final cash-in protocol are linked in an unbreakable manner. This can be done using hash chains.

In [34], Schneier and Kelsey describe a solution to the hash chaining problem. Each signature includes a hash of the previous message in addition to a hash of the message being signed. Hence each signature in the chain forms a link with the previous signature which cannot be changed, even if the signing key is later broken. This means that all actions in the casino are tied together by the signatures generated by the gamblers and the casinos. Therefore, all important events in the games (which are really all events) require a signature from the player stating them.

7. Conclusion and Open Problems

We have explored several issues that are important in online gambling. In addition we have drawn out several protocols which are useful in implementing some of the classic casino games. These include generating random numbers and shuffling a deck of cards. We discussed some of the issues involved in betting and payoffs; however, a full-blown dicussion is beyond the scope of this paper. Finally, we considered how to make the games non-refutable and briefly considered anonymity.

The protocols that we presented are not perfect. There are some open problems which must be solved before we can improve on the protocols. These are the problems as we see them:

1. Is it possible to design a general N-person ($N > 2$), asynchronous coin flipping protocol in which each person contributes to the outcome of the protocol and is immune to player collusion? We briefly addressed this issue when discussing N-person protocols for generating random numbers. However, our protocols are not immune to the effects of player collusion (as we pointed out). We believe that we have an informal proof that such a protocol does not exist, but to the best of our knowledge no formal proof or counterexample exists.

2. Can N people work together to shuffle a deck of cards so that no person knows the arrangment of the deck and that each player does not have to generate a new public-key key pair for the protocol? As we stated previously, the simplest protocols for shuffling a deck of cards require each player to create a new key pair for each iteration of the protocol. For online gambling systems this is impractical.

3. Is there a card shuffling protocol that works even if the dealer is not trusted? There are theoretical solutions—secure multi-party protocols—but are there any practical ones?

8. Acknowledgments

The authors would like to thank John Kelsey, James Jorasch, and Jay Walker for their helpful comments.

References

[1] M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "iKP - A Family of Secure Electronic Payment Protocols", *The First USENIX Workshop on Electronic Commerce,* USENIX Association, 1995, pp. 89–106.

[2] M. Blum, "Coin Flipping by Telephone: A Protocol for Solving Impossible Problems," *Proceedings of the 24th IEEE Computer Conference (CompCon)*, 1982, pp. 133–137.

[3] S. Brands, "Untraceable Off-line Cash in Wallets with Observers," *Advances in Cryptology—CRYPTO '93 Proceedings,* Springer-Verlag, 1994, pp. 302–318.

[4] S. Brands, "An Efficient Off-line Electronic Cash Systems Based on the Representation Problem," C.W.I. Technical Report CS-R9323, 1993.

[5] G. Brassard, *Modern Cryptology: A Tutorial*, Lecture Notes in Computer Science 325, Springer-Verlag, 1988.

[6] D. Chaum, A. Fiat and M. Naor, "Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 319–327.

[7] CitiBank and S. S. Rosen, "Electronic-Monetary System," International Publication Number WO 93/10503; May 27 1993.

[8] C. Crepeau, "A Secure Poker Protocol that Minimizes the Effect of Player Coalitions," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 73–86.

[9] C. Crepeau, "A Zero-Knowledge Poker Protocol that Achieves Confidentiality of the Players' Strategy, or How to Achieve an Electronic Poker Face," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 239–247.

[10] R. DeMillo and M. Merritt, "Protocols for Data Security," *Computer*, v. 16, n. 2, Feb. 1983, pp. 39–50.

[11] H. Dobbertin, A. Booselaers, and B. Preneel, "RIPEMD-160: A Strengthened Version of RIEPMD," *Fast Sofrware Encryption, Third International Workshop,* Springer-Verlag, 1996, pp. 71–82.

[12] J. Edwards, "Implementing Electronic Poker: A Practical Exercise in Zero-Knowledge Interactive Proofs," Master's thesis, Department of Computer Science, University of Kentucky, 1994.

[13] N. Ferguson, "Extensions of Single-Term Coins," *Advances in Cryptology – Proceedings on CRYPTO '93,* Springer-Verlag, 1994, pp. 292–301.

[14] S. Fortune and M. Merritt, "Poker Protocols," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 454–464.

[15] M. Franklin and M. Yung, "Towards Provably Secure Efficient Electronic Cash," Columbia Univ. Dept of C.S. TR CUCS-018-92, April 24, 1992.

[16] S. Goldwasser and S. Micali, "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information," *Proceedings of the 14th ACM Symposium on the Theory of Computing*, 1982, pp. 270–299.

[17] S. Haber, W.S. Stornetta, "How to Time Stamp a Digital Document," in *Advances in Cryptology–Crypto '90*, Springer-Verlag, 1991.

[18] J. Kelsey and B. Schneier, "Cryptographic Support for Secure Logs on Untrusted Machines," in preparation.

[19] X. Lai, J. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology—CRYPTO '91*, Springer-Verlag, 1991, pp. 17–38.

[20] S. H. Low, N. F. Maxemchuk, and S. Paul, "Anonymous Credit Cards," *The Second ACM Conference on Computer and Communications Security,* ACM Press, 1994, pp. 108–117.

[21] G. Medvinsky and B. C. Neuman, "Netcash: A Design for Practical Electronic Currency on the Internet," *The First ACM Conference on Computer and Communications Security,* ACM Press, 1993, pp. 102–106.

[22] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[23] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[24] National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, May 1993.

[25] National Institute of Standards and Technologies, NIST FIPS PUB 186, "Digital Signature Standard," U.S. Department of Commerce, May 1994.

[26] B. C. Neuman and G. Medvinsky, "Requirements for Network Payment: The NetCheque^{TM} Perspective," Compcon '95, pp. 32-36

[27] T Okamoto, "An Efficient Divisible Electronic Cash Scheme," *Advances in Cryptology—Crypto '95 Proceedings*, Springer-Verlag, 1995, pp. 438–451.

[28] T. Okamoto and K. Ohta, "Disposable Zero-Knowledge Authentication and Their Applications to Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 481–496

[29] T. Okamoto and K. Ohta, "Universal Electronic Cash," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 324–337.

[30] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120–126.

[31] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191-204.

[32] B. Schneier, *Applied Cryptography, Second Edition,* John Wiley & Sons, 1996.

[33] B. Schneier and J. Kelsey, "A Peer-to-Peer Metering System," *The Second USENIX Workshop on Electronic Commerce Proceedings,* USENIX Association, 1996, pp. 279–286.

[34] B. Schneier and J. Kelsey, "Automatic Event-Stream Notarization Using Digital Signatures," *Security Protocols: International Workshop, Cambridge, United Kingdom, April 1996 Proceedings*, Springer-Verlag, 1997, pp. 155–170.

[35] A. Shamir, R. Rivest, and L. Adleman, "Mental Poker," in *Mathematical Gardner*, D.E. Klarner, ed., Wadsworth International, 1981, pp. 37–43.

[36] M. Sirbu and J. D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," Compcon '95, pp. 20–25.

[37] D.R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 1995.

[38] J. M. Tenenbaum, C. Medich, A. M. Schiffman, and W. T. Wong, "CommerceNet: Spontaneous Electronic Commerce on the Internet," Compcon '95, pp. 38–43.

[39] M. Yung, "Cryptoprotocols: Subscriptions to a Public Key, the Secret Blocking, and the Multi-Player Mental Poker Game," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 439–453.