Using smartcards to secure a personalized gambling device

William A. Aiello Aviel D. Rubin Martin J. Strauss

AT&T Labs – Research, Florham Park, NJ, USA
{aiello,rubin,mstrauss}@research.att.com

Abstract

We introduce a technique for using an untrusted device, such as a hand-held personal digital assistant or a laptop to perform real financial transactions without a network. We utilize the tamper-resistant nature of smartcards to store value on them and perform probabilistic computations based on user input. We discuss an application of this to gambling. The technique has the properties that the user is guaranteed to make money when he wins and the house is guaranteed to make money when the user loses.

1 Introduction

We are currently experiencing a proliferation of lightweight handheld devices, such as the 3Com palm pilot, windows CE devices, and even laptops that weigh under three pounds. Many of these devices are so portable that people have them in their pockets and use them at all times. Our goal is to enable people to utilize these devices during otherwise dead time perhaps at the doctor's office, or in line at the supermarket. We are interested in utilizing the time when the user does not have access to online (e.g. on the Internet) connectivity. When users have network access, then applications such as banking, shopping, and gambling can be implemented securely because the resources that need protection, namely money, can be secured by cryptographic means. There are many protocols for secure commerce on the Internet (e.g. SSL [10]), and the protocols are already implemented on many mobile devices. Hall *et. al.* present protocols for remote electronic gambling [9] for online users. However, there are many times when a user is in possession of a small computing device, but not on the network. In such scenarios, it is difficult to allow the user to perform financial transactions because interaction with servers is not possible. We assume that the user has full access to the device, so secrets (such as cryptographic keys) cannot be safely stored on it. In our design, the portable computing device is augmented with a smartcard reader. The user obtains a smartcard and connects it to the device. We show how this design can be used for an untrusted user to perform transactions such as placing bets on the outcome of a probabilistic computation. This has direct applications to gambling. In addition, we introduce protocols for adding (purchasing) or removing (selling) value on a smartcard without requiring a network connection. We present protocols whereby neither the user nor the house can benefit from cheating. For the remainder of the paper, we focus on the gambling example, although the architecture is applicable to other scenarios as well.

2 Architecture

There are several principals who participate in the protocols. (The term "principal" is used for a participant in a protocol.) We define them as follows.

The user The user is the entity that gambles against the house. The user is in possession of a portable computing device and purchases smartcards with value on them in order to play.

The house The house is the entity that runs the gambling operation. It issues smartcards with value on them in exchange for money. The house is responsible for redemption of money on smartcards. It is assumed that there is, associated with the house, a public/private key pair whose public component is well-known.

The device We assume that there is a relatively small, portable device that is capable of computation. This could be a special-purpose device built for gambling applications or a standard device such as a palm pilot or a laptop computer. The only physical requirement is that there needs to be a way to read a smartcard. There must also be a source of random numbers. The device is used to interface with the user, so it should have a nice graphical display. We also assume that the public key of the house is available on the device. We assume that the user trusts the device to behave properly. In the extreme, he can build his own device to ensure that. In other words, the device is the agent of the user. The functionality of the device can be fully specified so that different manufacturers can produce devices that interoperate. There are no security requirements associated with the design of the device.

The smartcard The system utilizes tamper-resistant smartcards. The cards contain a processor and some memory. We assume that values can be stored on the card such that reading or modifying them is more costly to an attacker than the benefit that could be derived from such an attack. As long as the total amount that can be stored on the card is kept relatively small (several thousand dollars should do fine), there are many cards available today that meet this requirement. Next, we assume that the smartcard has a serial number, imprinted visibly on the outside, and that information depending on both the card and the house can be placed in the smartcards' secure memory at its manufacture time. (For example, we assume the card's secure memory can be given a signature from the house on the card's serial number. We'll discuss this again later when present our protocols.) Finally, the smartcard must be able to generate random numbers. This can be achieved either in hardware (e.g. using a noisy diode or a low-accuracy clock) or in software (e.g. using a built-in seed and a cryptographic pseudorandom number generator [11]). The latter requires some nonvolatile memory, where some state can be maintained over time. Ideally some combination of these techniques is used.

The arbiter The arbiter is a party that is used to resolve disputes. In practice this can be a court of law or an entity mutually agreed upon by the users and the house.

3 Security requirements

In this section, we describe the security requirements of the system as they apply to the gambling application. The requirements are all met by the protocols described later.

1. Only the house should be able to add or subtract money from the smartcard without participating in a game.
2. Once a user places a bet and plays a game, he cannot prevent the loss of that amount if he loses the game.
3. A user can detect the situation where he wins a game, but is not credited for his bet. He can also prove this to the arbiter.
4. The house must publicize the algorithms and probabilities that are used on the smartcard. That is, the house must announce the rules for each game. It must be impossible for the smartcard to weigh the probability in favor of the house beyond those given by the rules.
5. The house can set limits, per game or over all games, on the smartcard, and games with bets beyond the limits do not count.
6. The user cannot risk more than the amount on the card.
7. The house must refund the amount on the smartcard whenever the user wishes. This requirement must be satisfied outside of the system. One possibility is for the house to keep a certain amount of money in escrow, under the control of the arbiter. When the money is refunded, the amount is deducted from the smartcard.

These requirements all hold in our system. Thus, the user can play games with assurance that if he wins, he will actually win the money in the bet, and the house knows that users will not be able to cheat. In addition, the user is guaranteed that the odds of winning published by the smartcard are accurate.

4 The protocols

In this section, we describe the protocols between the various parties for the gambling system.

4.1 Preliminaries

Here, we describe some notation and structure of our protocols. This section describes protocol aspects that are common throughout the paper.

4.1.1 Messages

We use the notation

$A \rightarrow B$: Message

to represent that A is sending *Message* to B. Unfortunately, this notation does not explicitly state any computation or verification that is performed by either A or B, so we include those descriptions in text that annotates the messages.

4.1.2 Digital signatures and encryption

We use the notation $[text]_{SC}$ to indicate that *text* is signed by the entity SC. Throughout the paper we use SC to represent the smartcard and D to represent the device. We also assume that the message containing $[text]_{SC}$ includes the public key certificate for SC signed by the house. So, in effect, when the message $[text]_{SC}$ appears for the first time in a protocol, it should be read as $[text]_{SC}, [SC, \text{public-key(SC)}]_H$, which represents a signature by the house. Thus, anyone in possession of the public key of the house can verify the certificate and then the signature by SC. We do not include the certificate in the messages below as to not burden the reader, but we assume they are implicit in the messages. We also assume that signature verification is part of the protocol and do not ever mention that explicitly.

In our protocols, the smartcard signs every message before it is sent. For simplicity, we do not explicitly show that the messages are signed. So, whenever

smartcard $\rightarrow$ device : msg

appears, it should be interpreted as

smartcard $\rightarrow$ device : $[\text{msg}]_{SC}$

4.1.3 Chaining messages together

Our system design is assymetric in the sense that the smartcard's protection against the user is tamper-resistant hardware, whereas the user's protection against the smartcard is the ability to take a transcript of the communication to the arbiter for dispute. Thus, the smartcard signs messages, but the device does not.

Hash chaining is a method for linking messages to each other within a communication session [8, 14, 9, 15]. Throughout the paper, when discussing hash functions, we refer to cryptographic hash functions such as MD5 [13] and SHA1 [2]. Our goal is for the device to store an undeniable transcript of all communication with the smartcard. That is, the house should not be able to repudiate that the messages in the transcript were sent.

In addition to the properties discussed above, we would like to make it impossible for the smartcard to misbehave by including a bogus message as the previous message received from the device. To achieve this, the device generates a random key, K_D upon startup, and uses this key to produce a MAC of messages that are sent to the smartcard. Subsequent MAC computations include all previous MACs, and we call this a running MAC. When the smartcard includes the previous message in its signed message, the running MAC is included as well. Since every message from the device contains a running MAC, it is impossible for the smartcard to produce a valid message that contains a forged message from the device.

We can also achieve nonrepudation of messages sent by the device if we can assume that there is no way for the device to generate two messages that map to the same MAC output with different keys. While this is not a proven property of MAC functions such as HMAC, it is widely believed to hold (The smartcard can improve things by including a new random value in every message.) The running MAC is included in every message sent from the device to the smartcard. It is computed over the previous message received from the smartcard, the current message being sent, and the running MAC from the previous message sent. The running MAC is defined as follows:

- $RMAC_1 = MAC_{K_D}(msg_1)$
- $RMAC_n = MAC_{K_D}(msg_{n-1}, msg_n, RMAC_{n-2})$

where ',' denotes concatenation.

To illustrate, the communication between the device and the smartcard is as follows:

device $\rightarrow$ smartcard : $x_1 = msg1, MAC_{K_D}(msg_1)$

smartcart $\rightarrow$ device : $x_2 = [msg_2, x_1]_{SC}$

device $\rightarrow$ smartcard : $x_3 = msg_3, MAC_{K_D}(msg_2, msg_3, MAC_{K_D}(msg_1))$

smartcart $\rightarrow$ device : $x_4 = [msg_4, x_3]_{SC}$

The third message could be written simply as

smartcart $\rightarrow$ device : $x_3 = msg_3, RMAC_3$

The device should verify the signature and the MAC values in each message before accepting it. The purpose of the signed MAC chain is non-repudiation. Armed with a history of the messages, the device can prove that it sent and received the messages in the order that they occurred. In the remainder of the paper, we assume that these hash values are included in each message from the smartcard, and we do not explicitly show them. So, we write the above protocol as:

device $\rightarrow$ smartcard : msg_1

smartcart $\rightarrow$ device : msg_2

device $\rightarrow$ smartcard : msg_3

smartcart $\rightarrow$ device : msg_4

and all the MACing and signing is implicit.

4.2 Setup

To set up the gambling application, the user must purchase a smartcard with value from the house. This happens outside of our system. The user must have some confidence that the smartcard really has the amount of money that the user has paid. Therefore, we define the following query which the smartcard and the device must support:

Device $\rightarrow$ Smartcard : *money-query*

Smartcard $\rightarrow$ Device : *value*

The device prompts the smartcard for the amount of money. The smartcard returns the current balance. Recall that the message from the smartcard is actually signed and contains several hash values, but we omit these details in our protocols description as described in Sections 4.1.2 and 4.1.3.

4.3 Dealing a random card

In this section, we describe a protocol for the smartcard to deal a *random* face-up card to the device. That is, we give a protocol whereby a card is chosen at random from the deck such that each card is equally likely to be chosen, and there is no way for the device nor the smartcard to bias the selection. In the end, the card is known to both parties. The basic idea is based on previous work on coin flipping by telephone [5]. In the remainder of the paper, when we say that the smartcard deals a random card to the device, we are referring to this protocol.

We focus our protocol description on a standard pocker deck of 52 cards, although the protocol can easily be generalized for other games. We map the cards in the deck to integers as follows so that the problem of dealing the first card reduces to picking a number from 1 to 52.

1 Ace of Spades

2 Two of Spades

...

13 King of Spades

14 Ace of Hearts

15 Two of Hearts

...

26 King of Hearts

...

52 King of Clubs

Thus, we say that the smartcard deals the Ace of Hearts to the device if the the number 14 is chosen. In order for a card to be *dealt*, the device and the smartcard run a protocol whereby they *agree* on a number from 1 to 52. To accomplish this, each side provides a random piece, and they are combined in such a way that a random choice results.

We first describe how the first card is chosen, and we then show how subsequent cards can be dealt from the deck. The smartcard chooses a random number $salt_{\mathrm{SC}}$ from 1 to 2^{160}, and sends it to the device, and the device chooses a random number $salt_{\mathrm{D}}$ from 1 to 2^{160}. Next, the device picks a random number from 1 to 52; we call this $value_{\mathrm{D}}$. The device then concatenates the three numbers and computes a one-way transformation, $half_{\mathrm{D}} = H(salt_{\mathrm{SC}}, salt_{\mathrm{D}}, value_{\mathrm{D}})$. Here we assume that for a randomly choosen $salt_D$ unknown to the smart card, $half_D$ appears pseudo random (and thus reveals only neglible information about $value_D$) to the computationally bounded smart card. In practice, we can use a cryptographic hash functions such as SHA1. The smartcard then picks a random number, $value_{\mathrm{SC}}$, from 1 to 52.

The device sends $half_{\mathrm{D}}$ to the smartcard. The smartcard replies with $value_{\mathrm{SC}}$ (recall from sections 4.1.2 and 4.1.3 that all messages from the smartcard are signed and contain a running MAC). At this point, both sides have committed to their values, but the smartcard does not know $value_{\mathrm{D}}$. So, the device sends $value_{\mathrm{D}}$, $salt_{\mathrm{D}}$ to the smartcard. Now, the smartcard can verify that the value and salt in this message, together with $salt_{\mathrm{SC}}$ hash to the half sent earlier. Both sides next compute $((value_{\mathrm{D}} + value_{\mathrm{SC}}) \bmod 52) + 1$. The result is a random number from 1 to 52. The purpose of $salt_{\mathrm{D}}$ is to prevent the smartcard from computing the value chosen by the device by exhaustively searching for the preimage of H. For example, if the device simply sent $H(value_{\mathrm{D}})$ or $H(value_{\mathrm{D}}, salt_{\mathrm{SC}})$, the smartcard could compute H for each number from 1 to 52 and see which one matched. It could then force any card it wanted to as the choice by picking its value appropriately. The purpose of introducing $salt_{\mathrm{SC}}$ is to prevent a nonuniform device from opening the commitment $H(\cdot)$ in two ways. For example, if the protocol requires the device simply to send $H(salt_{\mathrm{D}}, value_{\mathrm{D}})$, then the device could compute, offline, values $salt_{\mathrm{D}}$, $value_{\mathrm{D}}$, $salt_{\mathrm{D}}'$, $value_{\mathrm{D}}'$ with $H(salt_{\mathrm{D}}, value_{\mathrm{D}}) = H(salt_{\mathrm{D}}', value_{\mathrm{D}}')$ but $value_{\mathrm{D}} \neq value_{\mathrm{D}}' mod 52$. This would allow the device to affect the outcome of $((value_{\mathrm{D}} + value_{\mathrm{SC}}) \bmod 52) + 1$ by choosing $value_{\mathrm{D}}$ or $value_{\mathrm{D}}'$ after learning $value_{\mathrm{SC}}$. It follows that either the device or the smartcard can insure that the resulting value is random, unbiased by the other party.[1]

The device and smartcard are running a commitment protocol. For general background on bit commitment, see [6] and [12]. Note that we are using a hash function to implement commitment, but there are implementations of commitment that use only pseudorandom number generators [12]. Also, note that the commitment is over a secure channel between two parties (the smartcard and the device), neither of which performs simultaneous transactions with other parties. Thus we don't need the commitment protocol to be non-malleable [7], even if several cards are dealt in parallel. This is good, because non-malleable commitment is inefficient.

Until now, we showed how to pick one card out of a deck of 52 cards. However, most interesting games require that more than one card be dealt. The device and the smartcard keep track of which cards have already been dealt using a 52-bit vector. A bit in the vector is set if the corresponding card is still in the deck, and it is a zero otherwise. The number of 1s in the vector represent the number of cards remaining in the deck.

We modify the protocol above as follows. The first message from the device to the smartcard is $(half_{\mathrm{D}}, vector)$, where *vector* is the 52-bit vector. The smartcard replies with $value_{\mathrm{SC}}$, *vector*. In this way, the two sides agree about which cards are in the deck. The two values, $value_{\mathrm{D}}$ and $value_{\mathrm{SC}}$ are chosen from 1 to n, where n is the number of 1s in the vector. Then, once all of the messages have been exchanged, the two sides compute $k = ((value_{\mathrm{D}} + value_{\mathrm{SC}}) \bmod n) + 1$. The result, k, is between 1 and n, inclusive. The card chosen corresponds to the position of the k^{th} 1 in the vector.

To illustrate, take the following example. The vector is

10110101111011111111010111
10111011011110111101100110

There are 38 1s in the vector. Say that $value_{\mathrm{D}} = 27$ and $value_{\mathrm{SC}} = 18$. Then, $k = ((27 + 18) \bmod 38) + 1$, which is 8. The 8^{th} 1 is in position 11 in the vector, so the card dealt is Jack of Spades.

4.4 The game

We assume that a smartcard can support many different types of games with their own bet limits and odds. Thus, we define the following messages where the smartcard informs the device of the games that are available.

Device $\rightarrow$ Smartcard : *game-query*

Smartcard $\rightarrow$ Device : *game-list*

Here, game-list is a list of games. It is assumed that the rules along with their implied probabilities are either known or included in the list. The list of games could simply be a list of numbers that index into a booklet where games are described in detail. The booklet could be available on the device, so that the user could browse the rules.

4.4.1 An example: high card

For the purpose of simplicity, we describe a game where there is no interaction with the user once the cards are dealt. We also assume cards are only dealt face-up. The example is a game of high-card where the user and house are each dealt a card from the same deck and whoever has the highest card wins. The house wins in the case of a tie and Aces are always high. It is assumed that each game is identified by some ID. An example is:

[1] The hash function we use here, as well as the hash functions used in Section 6, must have a number of scrambling properties of the sort commonly assumed in the literature and commonly attributed to SHA1. In particular, the hash functions need to interact securely with other operations such as signatures and concatenation, as well as the particular rules of the card game implemented. The precise requirements of the hash function are straightforward, though tedious, to enumerate precisely.

Game:	high-card
Odds:	27:24
Limit:	$100 per game

The details of how games and odds are represented are not important for this paper, as long as we assume that they are somehow represented.

Before the user begins a game, the device does a money-query to link the balance in the card before the game to the transcript for the game. The user picks a game and specifies the bet. Of course, the messages from the smartcard are signed and contain hashes of previous messages as described earlier. A detailed example of the high-card protocol is given in Figure 1.

We refer to Section 4.3 where the random deal is described in detail. In message 5, the device specifies that the user bets $15 on a game of high card. In messages 6 through 9, the device and smartcard perform the protocol for dealing a single card (the device's card), and in messages 10 through 13 the device and smartcard deal the smartcard's card. The final messages have the effect of entering the outcome of the bet and the new balance into the hash-chained transcript maintained by the device.

It is important that the commitment made in message 7 is verified. So, in our example, after the smartcard receives message 9, it must check that the value submitted is the same one that was committed to earlier. To do this, the smartcard recomputes the hash of two salts and the value and compares it to the value submitted in message 7. It also verifies that the card chosen is the correct one, the 8 of Hearts in our example.

In practice, one would exchange the roles of the smartcard and device for the card dealing protocol in messages 10 through 15 to increase the number of messages sent in the same direction as previous messages. Successive messages in the same direction may be collapsed to shorten the protocol. Another simple optimization is to deal all of the cards at once. This could easily be accomplished by combining messages 6 and 10, 7 and 11, etc. To deal n cards, the size of each message increases by a factor of n, but the number of messages remains constant at four.

5 Other games

The same techniques described above can be used to play other games. We briefly describe how the techniques in this paper could be used to play the following:

Blackjack The random deal could be used to play blackjack against the house. First, the house "deals" two face up cards to the user, using the techniques of Section 4.3. Then, the house deals itself a card. The device can display a face down card to the user, but the card has actually not been dealt yet. The user then decides how to play his hand, and cards are dealt as requested. Finally, when the user decides to hold his hand, the dealer's second card is dealt. In the device, the down card appears to flip over. Finally, any additional cards needed by the house are dealt. The signed transcript is used to settle any disputes.

Slots The techniques used to deal cards could be used to pick random numbers of any size. A slot machine is easy to implement with such a tool. The pictures on each wheel of the slot machine are numbered, and the spinning of each wheel corresponds to the house "dealing" a random number in the proper range. If the slot machine displays five images, then five random numbers are agreed upon by the device and the smartcard, and the graphical user interface is used to display five pictures corresponding to the numbers chosen.

Craps Rolling the dice to play Craps corresponds to picking two random numbers between one and six. It is straightforward to apply the techniques from Section 4.3 to do this.

Poker A typical poker machine works as follows. The house deals five cards to the user. The user discards up to four of them (four is only allowed if the fifth one is an ace). The house then deals cards to replace the discarded ones. If the quality of the hand is above a certain threshold, the user wins. This can all be accomplished using techniques from Section 4.3.

6 Adding or removing value

In this section we explore protocols for increasing or decreasing the value on the card. To increase the value on the card the user pays money to the house, and the house somehow increases the balance on the card. To remove value, or "cash out" the balance on the card is reduced and the house pays the money to the user. These protocols are applicable beyond the domain of gambling.

Both of these protocols are easy if we assume a high bandwidth channel between the card and the house.

1. Device $\rightarrow$ Smartcard : *game-query*
2. Smartcard $\rightarrow$ Device : *game-list*
3. Device $\rightarrow$ Smartcard : *money-query*
4. Smartcard $\rightarrow$ Device : \$1545
5. Device $\rightarrow$ Smartcard : \$15, high-card
6. Smartcard $\rightarrow$ Device : $salt_{\mathrm{SC}}$
7. Device $\rightarrow$ Smartcard : $half_{\mathrm{D}}$, $vector$
8. Smartcard $\rightarrow$ Device : $value_{\mathrm{SC}}$, $vector$
9. Device $\rightarrow$ Smartcard : $value_{\mathrm{D}}$, $salt_{\mathrm{D}}$, I get 8 of Hearts
10. Smartcard $\rightarrow$ Device : $salt_{\mathrm{SC}}'$
11. Device $\rightarrow$ Smartcard : $half_{\mathrm{D}}'$, $vector'$
12. Smartcard $\rightarrow$ Device : $value_{\mathrm{SC}}'$, $vector'$
13. Device $\rightarrow$ Smartcard : $value_{\mathrm{D}}'$, $salt_{\mathrm{D}}'$, You get 5 of Clubs,
14. Device $\rightarrow$ Smartcard : I win \$15
15. Smartcard $\rightarrow$ Device : You win \$15, new balance: \$1560

Figure 1: **Protocol for High-Card.**

This could be achieved, perhaps, by equipping the device with a modem that could dial into a modem at the house. For example, to add \$100, the user would pay \$100 to the house, say using a credit card, over the phone or on the web. Then, the user would dial the house from the device, and the house would send a signed message to the card to increase the balance by \$100. A challenge/response protocol could be used to avoid replay. The smartcard verifies the signature of the house and increases the balance.

To cash out in this scenario, the user indicates to the house that he wishes to cash out, and the smartcard sends a signed message to the house, over the modem connection, indicating that the user cashes out a particular balance. The smartcard then sets the balance to zero, and the house issues a check for the amount to the user. Of course, measures are taken to protect against replay.

It would be nice to remove the assumption of a modem and a high bandwidth connection. In the next section, we present a protocols for adding cash and cashing out without direct communication between the device and the house.

6.1 Purchasing more credit on the card

We describe a protocol whereby a user can purchase more credit for his smartcard without a direct connection between the device and the house. We assume some low bandwith connection between the user and the house and between the user and the smartcard. For example, the user could make a phone call to the house, enter a credit card number and expiration date, and receive a short string of alphanumeric characters back. The same could be accomplished over the web. The actual transport does not matter, except for the limiting factor that there is no way for the house or the device to communicate thousands of bits to each other in a user friendly fashion.

Once the user has paid for the credit, there must be some way for the house to add the credit to the smartcard in such a way that the amount is added exactly once, even in the face of a malicious user who is trying to maximize his value. In our protocol, the smartcard authenticates the request from the house by verifying that only the house could have possibly generated some string. One way to achieve this is to establish a shared secret between the house and the smartcard.

However, every smartcard must have a different secret. Otherwise, compromising one smartcard could lead to impersonation of all the other smartcards. In addition we do not wish to store a large number of secret keys at the house.

The solution we employ makes use of psuedorandom functions. The house generates a *master key*, MK, which it uses to compute the secret keys on the smartcards. Every smartcard, SC comes with a unique serial number, SN, that is visible on the outside of the card. When a smartcard is manufactured, the house computes $K_{SC} = f_{MK}(SN)$, which is the pseudorandom function keyed by the master key and evaluated at the serial number. The secret key, K_{SC}, is stored in the tamper resistant portion of the smartcard. The key need not be saved by the house. It can be recomputed from the serial number and the master key.

Once the shared secret, K_{SC}, is established, we can authenticate messages using a secure Message Authentication Code (MAC). The best-known and most reliable MAC we are aware of is HMAC [3]. This is a function of a message and a secret key, such that only a party in possession of the secret key could have produced the MAC.

In the following protocol, we assume that messages are still signed and chained as explained in Section 4.1.3. Communication from the user to the SC is via the device implicitly. We use $500 as an example.

1. User $\rightarrow$ Smartcard: "Add $500"
2. Smartcard $\rightarrow$ User: N_0
3. User $\rightarrow$ House: N_0, SN, $500, "Add $500"
4. House $\rightarrow$ User: AUTH = $Trunc_{80}(HMAC_{K_{SC}}(N_0,$ SN, House, "Add $500"))
5. User $\rightarrow$ Smartcard : AUTH

In message 1, the user utilizes the GUI on the device to indicate that he wishes to add $500 to the card. In message 2, the smartcard issues a challenge consisting of an 80 bit random nonce, N_0. The nonce can be represented by 16 alphanumeric characters, which is a reasonable amount for the user to read and convey to the house. Once the smartcard issues the challenge, it locks up and refuses any message except a valid AUTH response corresponding to the amount in message 1.

In message 3, the user pays the house $500. This could be by credit card, check, cash, or any other form. In addition, he passes along the nonce and the serial number, which he reads off the outside of the smartcard. The entire transaction could take place by telephone if the house is willing to accept credit card payments by phone. Once the house receives the challenge, it computes K_{SC} from the serial number and the master key. It then computes the MAC of the nonce and the request to add $500. Finally, the result is truncated to contain only the first 80 bits. These are sent to the user as 16 alphanumeric characters.

The user enters AUTH into the device, which passes it along to the smartcard. The smartcard uses its stored secret key, K_{SC} to compute the truncated MAC on the request to add $500 and compares it to the one received from the house. If the computed value does not match the value from the house, then the card remains locked. Otherwise, the balance on the card is increased by $500. The protocol is illustrated in Figure 2.

After outputting N_0, it is important that the card lock up until it receives a correct AUTH from the user. This is to prevent parallel runs of the protocol, which could lead to potential attacks [1].

The protocol is not susceptible to replay. To successfully add any amount to the smartcard, the user must receive a new challenge from the smartcard, and the smartcard does nothing until the AUTH for that amount is received. The house only releases AUTH values for amounts that are paid, so replaying any of the messages in the protocol cannot result in stealing money from the house.

An adversary who does not possess the master key cannot produce a valid AUTH for an arbitrary smartcard. Furthermore, if someone breaks into a smartcard, he can only expose the secret key for that card, because keys are independent from each other, given the properties of pseudorandom functions. However, if the person breaks into a card, that card is pretty much compromised, so the scheme introduces no further loss. The important thing is that compromising one card does not give one the ability to add value to another card. In Section 7 we discuss ways of identifying and dealing with compromised cards.

Bellare *et. al.* [4] show that functions that give the desirable pseudorandom properties for f above can be easily constructed from simple hash functions. For our application HMAC is a reasonable choice.

6.2 Cashing out

In this section, we describe a protocol for removing value from the smartcard and receiving money. Once again we assume the bandwidth between the house and smartcard is limited by a user in the middle of the protocol. We assume the entire transaction takes place with a single phone call from the user to the house, but of course, this could be done over the web, or many other ways.

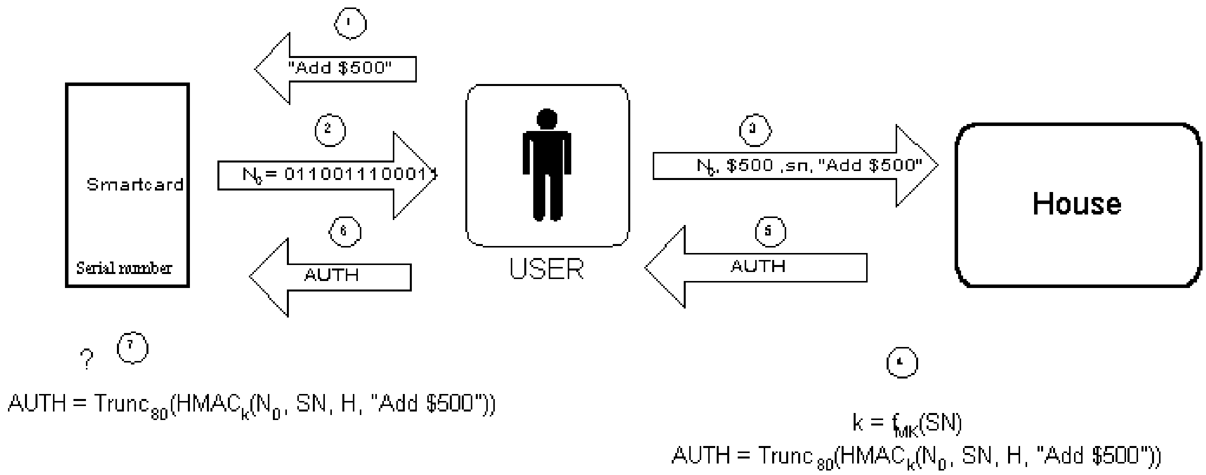

Figure 2: **Protocol to add value to the smartcard** In step 1, the user indicates to the smartcard that the amount to add is \$500. In step 2, the smartcard sends a random nonce, N_0, to the user. In step 3, the user sends the serial number on the smartcard and the nonce to the house and pays the house \$500. Step 4 shows that the house computes the smartcard key using the psuedorandom function, f and then the HMAC of the amount to add and the nonce, and then truncates to 80 bits. The result is AUTH, which is sent to the user and directly to the smartcard. In step 7, the smartcard performs the same operation using the stored key. If this computed amount matches the AUTH value, the smartcard credits the user with \$500.

We illustrate the protocol with an example. Assume the user wishes to cash out \$500 from the smartcard. The protocol is as follows:

1. User $\rightarrow$ House: Withdraw
2. House $\rightarrow$ User: N_0
3. User $\rightarrow$ Smartcard: N_0, "Withdraw \$500"
4. Smartcard $\rightarrow$ User: AUTH $= Trunc_{80}($ $HMAC_{K_{SC}}(N_0,$ SN, House,"Withdraw \$500"$))$
5. User $\rightarrow$ House: AUTH, SN, "Withdraw \$500"
6. House $\rightarrow$ User: \$500

This protocol is similar in spirit to the one for adding value. We use a scenario to describe how the protocol works. The user calls up the house on the phone and says that he wants to withdraw money. The house provides him with a challenge, which consists of a random nonce of length 80 bits, encoded in 16 alphanumeric characters. The user enters the 16 characters into the device, which feeds them to the smartcard, along with the amount, \$500. In message 4, if the smartcard does not have \$500, an error is returned. Otherwise, the smartcard deducts \$500 from the card and constructs the HMAC of the challenge and the amount to be withdrawn. The message is truncated to 80 bits to produce AUTH.

In message 5, the user reads AUTH to the house, along with the serial number on the back of the smartcard. The user also indicates that this authorization is for \$500. The house then constructs the user's key, K_{SC} from the serial number and computes the HMAC, and compares the resulting string to the authorization from the user. If they match, the house then mails a check for \$500 to the user. Of course, the payment could consist of a wire transfer or any other form of payment.

The *add value* and *cash out* protocols work because both the smartcard and the house have access to K_{SC}, while nobody else does.

7 Audit process

Securing systems requires more than just cryptography and sound protocols. Logging, audit and controls are an integral part of any complete system. In our system, it is important that the house monitor cash out requests very carefully. In the unlikely event that a particular smartcard is physically compromised, there is a danger that an attacker could manufacture money. Therefore, the house should log all cash out requests by serial number. If a particular card requests cash out with frequency or amount above a certain threshold, an alarm should be triggered. The suspected serial number should be added to a watch list, and if the behavior continues, an investigation may be required. An example of a countermeasure is to notify the user the next time he tries to cash out that his card is being replaced. The replaced smartcard should automatically be on a hotlist that is closely monitored.

Another possibility is to issue smartcards with expiration times. This limits the exposure of the smartcard to its valid period. This could also have side effects of increasing the *take* of the house because of money lost to expired cards. However, this may not go over very well with users.

8 Conclusions

We describe a system that utilizes the tamper-resistant nature of smartcards to enable a personalized gambling device. The protocols presented have the property that neither the house nor the user can cheat. In addition, we present protocols for adding or removing money from the smartcards over a low-bandwidth channel, such as a person on the telephone. These protocols are relevant even outside of the context of gambling.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, 1994.

[2] Accredited Standards Committee X9. *Working Draft: American National Standard X9.30-1993: Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry: Part 2: The Secure Hash Algorithm (SHA)*, 1993.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Crypto 96 Proceedings*, 1996.

[4] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. *Proc. 37th Annual Symposium on the Foundations of Computer Science*, 1996.

[5] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137. IEEE, 1982.

[6] G. Brassard, D. Chaum, and C. Crepeau. Minimum discloser proofs of knowledge. *Journal of Computer and System Sciences*, 37:156–189, 1998.

[7] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[8] S. Haber and W.S. Stornetta. How to time-stamp a digital document. In A.J. Menezes and S. A. Vanstone, editors, *CRYPTO90*, pages 437–455. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.

[9] Chris Hall and Bruce Schneier. Remote electronic gambling. *13th Annual Computer Security Applications Conference*, pages 227–230, December 1997.

[10] Kipp E. B. Hickman and Taher Elgamal. The SSL protocol. *Internet draft draft-hickman-netscape-ssl-01.txt*, 1995.

[11] A.J. Menezes, P. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[12] M. Naor. Bit commitment using pseudorandomness. In *Advances in Cryptology—CRYPTO '89 Proceedings*, pages 128–136. Springer Verlag, 1990.

[13] R. Rivest. The md5 message digest algorithm. *RFC 1321*, April 1992.

[14] Bruce Schneier and John Kelsey. Automatic event-stream notarization using digital signatures. *Security Protocols: International Workshop, Cambridge, UK*, pages 279–286, 1997.

[15] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. *Seventh USENIX Security Symposium*, pages 53–62, January 1998.