Secure Mobile Gambling

Markus Jakobsson[1], David Pointcheval[2], and Adam Young[3]

[1] Bell Laboratories, Lucent Technologies
Information Sciences Research Center
Murray Hill, NJ 07974, USA
[2] Dépt d'Informatique, ENS – CNRS
45 rue d'Ulm, 75230 Paris Cedex 05, France
[3] Lockheed Martin, King of Prussia, PA, USA

Abstract. We study lightweight and secure gambling methods, and propose a general framework that is secure against various "disconnection" and "payment refusal" attacks. Our method can be employed for single- and multi-player games in which players are independent, such as slot machines, roulette and blackjack. We focus on "open card" games, i.e., games where the casino's best game strategy is not affected by knowledge of the randomness used by the players (once both or all parties have committed to their random strings.) Our method allows players as well as casinos to ascertain that the game is played exactly according to the rules agreed on, including that the various random events in fact are random. Given the low computational costs involved, we can implement the games on cellular phones, without concerns of excessive computation or power consumption.

Keywords: Fair, gambling, lightweight, Merkle, publicly verifiable, robust

1 Introduction

It is anticipated that a large part of the future revenue in the communication industry will come from services related to entertainment. It is believed that cell phones will play an increasingly important role in this trend, given their large market penetration and portable nature (making them available whenever boredom arises.) Entertainment-related services can be categorized into services that *locate* entertainment, and services that *are* entertainment. In this paper, we will only consider the latter type, and in particular, only one particular type of entertainment services, namely gambling.

Putting local legal restrictions aside for a moment, we argue that cell phones are perfect vehicles for gambling, since they by nature are portable, can communicate, and have some computational abilities. Furthermore, cellular phones are already connected to a billing infrastructure, which could easily be augmented to incorporate payments and cash-outs. With improved graphical interfaces – which we can soon expect on the market – cellular phones can become very desirable "gambling terminals." However, if *mobile gambling* were to proliferate, there is a

substantial risk that some providers would skew the probabilities of winning in their favor (and without telling the gamblers). While this problem already exists for "real-world" casinos, it is aggravated in an Internet and wireless setting. The reason is that with many small service providers, some of which may reside in foreign jurisdictions, and some of which may operate from garages, auditing becomes a more difficult task. On-line services can also change their physical location if "the going gets rough", making the task of law enforcement more difficult.

On the other hand, while the honesty of real-world casinos can only be verified using auditing methods, it is possible to guarantee fairness in an on-line setting using cryptographic methods, allowing for *public verifiability* of outcomes. The idea is first to let the randomness that decides the outcome of the game be generated by both the casino and the portable device of the consumer, according to the principles of coin-flipping over the phone [4]. Then, in order to avoid problems arising from disconnections (both accidental and intentional), it is necessary to allow for an efficient recovery of the state of an interrupted game. This state recovery must be secure against replay attacks (in which a winner attempts to collect twice), and must be auditable by third parties. Finally, in order to make the service feasible, it must be computationally lightweight, meaning that it will not demand excessive resources, and that it can be run on standard cellular devices.

We propose a framework that allows games to be played on computationally restricted devices, and automatically audited by all participants. Our solution can in principle be applied to obtain any game – expressed by a function f on the random inputs. (However, due to considerations aimed at avoiding game interruptions caused by disconnected players, we only consider games in which the players are independent.) While in theory this functionality can be obtained from a scheme in which signatures are exchanged (potentially using methods for a fair exchange [2, 14]), such a solution is not computationally manageable in the model we work. Thus, our solution is based on the use of hash function evaluations alone for all but the setup phase, and utilizes a particular graph structure for optimal auditing speed and minimal communication bandwidth. The use of number theoretic building blocks is limited to the setup phase as far as players are concerned. Players may either perform this computation on a computationally limited device such as a cellular phone, where it takes time but is still feasible, or on a trusted computer, such as a home computer. Our main contribution lies in proposing the problem, elaborating on the model and architecture, and proposing efficient protocols to achieve our goals.

We show how to make payments implicit, by causing the function f to output digital currency according to the outcome of the game. That is, the output will constitute one digital payment to the casino and another to the player(s), where the amounts depend on the outcome of the game, and may be zero. We say that a game is *fair* if it guarantees all parties involved that the outcome of a completed game will be generated according to the rules agreed upon, including a correct distribution of the random outcomes.

Moreover, given the risk for disconnection – both accidental and intentional – we must make sure that this does not constitute a security loophole. Consequently, an interrupted game must always be possible to restart at the point of interruption, so that neither casinos nor players can profit from disconnections by interrupting and restarting games to their favor. We say that a solution is *robust* if it always allows the completion of a game for which one party has received a first transcript from the other party – independently of whether a disconnected party agrees to restart the protocol or not. (We note that the completion of the game is not the point at which the participants learn about the outcome, but rather, the point at which their corresponding payments are issued.) Thus, instead of considering an opponent's strategy for the *game played* (e.g., blackjack), we must consider the "meta game" played. One component of the meta game is the actual game; other components are the strategies for disconnection, state reporting, and profit-collection. We show our solution to be robust and fair.

Our solution allows the transfer of state between various devices operated by one and the same player. We describe how to transfer the state securely, and without direct interaction between the devices in question. (In other words, the state is transferred via the casino, posing us with additional security considerations, in that we must guarantee a continuous sequence of events, and prohibit "rewinding".)

Outline: In section 2, we present the constraints we must observe, corresponding to our model for communication, computation, and trust. We also detail the goals of our efforts, and discuss practical problems. In section 3, we explain the required setup. In section 4, we show how a game is played (including how players perform game-dependent decisions, cash in profits, and perform conflict resolution, if necessary.) We also explain how to transfer the state between various devices operated by one and the same player, e.g., a home computer and a cell phone. We note that the transfer does not require any interaction between the devices between which the state is transferred. We elaborate on the security properties of our scheme in section 5.

2 Constraints and Goals

Device Constraints. There are two types of constraints: those describing the typical setting of the game, and those describing the computational model. While the former relates to efficient implementations (and corresponds to maximum costs of building blocks), the latter is concerned with the security of the protocol (and therefore the minimum security of the building blocks.)

In terms of *typical* device constraints, we assume that players have very limited computational capabilities. Without clearly describing what operations we consider feasible, we exclude the common use of all number theoretic operations for all but a setup phase. Also, we assume a limited storage space for players, limiting the amount of storage required by the application to a few hundred bytes. We may achieve this by shifting the storage requirements to the casino,

on which we will assume no *typical* device constraints. Alternatively, we may construct the randomness associated with each node as the output of a pseudo-random number generator taking a seed and the node identifier as its input. This allows a local reconstruction of values, either routinely or in case of conflict. See [3] for a good overview of possible constructions.

In terms of security, we make standard cryptographic assumptions (as described by poly-time Turing Machines) for both players and casinos. In particular, we will make standard assumptions regarding the hardness of inverting or finding collisions for particular functions, as will be apparent from the protocol description.

Adversarial Model. The aim of an adversary may either be to increase its expected profit beyond what an honest set of participants in the same games would obtain *or* to minimize the expected gains of a victim in relation to an honest setting.

We consider players and casinos as mutually distrustful parties, and assume that any collusion of such participants is possible. In particular, we allow any such collusion of participants to perform any sequence of malicious operations, including setups, game rounds, disconnections, and bank deposits. We do not allow the adversary to consistently deny a player access to casinos, but do allow temporary access refusals. (This corresponds to a sound business model, since the casino's profits depend on continuous availability.) We assume that the bank will transfer funds between accounts in accordance with the protocol description. We also assume that the state kept by the different participants will not be erased, as is reasonable to assume by use of standard backup techniques. However, and as will be clear from our protocol description, we do not require the recoverable state to be constantly updated, as we allow recovery of a current state from an old state.

Game Constraints. We focus on games in which a player can play with "open cards" without this reducing his expected profit. Here, *open cards* corresponds to publicly known randomness, and not necessary to cards *per se*, and means that as soon as the player learns the random outputs or partial outputs of the game, so does the casino (in a worst case.) We do allow the participants to introduce random information during the course of the game, as we allow the use of values associated with the decisions to derive random values. However, this only allows the drawing from known distributions, and so, cannot model drawing card from a deck from which some cards have already been drawn, but it is not known which ones. This constraint would rule out games such as poker, where it is important that the hand is secret. However, our constraint is one purely motivated by efficiency considerations, and it is possible to implement poker, and any game in which one cannot play with open cards, by means of public key based protocols. (A mix network [6, 1, 9], may, for example, be used to shuffle a deck of cards.)

3 Setup

To optimize the game with respect to the communication and computation overhead, we use a tree-based hash structure for commitments to randomness and game decisions. For each player, and each type of game offered by the casino, two such structures will be computed – one for the player, and one for the casino. (We note that it is possible to construct a new game from two or more traditional games, where the first decision of the player in the new game selects what traditional game to play. This would allow the use of the same structure for multiple games.)

To minimize the amount of storage required by the players, the casino may store these structures, and send over portions of them as required. We note that the player structures will be stored in an encrypted manner, preventing the casino from evaluating the game function on the structures until the game is initiated by the player. In case of conflict (where the player believes that he got the incorrect data from the casino) it is important that the player can locally generate the data himself, given his secret seed and a counter corresponding to the contested data.

Building Blocks. Let (E, D) be a secure probabilistic symmetric cipher [7, 10], with semantic security. Furthermore, let h be a hash function for which collisions are intractable to find, and which therefore constitutes a one-way function [12], hence it is hard to invert on average (i.e., for any poly-time ensemble A, the probability that $A(h(X))$ is an inverse of $h(X)$ is small, where X is drawn uniformly from the domain of h). Furthermore, let C be a perfect commitment. This may be a hash function which hides all partial information [5]. Finally, we assume the use of some signature scheme that is existentially unforgeable [8].

Nomenclature: We use *game type* to correspond to the rules governing the interaction between players and casino. An example of a game type is therefore *blackjack*. We refer to particular instances of a game type as *games*, or *game rounds* (where the latter signifies that a complete instance of a game corresponds to multiple rounds, between which there are state dependences.) Each game, or game round, may consist of some number of consecutive *moves*, each one of which allows the players and the casino to commit to a decision. A *game node* is a block of data that determines the randomness contributed to a game round by its holder. We refer to values of a game node that encode possible decisions to be made as the *decision preimages* for the game. Finally, a *game tree* is a collection of game nodes, arranged in the hierarchy of a tree for purposes of efficiency.

At the time of setup, the player and the casino agree on the size of the tree, where the number N of nodes corresponds to the maximum number of rounds of the game type in question that they can play without re-performing the setup.

Game Nodes. (See figure 1a) Different games require different numbers of user choices to be made. Slot machines allow for few or none; blackjack for several;

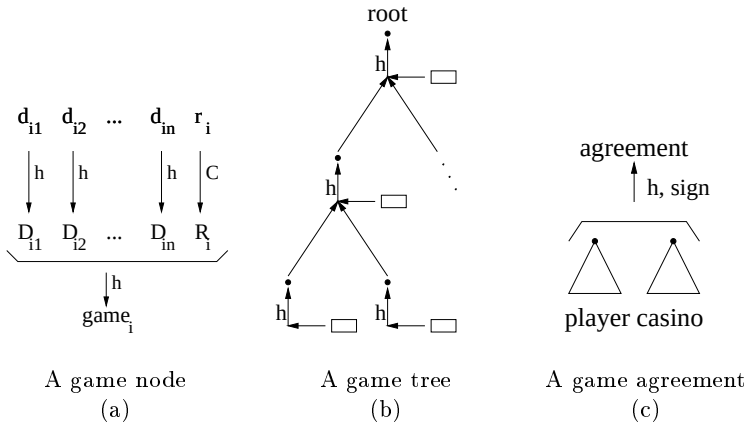

Fig. 1. Game Structures

and roulette for a tremendous number – given the vast number of combinations with which a player can bet. We use a method inspired by Merkle signatures ([13, 11]) to encode and implement player decisions. More particularly, for each type of game in question, we let players and casinos commit to decisions by revealing *decision preimages* according to some encoding scheme, and similar to how bits are committed to in Merkle signatures. In the setup-phase, the player selects some n uniformly distributed random numbers $d_{i1}, \ldots, d_{in}$, for each node i of the tree (each such node corresponding to one round of the game); these allow him later to make choices by revealing preimages in one or more moves, according to some scheme encoding his decisions. The player also selects a random number r_i uniformly at random for each node. All of these random values are assumed to be of size 160 bits or more, to avoid the birthday paradox problem. Indeed, in case of a collision, the opponent could claim that another decision has been taken by the player. The player computes a value $game_i = \langle h(D_{i1}, \ldots, D_{in}), R_i \rangle$, where $D_{ij} = h(d_{ij})$ and $R_i = C(r_i)$. We denote $preimage_i = (d_{i1}, \ldots, d_{in}, r_i)$ the secret preimage to $game_i$.

Game Trees. (See figure 1b) The player computes a structure S_{player} consisting of N nodes, each one of which is connected to one parent node (except the root); two children nodes (except the leaves), and one *game node*, which is described by the value $game_i$ (described above) for the ith such node. We enumerate game nodes according to their depth-first traversal order in the tree. Each node in the tree has a value which is the hash of all its children's values; of its game node value; and of a descriptor *game* that describes what game type that it corresponds to. Let $root_{(player,game)}$ be the value describing the root of the tree for the game in question.

Each player constructs one such value $root_{(player,game)}$ for each game type he wishes to be able to play, and the casino prepares a similar structure (unique

to the game type and the player in question) for each player structure. Let $root_{(casino,game)}$ describe the root of this tree. (We note that the structures may be of slightly different formats if the player and casino have different number of maximum choices per round.)

Let $agreement_{(casino,player)}$ be a document consisting of the above root values for the player and the casino, a hash value on the game function f_{game}, and of signatures on this information by both the casino and the player – see figure 1c (We assume the use of certified or otherwise publicly registered public keys.)

Storage. The above mentioned value, $agreement_{(casino,player)}$, along with relevant certificates, is stored by both the player and the casino. The player needs not store the value on his portable device, but only in some manner that allows him to retrieve it in case of a conflict.

The player may store his game trees on his device, or may encrypt these in portions corresponding to game nodes, and have these stored by the casino. We focus on the latter case, and let $E_i = E_{K_{player}}(preimage_i, red_i)$ be the encryption of $preimage_i$ under the symmetric key K_{player}, using redundancy red_i of sufficient length to determine with an overwhelming probability whether a ciphertext is correctly decrypted. We may choose $|red_i| = 80$, and assume that the counter i be a part of red_i.

The casino stores records of the format $(i,\ E_i,\ game_{player,i},\ game_{casino,i})$ along with a counter cnt indicating what games have been played. This counter is specific to the player and the type of game associated with the node. (We simplify our denotation by considering only one counter, but note that the scheme tolerates any number of these.) The casino also stores all the functions f_{game}.

The key K_{player} is stored by the player in his portable device, along with the counter cnt. The player also keeps a backup of the symmetric key, whether in the form of a file on his home computer, or in terms of a passphraze used to generate the key. Furthermore, the player stores either the functions f_{game} of the games he is interested in playing, or merely hash values of these. It is possible (but not necessary) for the player also to have the value cnt backed up with regular intervals, e.g, on a home computer.

The bank will store elements corresponding to payment requests, allowing it to detect duplicates and inconsistencies. We will elaborate on the format of this later, after having presented our suggested integrated payment scheme.

State Compression. If the preimage $preimage_i = (d_{i1}, \ldots, d_{in}, r_i)$ is selected by the player as the output of a PRNG whose input is $(seed_{player}, game_i)$, then it can be generated (and re-generated) locally when required. Depending on the difference in speed and power consumption between the PRNG and the decryption function, and taking the communication costs into consideration, it may be beneficial not to use the casino as a repository for encrypted game nodes, but always to recreate these locally, from the seed, when needed.

Certificates of Fairness. Our model allows auditing organizations and other entities to review the game functions f_{game} (or crucial portions of these) to ascertain that they correspond to fair games. Here, *fair* is simply used to mean "in accordance with the disclosed rules". The rules specify the different events corresponding to the outcomes of the games, their probabilities of occurrence, and the costs and payoffs associated with the game. If an auditing entity decides that the game described by f_{game} is fair in this sense, it can issue a digital certificate on f_{game} along with a description of the rules. This certificate may either be publicly verifiable, or verifiable by interaction with some entity, such as the auditing organization. Users may verify the fairness of games by verifying the validity of the corresponding certificates.

4 Playing

Request. To initiate a game, the player sends a request $(player, game)$ to the casino, where *player* is the name or pseudonym of the player, and *game* is the name of the game the player wishes to initiate. We note that the request is not authenticated. We also note that games will be selected in a depth-first manner (which we show will minimize the communication requirements.) The games will be enumerated correspondingly.

If *player* has performed a setup of the game *game* and some unplayed game nodes of this type remain, then the casino returns a message

$$(E_{cnt}, game_{player,cnt}, game_{casino,cnt});$$

otherwise he returns a random string of the same length and distribution.

The player decrypts E_{cnt} to obtain $preimage_{cnt}$ and cnt, and verifies the correctness of the redundancy.

Playing a Game. A game is executed by performing the following steps (we later consider what to do in case of communication disconnection):

1. The player initiates a game by sending the value $r_{player,cnt}$ to the casino. The casino verifies that this is the correct preimage to $R_{player,cnt}$ and halts if not. (We note that $R_{player,cnt}$ is part of $game_{player,cnt}$, which is available to the casino.)
2. The casino and the players take turn making moves:
   (a) The casino reveals decision preimages encoding its move.
   (b) A man-machine interface presents the choices to the human user, collects a response, and translates this (according to some fixed enumeration) into what decision preimages to reveal. These values are sent to the casino.

   The above two steps are executed one or more times, corresponding to the structure of the game. In the above, the recipient of values verifies the correctness of these. If any value is incorrect, then the recipient requests that the value is resent. All preimages are temporarily stored (until the completion of step 4 of the protocol) by both casino and player.

3. The casino responds with $r_{casino,cnt}$, which is verified correspondingly by the player.
4. The function f_{game} is evaluated on the disclosed portions of $preimage_{payer,cnt}$ and $preimage_{casino,cnt}$. (We discuss requirements on the function below.) The output is presented to the player and the casino, and the appropriate payment transcripts are sent to the bank. (We elaborate on this aspect later.)
5. The player and the casino updates the counter cnt, along with other state information.

Evaluation. The outcome of the function f_{game} depends on some portion of the values in $preimage_{player,cnt}$ and on $r_{casino,cnt}$. In games where the randomness is not public until the end of the game (e.g., when the hand is shown) it also depends on the actual *values* of the decision preimages given by the players and the casino (as opposed to the *choices* alone). This also holds if step 2 above consists of several moves (i.e., an iteration of the two participants' disclosing of decisions). In such a case, h needs to satisfy the same requirements as C does, i.e., be a perfect commitment that hides all partial information. Using the decision preimages to derive randomness (used in combination with values disclosed in step 3 to avoid predictability), allows the introduction of new random values throughout the game.

When we say that a result depends on a value, we mean that one cannot compute any non-trivial function of the result value without access to the value on which it depends. (This is meant in a computational sense, and not in an information theoretical sense, and so, is relative to the hardness of the cryptographic primitives employed.)

Example: Slot Machines. Slot machines provide the probably simplest setting in that one only needs two random strings, one for the player and one for the casino, where an XOR of these values may be used to directly determine the outcome of the game. For slot machines that allow one or more wheels to be locked and the other rotated again, this simply corresponds to letting a first-round decision of a game node encode "keeping" an outcome from the previous game node. The result of stopping a wheel from spinning at some point can be ignored in terms game impact, as it does not alter the distribution of the outcome.

Example: Variable Length Decisions. In roulette, the player can place bets on various portions of the board, in a large number of configurations. It is possible either to limit the maximum bet to keep the number of combinations down, or to use several consecutive game nodes to express one bet. Let us consider how to do the latter in a secure fashion.

Let one of the decision preimages, when revealed, mean "link with next game node", and let another decision preimage mean "do not link with the next game node". Clearly, the player will only reveal one of these. After the conclusion of the game, one has to deposit all game nodes in a sequence, along with the game node of the previous game (unless already known by the bank), and each of these

game nodes need to have exactly one of the above mentioned preimages revealed. This allows the player to encode arbitrary-length decisions, as his decision will be encoded by all the preimages of all the "linked" game nodes.

Whether multiple game nodes are linked or not, we have that if a game allows variable length decisions, then either there must be some decision preimages that encode the length of the decision, or both casino and players need to submit game transcripts to the bank, to avoid that only a prefix decision is submitted.

Example: Multi-Player Games. In our main construction, we only consider games where the strategies and games of different players are independent of each other. This, however, is purely for reasons of service continuity (recognizing that users relatively often get disconnected when using mobile devices.) To play a multi-player game where the outcome of each player's game depends on the strategies of other players, each player may use one portion of the decision preimage field to encode the public values of the game nodes of the other players participating in the game. The game would then start by a round in which all players open up preimages corresponding to their view of the game nodes of the other players, and then by the game, as previously described.

Example: Drawing Cards Face-Down. In poker, the players of the game (of which the casino may be one) take turns making decisions (specifying what cards to keep, and how many new cards to request), and obtain cards from a common deck. The values of these cards are not publicly available until the end of the game. The decision preimages are therefore used both to commit to the decisions and to provide randomness determining what cards are drawn. In a situation where the casino plainly deals, and is trusted not to collude with other players, it is possible to let the casino know the hands of the different players, which allows for a simple solution, but which raises the concern of collusions between players and casino. To avoid this, it appears necessary to employ public key based methods. We do not consider such solutions herein, due to the computational restrictions we set forth, but notice that with more computational resources, such solutions would be possible.

Handling Disconnections. As will be seen from the description of the payment generation, the player commits to performing the game in step 2 of the protocol for playing the game. Therefore, disconnections are handled differently depending on the stage of the protocol execution.

The casino will take a relatively passive role in reacting to disconnections, as it will ignore disconnections before the execution of step 2 of the protocol (and merely rewind its internal state to what it had before the initiation of the first protocol step). Disconnections during step 2 are handled by the bank acting as an intermediary between the player and casino (if wanted by the player), or by charging the player according to the most expensive outcome given the transcript seen (if the player refuses connection.) The casino will handle disconnections after step 2 by executing its parts of steps 4 and 5 of the protocol. It also stores the player's decision preimages, if received.

If the player detects a disconnection of the game before executing step 2 of the protocol, then he will rewind his state to the state held at the beginning of the protocol. If the player detects the disconnection after that stage, then he will request a *replay*, and perform the following protocol:

1. The player sends the casino the string

$$(player, cnt, r_{player,cnt}, \mathcal{D}_{casino}, \mathcal{D}_{player}).$$

   In the above, $\mathcal{D}_{casino}$ represents the decision preimages of the casino (recorded by the player), and $\mathcal{D}_{player}$ those of the player. (Note that these are the choices that have already been made. The player does not get to make a new game decision for the reconnected game, as this is just a continuation of the disconnected game.)
2. The casino verifies the correctness of the received values with respect to the game nodes $game_{casino,cnt}$ and $game_{player,cnt}$. If not all values are correct, then it halts.
3. If the casino has previously recorded decision preimages other than those received in the current protocol, then it selects the set $\mathcal{D}'_{player}$ that maximizes its benefit.
4. The participants perform steps 3-5 of the game-playing protocol, both of them sending payment invoking transcripts to the bank. (If the bank receives different transcripts, it will perform a particular type of conflict resolution before performing the payments – we describe this below.)

If the above fails, the player will attempt it with the bank as an intermediary.

Payment Generation. In the following, we show how the bank can determine the charges and the credits by evaluating the game function on the provided transcripts. The transcripts determine both who won, and how much – the latter may depend both on the outcome of the game, and on decisions by players and casino (such as how much is bet.)

A payment request *by the casino* consists of

1. the player identifier ($player$), the value cnt, the value $game_{player,cnt}$, and the player decision preimages $\mathcal{D}_{player,cnt}$,
2. all values on the path from the game node $game_{player,cnt}$ up to the root $root_{player,game}$; the game nodes $game_{player,i}$ of every node in the tree that is a sibling with any of the nodes on the above mentioned path; and the value $agreement_{casino,player}$.

The bank checks the consistency of all of these, and verifies that they have not already been submitted (in which case it runs a particular conflict resolution protocol, detailed below). The bank then transfers funds from the player's account to the casino in accordance with the cost of playing a game as governed by the rules, the decision preimages $\mathcal{D}_{player,cnt}$. (We note that the verification does

not include verifying who won the game, as we take the approach of charging for each game, including games in which the user wins.)

In the above, only the first triple of values $(player, cnt, \mathcal{D}_{player,cnt})$ is sent, unless the other values are requested by the bank. The bank stores all values received, and only requests the further information if it is not available.

A payment request *by the player* consists of

1. the player identifier $(player)$, the value cnt, the value $game_{player,cnt}$, and the player decision preimages $\mathcal{D}_{player,cnt}$,
2. the values $r_{player,cnt}, r_{casino,cnt}$, and the casino decision preimages $\mathcal{D}_{casino,cnt}$
3. all values on the path from the game node $game_{player,cnt}$ up to the root $root_{player,game}$; the game nodes $game_{player,i}$ of every node in the tree that is a sibling with any of the nodes on the above mentioned path; and the value $agreement_{casino,player}$.

As above, the last portion is not sent unless requested. If the casino is storing information for the player, and the information is requested by the bank, then the casino will be contacted to give the information. If it refuses, then a special conflict resolution is run, see below. When all the necessary information is received, the bank verifies the same, evaluates the function f_{game}, and determines what the pay-out is. It then verifies whether this transcript has already been deposited. If it has, then it runs the conflict resolution protocol below. Otherwise, it credits the accounts accordingly.

In the above, the bank indexed payment requests by the value $r_{player,cnt}$, which has to be submitted for all requests. We note that the bank may require both casino and player to deposit the transcript corresponding to a game in order to avoid "partial" transcripts to be deposited. (With a partial transcript we mean a transcript where some of the decision preimages revealed by player or casino are not reported.) Depending on the nature of the game, deposits may routinely be performed by both parties, or be performed on demand by the bank.

Conflict Resolution. Conflict resultion is performed in the following cases:

- **Two or more identical "deposits" for the same game.**
  If more than one payment request for a particular game is deposited, then only the first is honored, and all duplicates are ignored.
- **Two or more different "deposits" for the same game.**
  If the bank receives correct transcripts corresponding to two or more different outcomes of a game, i.e., transcripts for which there are different sets of decision preimages recorded, then it decides as follows. If there are two or more different decision transcripts of the casino, but consistent versions for the player decision transcripts, then it judges in favor of the player. If, on the other hand, the casino preimages are consistent, but the player images are not, then it judges in favor of the casino. If neither is consistent, then alternate resolution mechanisms (not described herein) are necessary.

- **Incomplete deposit.**
  If a transcript does not contain all decision preimages required to complete the game, then the bank will rule in favor of the participant submitting the transcript after having tried to obtain the transcript from the other participant, and failed to have the participants complete the game with the bank as an intermediary.
- **The casino refuses to disclose values.**
  If the bank requests path information from a casino during the deposit by a player, and the casino refuses to provide this information, then the player's account is credited with the amount corresponding to the deposited game transcript (possibly after some reasonable hold period.) The casino's account is charged the same amount, plus possible fines.
- **Player out of funds.**
  If the casino deposits a game transcript for which there are insufficient funds, it is notified about this, and may (but is not required to) temporarily lock the access of the player to the games. (In fact, the bank can alert the casino of a low player balance if this falls below a particular preset level, which has to be established by agreement between the player ad casino during account establishment, or by implicit agreement for playing any particular game.) Any deposits made after the casino has been notified of the player being out of funds are put on hold, and are credited and charged only after a sufficient balance is available.
- **Casino out of funds.**
  If the casino's balance falls below a preset level, then each player depositing transcripts is paid according to the outcome, but barred from any further deposits from the casino (until service by the casino is re-established). The player is notified of this condition, and his device temporarily disables the gambling service. If the casino's balance falls below a second and lower level, then all registered players are notified that no further deposits will be accepted after some cut-off time, and the player devices disable the service.

Transferring State. We note that there are only two parameters that need to be transferred between devices in order to allow the user to transfer the state between devices. One is the secret master key used to decrypt the received transcripts; the other is the counter determining what games have been played and which ones remain to be played. The master key can be installed on both user devices during setup, or may be generated on the fly from a passphrase. We can allow the casino to store the counter, and send this to the player for when requested. While this would enable the casino to perform rewinding attacks, these can be defended against as follows: If the player notifies the bank of the counter at the end of each game or sequence of games, the bank can verify that the corresponding transcripts are deposited by the casino within some short period of time (shorter than the period between two game sessions with an intermediary state transfer.) If the casino deposits two different game nodes (potentially with different outcomes) then only the first is accepted. This prevents the bank from abstaining from performing deposits, and performing a rewind attack. To avoid

the user from blocking casino deposits by the above mechanism, one can require the casino to verify with the bank that they have a consistent state before the casino allows the transfer of state.

5 Security

We state the security properties of our scheme, and provide correctness arguments.

Public Verifiability. Assuming the non-forgeability of the signature scheme and that the hash function is a one-way function [12], our scheme satisfies *public verifiability.* This means that a third party (such as the bank) is always able to determine who won a particular game, given the corresponding game nodes with appropriate preimages revealed, and the paths from the game nodes to the root.

Since all game nodes are connected to a binary tree (each node of which is associated with a game node by means of a hash image of the latter), it is not possible to replace or alter a game node without finding a hash collision for at least one place on the path from the game node to the root. Therefore, since the signature on the set of roots cannot be forged, it is not possible for one party to replace a game tree signed by the other. Furthermore, he can also not replace a game tree signed by himself, since the opponent has a copy of his original signature, and can submit that to the bank as evidence of the bait-and-switch attempt. Therefore, a completed game (corresponding to an honestly submitted transcript of the game) can always be evaluated by a third party, who can determine the outcome of the game.

Fairness. Assuming the collision-freeness of the hash function h employed for the hash-tree, a hash function C the hides any partial information for committing the random coins, and the semantic security of the cipher, the game will be fair in that its outcome will be determined based on the agreed-upon rules, and on random strings of the correct distribution.

A participant commits to a game (without committing to *play* the game) by selecting a string, chosen uniformly at random from the set of strings of the appropriate length. The game is evaluated by evaluating the agreed-upon function (whether certified or merely recorded with the bank) on the two or more random strings provided by the two or more participants. The game function uses a random string that is a combination of the provided random strings. Therefore, if at least one of the strings is chosen uniformly at random, the output will be generated according to the agreed rules. If a participant does not select his string uniformly at random, this only provides an advantage to the opponent. Assuming that the cipher is semantically secure, it is infeasible for the casino to determine the preimages of a player's game node from the information he stores; therefore, the casino cannot obtain an advantage (in making his decisions) from analysis of the stored information. Assuming the partial information hiding of the commitment C, it is not possible for either party to perform a bait-and-switch operation, having seen part of the game.

Robustness. As soon as a participant has committed to playing a game, it is possible for the bank to determine how to transfer funds according to the outcome of the game. If a participant withholds information from the bank, this cannot financially benefit him.

We have already established that the game is publicly verifiable. If a player halts the game before step 2 of the protocol for playing a game, he cannot guess the outcome of the game with a better probability than what he could before the beginning of the game. If he halts during step 2, the deposited transcript can be evaluated by the bank, and will be charged according to the worst possible outcome for the player, unless the player submits information that allows the continuation of the game (in which case we say that the game is not halted, but merely executed with the bank as an intermediary.) If the player halts after step 2, the casino has all information required to perform a correct deposit. The casino cannot guess the outcome of the game with better probability than before the beginning of the game after having executed the first step of the protocol for playing a game. If the casino halts in the middle of step 2 or before concluding step 3, the game can be continued (if desired by the player) with the bank as an intermediary, and so, there is no financial incentive for the casino to do so. If the casino does not send the correct encrypted game node from its repository, the player will generate the information locally.

Conclusion

We have proposed an architecture allowing a wide array of games to be played on devices with severe computational limitations. Our model is rather cautious in that it allows arbitrary disconnections, an aspect seldomly factored into high-level protocol design. Our solution, which is shown to be robust under these circumstances, allows for both single-player and multi-player games.

Instead of considering the security and robustness of the game played, we consider these aspects of the meta-game in which the actual game played is one portion, and other decisions form another portion. Aspects belonging to this latter portion is whether to disconnect, and how to report profits to the bank, among other things.

An open problem is how to efficiently implement games based on drawing cards without repetition, and where there are at least two participants, both of whom keep their hands secret for some portion of the game.

Acknowledgments

Many thanks to Daniel Bleichenbacher for helpful suggestions.

References

1. M. Abe. Universally Verifiable Mix-Net with Verification Work Independent of the Number of Mix-Servers. In *Eurocrypt '98*, LNCS 1403, pages 437–447. Springer-Verlag, Berlin, 1998.

2. N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In *Proc. of the 4th CCS*. ACM Press, New York, 1997.
3. D. Bleichenbacher and U. Maurer. Directed Acyclic Graphs, One-way Functions and Digital Signatures. In *Crypto '94*, LNCS 839, pages 75–82. Springer-Verlag, Berlin, 1994.
4. M. Blum. Coin Flipping by Telephone: a Protocol for Solving Impossible Problems. In *Crypto '81*, pages 11–15. ECE Dpt, UCSB, Santa Barbara, CA 93106, 1982.
5. R. Canetti. Towards Realizing Random Oracles: Hash Functions that Hide All Partial Information. In *Crypto '97*, LNCS 1294, pages 455–469. Springer-Verlag, Berlin, 1997.
6. D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.
7. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
8. S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptative Chosen-Message Attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.
9. M. Jakobsson. Flash Mixing. In *Proc. of the 18th PODC*, pages 83–89. ACM Press, New York, 1999.
10. J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. In *Proc. of the 32nd STOC*. ACM Press, New York, 2000.
11. L. Lamport. Constructing Digital Signatures from a One-Way Function. Technical Report CSL 98, SRI Intl., 1979.
12. M. Luby. Pseudorandomness and Cryptographic Applications, Princeton University Press, page 27, 1996.
13. R. Merkle. A Certified Digital Signature. In *Crypto '89*, LNCS 435, pages 218–238. Springer-Verlag, Berlin, 1990.
14. S. Micali. Certified e-Mail with Invisible Post Offices. Presented at the 1997 RSA Security Conference, 1997.
15. A. C. Yao. Protocols for Secure Computations. In *Proc. of the 22nd FOCS*, pages 160–164. IEEE, New York, 1982.