Fair e-Lotteries and e-Casinos

Eyal Kushilevitz[1]⋆ and Tal Rabin[2]

[1] Department of Computer Science, Technion, Haifa 32000, Israel.
eyalk@cs.technion.ac.il.

[2] IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, USA.
talr@watson.ibm.com.

Abstract. In this paper we provide protocols for *fair* lottery and casino games. These fair protocols enable to remove the trust from the casino/lottery without resorting to another trusted third party, by allowing the user playing the game to participate in the generation of the specific run of the game. Furthermore, the user is able to verify the correctness of the execution of the game at the end of the run. On-line lotteries and on-line casinos have different properties and we address the needs of the two different types of games.

Keywords: e-lotteries, e-casinos, delaying functions, fair lotteries, publicly verifiable lotteries

1 Introduction

On-line gaming is a multi-billion dollar, growing industry. There are hundreds of web sites that offer various kinds of games ranging from simple lotteries to full online-casinos (where you can find most of the games that are found in real casinos like blackjack, video-poker, slot-machines etc.). The basic question that is addressed in this work is how can a user trust such a site for playing in a "fair" way. On an intuitive level, a game is fair if the chances of the user to "win" are as published by the casino owner (unfortunately, some web sites do not even bother to publish this information). In some cases, users trust the particular on-line casino based on its reputation. We note however that this should be done with caution.[1]

The first distinction that we make is between *interactive games* and *lotteries*. The typical scenario in an interactive game is a player who plays a game with the casino (a typical, popular game is blackjack). The fact that the game is interactive by its nature allows for using (interactive) protocols so as to guarantee

⋆ Most of this research was done while the author was a visiting scientist at the IBM T.J. Watson Research Center.

[1] For example, the official web site of the New-York lottery is www.nylottery.org while if you enter www.nylottery.com you get a different web-site that until recently used to offer lotteries.

the fairness of the game. Such protocols are presented in Section 2. The main, simple idea is to let the player influence the choice of randomness for the game. Lotteries, on the other hand, are characterized by very large number of users participating (potentially, in the millions); moreover, these users are not on-line for the whole duration of the game. Hence, the type of protocols that one can employ is much more restricted. Another difference is that lotteries usually span a relatively long time (e.g., a week from the time that users can start buying tickets until the time that the winner is announced). It is therefore required that the "fairness" of the game will withstand the long time that is available for the "bad" players or lottery house to bias the outcome (and that the fact that a user is not on-line at a certain time cannot be used to discriminate against her).

The issue of fair (or "publicly verifiable") lotteries, was addressed by Goldschlag and Stubblebine [3]. They use so-called "delaying functions" to design a lottery protocol. Informally, delaying functions are functions which are not hard to compute but still require a "significant" amount of time to complete the computation. It is assumed that by appropriately selecting the parameters of the function it can be tuned so that its evaluation will take, e.g., a few hours (and that different parameters can set the evaluation time to, say, few minutes). One significant disadvantage of the protocol of [3] is that it requires an early *registration* step of the users that by itself requires either the use of certificates or the use of a special hardware. This registration step is later used to control the identity of users who buy tickets during the "critical phase" of the lottery and to make sure that no small group of players "blocks" this critical phase so that other players cannot buy tickets at this time. For this, [3] put a limit on the number of tickets a user can buy (e.g., one). Clearly, it is a desirable property to enable purchases of multiple tickets, as people tend to buy more than one lottery ticket (especially in lotteries where the jackpot is large). We note that in our protocol each user can buy many tickets but is discouraged to buy a really large number of tickets since he will not be able to check whether it holds a winning ticket during the limited time it has (because it needs to evaluate a delaying function for each ticket that it checks).

In Section 3.3, we present our protocol for fair lotteries, which allows each user to buy more than one ticket, while withstanding the attacks described in Section 3.2. We also make use of delaying functions; however, while [3] use delaying functions for *computing the winning ticket*, we use delaying functions in the *winning verification stage*. The transfer of the delay to the verification process enables us to allow users to purchase more than a single ticket. Yet, this transfer should be done in such a manner that will not render the lottery useless. Furthermore, the delaying functions of [3] are required to have a long delay – they may require this period to be measured in hours. In contrast, our delaying functions can be of minutes. We note that setting bounds of security for delaying functions would require a large overhead for the long delaying functions while maintaining a low overhead for the short delaying functions. As we show, our use of delaying functions can substitute that of [3] and still achieve all the needed securities, but if it is desired then they can be combined within the same protocol (i.e., use delays both in the computation of the winning ticket and in the winning verification stage).

Related Work: The idea of using delaying functions is not new and goes back to [4] (where the terminology of "puzzles" is used). Other papers, e.g. [7,1,2], also discuss both the construction of such functions and their applications in various contexts. Syverson ([9]) presents another protocol for fair lotteries, these protocols rely on weak bit commitment. Some of the issues which are orthogonal to the specific design of the lottery which are discussed in [9] can also be added to our design. Yet, the basic protocol and the method by which it achieves its goal varies greatly from the protocol presented in this paper.

Lotteries are useful not only for their own sake but have various applications; e.g., in [6] lotteries are used to design a probabilistic micropayment scheme.

2 Interactive Games

In this section we discuss interactive games. In such a game the casino $\mathcal{C}$ and a user $\mathcal{U}$ participate in a protocol. In our protocol, we need to know very little about the specific game to be played. We do assume however that ℓ, the length of the game (e.g., the number of rounds in a blackjack game), is fixed and known to all; otherwise, fixing ℓ can be added as part of the protocol. Also, all the games are based on randomness; it is common to generate this randomness via a pseudorandom generator. We assume that the algorithm that implements the game based on this randomness (including the pseudorandom generator itself) is publicly known and hence can be tested to meet the published winning probabilities (again, if the algorithm is not known in advance then the protocol below can be slightly modified so that the casino publishes and sign this algorithm as part of the protocol).

The key idea in our protocol is that the casino and the players will jointly choose the randomness (in fact, for efficiency purposes, they choose a seed for the pseudorandom generator). During the game, the user does not know the randomness (this keeps the game fair and fun) but after the game ends it can be verified that the "correct" randomness was used; only then the player makes his payments or claims his earnings. The protocol offers user $\mathcal{U}$ the guarantee that even if the casino $\mathcal{C}$ tries to cheat by biasing the randomness it cannot do so.

As a central building block in our protocol, we use a commitment scheme `commit`. We denote by $\texttt{commit}_{\mathcal{C}}(r, \xi)$ the commitment of $\mathcal{C}$ to a value r using randomness ξ, and we require that it satisfies the following properties: (a) security: given $y = \texttt{commit}_{\mathcal{C}}(r, \xi)$ the value r is semantically secure. (b) decommitment: given r and the randomness ξ used by $\mathcal{C}$ in $\texttt{commit}_{\mathcal{C}}$ it is easy to verify that indeed $y = \texttt{commit}_{\mathcal{C}}(r, \xi)$. (c) collision resistant: for all $r' \neq r$ it is hard (even for $\mathcal{C}$) to find randomness ξ' such that $y = \texttt{commit}_{\mathcal{C}}(r', \xi')$. In addition, we assume a non-forgeable signature scheme. We denote by $\texttt{SIG}_A(m)$ the signature of player A on the message m.
The protocol works as follows:

1. $\mathcal{C}$ picks a random seed r. It sends to $\mathcal{U}$ the value $\texttt{SIG}_{\mathcal{C}}(\texttt{commit}_{\mathcal{C}}(r), id_{game})$, where id_{game} is a unique identifier for the game.
2. User $\mathcal{U}$ chooses at random a value $r_{\mathcal{U}}$ and sends $\mathcal{C}$ the value $\texttt{SIG}_{\mathcal{U}}(r_{\mathcal{U}})$.
3. $\mathcal{C}$ and $\mathcal{U}$ play the game while $\mathcal{C}$ uses $r^{\star} = r \oplus r_{\mathcal{U}}$ as a seed for the pseudorandom generator.
4. When the game is over (but before payment) the casino $\mathcal{C}$ de-commits its r. User $\mathcal{U}$ computes $r^{\star} = r \oplus r_{\mathcal{U}}$ and verifies that all the moves made by $\mathcal{C}$ are consistent with $r^{\star}$ and the algorithm that $\mathcal{C}$ uses. If any of these tests fail (or if $\mathcal{C}$ refuses to de-commit) then the user $\mathcal{U}$ complains against the casino, by presenting the casino's randomness (and signature on this value) and its own random value $r_{\mathcal{U}}$. If the value which the user submits, $r_{\mathcal{U}}$, is not the value which it had given the casino, then the casino presents the user's signature on a different value.

The analysis of the above protocol is simple, given the security of the commitment scheme. The basic idea is that after Step 2 nobody can change its mind regarding its share of $r^{\star}$. Moreover, the choices made by a "bad" casino are independent of those made by the good $\mathcal{U}$. And a faulty user clearly cannot influence the value of $r^{\star}$ as he sends and signs his value $r_{\mathcal{U}}$ after seeing only a commitment to the casino's value. As the commitment is semantically secure it does not expose any information of the value of r.

3 Lotteries

In this section we study fair on-line lotteries. We start by formalizing the properties which we would require from a fair lottery (Section 3.1). We then proceed to describe (in Section 3.2) various attacks which can make the lottery unfair. In Section 3.3 we describe our protocol. We conclude by proving that our protocol satisfies the requirements of a fair lottery system.

3.1 Fair On-line Lotteries

For simplicity, we consider lotteries in which there is a single prize. This prize may be shared among several winners (or there may be no winner at all). Let β be such that the winning probability of a ticket is $\approx 2^{-\beta}$ (e.g., $\beta = 24$ reflects the winning probability in several popular lotteries). The setting is as follows: there is a lottery agency $\mathcal{L}$ and some $k \geq 1$ users $\mathcal{U}_1, \ldots, \mathcal{U}_k$ who participate in the lottery.

The basis of our requirements for a fair lottery are taken from [3] yet we expand them to include requirements for the case in which each participant can purchase more than a single ticket.

Assuming an adversary $\mathcal{A}$ who controls some subset of the users and possibly the lottery agency $\mathcal{L}$ and given β, the distribution parameter, we would require the following.

Uniform distribution: Each ticket has probability of $\approx 2^{-\beta}$ to be chosen regardless of the actions of $\mathcal{A}$.

Independence of ticket values: The tickets purchased by $\mathcal{A}$ are independent of the tickets purchased by the non-corrupted users (e.g., it is impossible for $\mathcal{A}$ to intentionally buy the same tickets as user $\mathcal{U}_i$).

Total purchase: Holding 2^β tickets does not guarantee winning with probability 1. Furthermore, there is a bound on the number of tickets which a user and/or the lottery agency $\mathcal{L}$ could verify as a winning ticket (the desired bound can be tuned by appropriately choosing the parameters for the delaying function).

Fixed set of tickets: Tickets cannot be changed or added after some predefined set time.

We further adopted the definition of [3] for *publicly verifiable* and *closed* lotteries. The first means that the lottery can be verified by all people at the termination of the lottery. The second means that the lottery computation does not require the participation of a trusted third party.

3.2 Possible Attacks

Here are several methods by which a lottery can be made unfair (some of these attacks are easier to protect against than others).

Biasing the winning number: The lottery agency, $\mathcal{L}$, might try to bias the choice of the winning number. In doing so $\mathcal{L}$ might have different goals each of which violates the fairness of the lottery: for example, it may try to pick a winning number that no user has picked, or it may try to pick a winning number different than the number that a specific user $\mathcal{U}_i$ has picked, or it may try to pick a winning number that matches a ticket that it (or a specific user $\mathcal{U}_j$ of its choice) bought.

Duplication: The lottery agency can "buy" (e.g., have a user $\mathcal{U}_j$ act on its behalf) the same ticket(s) as user $\mathcal{U}_i$ does. This means that if $\mathcal{U}_i$ wins the lottery he will not be a single winner, and thus will not be able to claim the full prize.

Buying all the tickets: Given that there is a possibility to purchase multiple tickets the lottery may claim that it has all possible ticket numbers. Thus, it is guaranteed to be a winner (whether or not $\mathcal{L}$ is the only winner depends on the choices made by other users). This mode of attack might be especially attractive for the lottery agency in weeks where the prize is large. It is important to note that $\mathcal{L}$ has an advantage over other users: it does not actually pay for the tickets. Even if the rules of the lottery guarantee that a certain percentage of the income is funneled into the prize it still can be viewed as if the lottery agency can buy the tickets at a discount price.

Forgery: After the winning number is chosen a user (and especially the lottery agency) may try to forge a winning ticket. We note that $\mathcal{L}$ has an extra advantage since it may know the winning number before it is announced. In addition, $\mathcal{L}$ may try to combine this attack with the Biasing attack described above.

3.3 Our Lottery Protocol

The protocol has three phases: Buying Phase, in which each user $\mathcal{U}_i$ who is interested in buying lottery ticket(s) is involved in a protocol with the lottery agency, $\mathcal{L}$; Termination Phase, in which $\mathcal{L}$ computes the winning number; and Claiming Phase, in which each user $\mathcal{U}_i$ can check whether his ticket is a winning one, and if so claim the prize (claiming the prize requires a protocol between each such winning user and $\mathcal{L}$; no interaction is required between non-winning users and $\mathcal{L}$, nor between the users).

Each lottery is characterized by the following information published by $\mathcal{L}$ in advance: t_{start} (the time after which users can start buying lottery tickets), t_{end} (the time after which users cannot buy lottery tickets), w_{val} (a string to be used for determining the winning ticket), and a commitment scheme `commit` and signature scheme as required in the previous section. In addition there is a parameter Δ_1 which is the amount of time that $\mathcal{L}$ has in order to publish the list of tickets bought, that is to commit (by signing) the set of tickets which are part of the current lottery. Similarly Δ_2 determines the length of the period, after the results are published, in which users can claim their prize. Δ_1 should be very short; e.g., a few seconds, and it should specifically be tuned so as to make sure that $\mathcal{L}$ can complete the computation of the hash and signature but not much more than that (for further discussion on how to set the value of Δ_1 see Section 3.4).

For concreteness, we present our protocol using the function `SHA1` (see [8]) as a building block. However, the protocols do not rely on properties of this specific function and can be replaced by other functions. In particular, as our protocols call for a "short-delaying function", i.e. one that takes minutes (rather than hours) to compute, we assume that we can use the following as such a function: Given as input a number α, a block B of size at most $(512-\alpha)$ bits and a target value w_{val} (of length at most 160 bits), find whether there is a string A of length α such that the output of $\texttt{SHA1}(B \circ A)$ starts with a prefix w_{val} (we choose the length of w_{val} to be $\alpha+\beta$ bits so as to make the probability that such A exists be $\approx 2^{-\beta}$). For $\alpha \approx 20$, this computation will take a few minutes on a "reasonable" machine (by simply checking all the 2^{α} possible A's). Furthermore, it is assumed that the security of `SHA1` implies that a few minutes are not only "sufficient" but also "necessary" for this computation; in particular, there is no significant shortcut that circumvents exhaustively trying the 2^{α} `SHA1` evaluations.[2]

Below we describe the three phases of the lottery. We start by describing the Buying Phase that allows users to buy tickets in the period between t_{start} and t_{end}. We use $\mathcal{U}_i$ to denote the i-th user to buy a ticket; note that these users are not necessarily distinct (in other words, one user may buy more than one ticket. As we shall see below, a user should clearly limit the number of tickets he buys to the number of tickets he will be able to check during the Claiming Phase).

[2] Again, we emphasize that this particular implementation is for sake of concreteness only. In particular note that the search for the value of A in this implementation can be parallelized; if we wish to eliminate this option we need to implement the delaying function in an "inherently sequential" way; see [7] for discussion.

In the following description we assume that there is a single execution of the lottery, and thus omit details of the identification number of the lottery.

Buying Phase: (Between time t_{start} and t_{end})

1. User $\mathcal{U}_i$ chooses a value v_i and randomness ξ_i (v_i is the actual ticket value and ξ_i is the randomness used by the commitment; the length of v_i is $512-160-\alpha$ bits[3]). It computes a ticket $T_i = \texttt{commit}(\mathcal{U}_i, v_i, \xi_i)$ and sends T_i (together with the payment for the ticket) to $\mathcal{L}$.
2. The user receives from $\mathcal{L}$ a signature on the ticket T_i, i.e. $\texttt{SIG}_{\mathcal{L}}(T_i, \mathcal{U}_i)$.

Termination Phase:

1. By time $t_{\text{end}} + \Delta_1$, the lottery agency $\mathcal{L}$ publishes the list of all m tickets that were bought $T_1, \ldots, T_m$. It also publishes the hash of this list $r = \texttt{SHA1}(T_1 \circ T_2 \circ \ldots \circ T_m)$ and its signature on r; i.e., $\texttt{SIG}_{\mathcal{L}}(r)$. [4]

Claiming Phase: (Between time $t_{\text{end}} + \Delta_1$ and $t_{\text{end}} + \Delta_2$)

1. Let r be the value computed by $\mathcal{L}$ in the Termination Phase, let v_i be the value chosen by $\mathcal{U}_i$ for his ticket value in the Buying Phase, and let f be an α-bit string referred to as the "free bits". For all values $f \in \{0,1\}^\alpha$ user $\mathcal{U}_i$ computes $\texttt{SHA1}(r \circ v_i \circ f)$; if the output starts with a prefix w_{val} then this ticket is a winner.
   To claim his prize, user $\mathcal{U}_i$ presents to $\mathcal{L}$ the values $\mathcal{U}_i, v_i$ and ξ_i (i.e., $\mathcal{U}_i$ decommits the value given in the Buying Phase), the corresponding free-bits string, f, and the signature generated by $\mathcal{L}$ for this ticket.
2. The lottery verifies that a claimed ticket is in fact a winner, by verifying the signature on the ticket, the commitment value, and that the "free bits" and the ticket compute the needed value.
3. $\mathcal{L}$ publishes the information related to all the winning tickets and announces the prize amount.
4. In addition, user $\mathcal{U}_i$ devotes some time to verify that $\mathcal{L}$ behaves properly. In particular, $\mathcal{U}_i$ should verify that his own ticket(s) appear in the list (otherwise, using the signature(s) it received during the Buying Phase as evidence of misconduct it complains against $\mathcal{L}$). $\mathcal{U}_i$ can also verify the computation of r from the list of tickets. The user also verifies that all the winning tickets announced by $\mathcal{L}$ indeed appear in the list of tickets and are valid winning tickets.

[3] This length was chosen so as to make the input to SHA1, in the Claiming Phase, exactly 512. We note however that one can choose the length of the v_i's to be much smaller and use some fixed string for padding; on the other hand, if the size of the v_i's is too small we will start getting collisions between users.

[4] If one wishes to combine a delaying function in the computation of r it can be done in this step.

If the computation of r is done using a delaying function (e.g., [3] suggest to do so with a function whose evaluation takes a few hours) we may not want the users to repeat the whole computation (in Step 4 above). This can be circumvented by using "witnesses"; that is, mid-point evaluation values some of which can be selected randomly and checked by the users.

The above protocols can be easily extended in various ways. For example, to allow a user to buy more than one ticket in a single Buying Phase; to include the serial number i in the ticket; to distribute the role of the lottery agency $\mathcal{L}$ among several agents $\mathcal{L}_1, \ldots, \mathcal{L}_k$ (to decrease the load in the Buying Phase) etc.

3.4 Security of the Lottery

Theorem 1. *The protocol described above satisfies the requirements of a fair lottery (as appear in Section 3.1).*

Proof. In the following we prove that our protocol satisfies the requirements for a fair lottery. We shall do so by showing how each component in our construction helps in achieving this goal. As in the case of [3], we assume for the security of the scheme that there is a ticket which was purchased by a user who is not controlled by the adversary $\mathcal{A}$ close to the time t_{end}, i.e. at time $t_{\text{end}} - \epsilon$. This limits the time that the adversary has for attacking the lottery.

1. The role of the commitment in the Buying Phase is two folded: on the one hand it disallows the users to change their mind with respect to their values; on the other hand, the fact that $\mathcal{A}$ does not know the value v_i of user $\mathcal{U}_i$ ensures that $\mathcal{L}$ cannot discriminate against $\mathcal{U}_i$ (in other words, if v_i appears in the clear then $\mathcal{L}$ may duplicate this value, or make sure that this value will not be the winning value). Thus, the values of the tickets purchased by $\mathcal{A}$ are independent of the ticket values of the non-corrupted users. Note, that $\mathcal{A}$ could duplicate the value T_i of user $\mathcal{U}_i$, and if this is a winning ticket, $\mathcal{U}_i$ will de-commit the ticket, which would enable $\mathcal{A}$ to de-commit his ticket as well. But the commitment includes in it the user's name (that is, $\mathcal{U}_i$) hence it does not help $\mathcal{A}$ to duplicate T_i.
2. Publishing (and signing) the list of tickets during the Termination Phase guarantees that $\mathcal{L}$ will not be able to manipulate the list of tickets in order to get a value r of its choice. Note that if $\mathcal{L}$ tries to add to the list a ticket which is a winning ticket with respect to a specific r, then due to the way by which r is computed (and the assumed avalanche properties of `SHA1`) this will immediately influence the value of r and so the new added ticket will (most likely) become useless, i.e. not a winning ticket.
3. Assume that $\mathcal{L}$ has computed a pair r, v_i such that v_i is a winning ticket given the value r, then in order to force the lottery to have v_i as a winner it must have r as the randomness. It is assumed due to the collision-resistant property of SHA-1 that given a value r it is hard to find a value x such that $\texttt{SHA1}(x) = r$. Note, that in this specific case the problem is even harder as x

must include in it all the tickets which were actually purchased by legitimate users.

4. Assume that $\mathcal{L}$ has a set of tickets and it wishes to find an r which would make one of the tickets a winner. The time which the lottery has to try to generate the needed r is $\epsilon + \Delta_1$. Thus it chooses some r at random, with the limitation that it is generated at least from all the legally purchased tickets, and for a given ticket v_i which it holds it needs to determine whether they are a winning pair. This computation is done using the delaying function. The value Δ_1 will be set so that it is much smaller than a single computation for determining whether v_i and r are a winning pair (in fixing the value for Δ_1 we should also make a more precise assumption regarding ϵ).
5. The delay in the Claiming Phase puts a limit on the number of tickets a single user can buy (i.e., the number of tickets it will be able to check in time $\Delta_2 - \Delta_1$ which would bring him to the cut-off time of the Claiming Phase). This delay also protects against an attempt of $\mathcal{L}$ to buy all the tickets (or even just "too many" of them) – $\mathcal{L}$ will not have enough time to check which of its tickets is a winner. Also note that the method of determining the winning number disallows systematically buying all possible tickets. Thus, a much larger number of tickets, as determined by "coupon collector" bounds, are needed to cover all the possible values.[5] The issue of "blocking" the lottery, i.e. preventing users from purchasing tickets, is outside the scope of this paper and needs to be dealt with by other means, e.g. user's complaining to an appropriate authority.

Acknowledgment

We thank Hugo Krawczyk for philosophical discussions on the issue of fair lotteries.

References

1. C. Dwork, and M. Naor, "Pricing via Processing or Combating Junk Mail", Proc. of *Crypto'92*.
2. M. K. Franklin, and D. Malkhi, "Auditable metering with lightweight security", Proc. of *Financial Cryptography*, Springer-Verlag LNCS Vol. 1318, pp. 151-160, 1997.
3. D. M. Goldschlag, and S. G. Stubblebine, "Publicly Verifiable Lotteries: Applications of Delaying Functions", Proc. of *Financial Cryptography*, Springer-Verlag LNCS, 1998.
4. R. C. Merkle, "Secure Communication over Insecure Channels", *CACM*, Vol. 21, pp. 294-299, 1978.
5. R. Motwani, and P. Raghavan, "Randomized Algorithms", Cambridge University Press, 1995.

[5] These bounds state, for example, that the expected number of tickets needed in order to have all the N possible tickets is essentially $N \ln N$ (see, e.g., [5]).

6. R. L. Rivest, "Electronic Lottery Tickets as Micropayments", Proc. of *Financial Cryptography*, Springer-Verlag LNCS Vol. 1318, pp. 307-314, 1997.
7. R. L. Rivest, A. Shamir, and D. A. Wagner. "Time-lock Puzzles and Timed-Release Crypto", Technical Memo MIT/LCS/TR-684, February 1996.
8. National Institute for Standards and Technology. Secure Hash Standard, April 17 1995.
9. P. Syverson, "Weakly Secret Bit Commitment: Applications to Lotteries and Fair Exchange", 11th IEEE Computer Security Foundations Workshop, 1998.