Publicly Verifiable Lotteries: Applications of Delaying Functions

David M. Goldschlag* Stuart G. Stubblebine**

Abstract. This paper uses *delaying functions*, functions that require significant calculation time, in the development of a one-pass lottery scheme in which winners are chosen fairly using only internal information. Since all this information may be published (even before the lottery closes), anyone can do the calculation and therefore verify that the winner was chosen correctly. Since the calculation uses a delaying function, ticket purchasers cannot take advantage of this information. Fraud on the part of the lottery agent is detectable and no single ticket purchaser needs to be trusted. Coalitions of purchasers attempting to control the winning ticket calculation are either unsuccessful or are detected. The scheme can be made resistant to coalitions of arbitrary size. Since we assume that coalitions of larger size are harder to assemble, the probability that the lottery is fair can be made arbitrarily high. The paper defines delaying functions and contrasts them with pricing functions [8] and time-lock puzzles [15].

1 Introduction

In a typical lottery, one or more winners are chosen using some process that is trusted to give each purchased ticket an equal chance of winning: choosing balls from an urn, for example. The process may be executed or monitored by an outside auditor. Unfortunately, because the process is random, it is not repeatable, and ticket purchasers must trust both the process and the auditing organization.

In this paper, we propose a scheme for fairly selecting lottery winners using information internal to the lottery (e.g., numbers on tickets). The winning number calculation is therefore repeatable. We make the calculation verifiable by obliging the lottery agent to publish all internal information. We put the lottery agent on a level playing field with his customers by obliging him to publish the internal information as he accumulates it. We prevent anyone from taking advantage of this internal information by using *delaying functions* as part of the winning number calculation. These functions require significant computational resources, which are unlikely to be available before the lottery closes.

The winning number calculation produces a random (i.e., fair) result if at least one of the tickets used in the calculation is random (i.e., not under the

---

* Divx, 570 Herndon Parkway, Herndon, VA 20170, USA, +1 703-708-4028 (voice), +1 703-708-4088 (fax), david.goldschlag@divx.com

** AT&T Labs–Research, Rm B235, 180 Park Avenue, Florham Park, NJ 07932, USA. http://www.research.att.com/˜stubblebine

control of a colluder). Therefore, the lottery could be controlled if all tickets are purchased by colluders. Since we assume that larger coalitions are harder to assemble, the lottery scheme is designed to detect coalitions of a pre-specified size and to be resistant to attack by smaller coalitions. (If a coalition is detected, the lottery may be extended.) By choosing the hardness of the delaying function and the detectable coalition size appropriately, the probability of a fair lottery can be made arbitrarily high. The lottery is one-pass, except (if tickets are anonymous) the winner may need to return to claim her winnings.

This paper uses three concepts in its lottery design:

1. Delaying functions prevent computationally bounded adversaries from cheating.
2. Trust is distributed by giving everyone equal access to sensitive information. Where one party has a role that makes him more trusted than another (i.e., the lottery agent), that party incriminates himself when cheating. Furthermore, it is easy for individual purchasers to provide the evidence.
3. We detect, instead of prevent, denial of service attacks.

Prior research [10] indicates that it is information theoretically impossible to construct a Boolean function that cannot be controlled by sufficiently large sized subsets of its inputs. In the absence of other assumptions, this work would preclude our result. However, that work does not place computational constraints on the coalition controlling the inputs.

Other work has used calculation time as a barrier [8, 9, 15, 12]. But none of the solutions have all the properties that we require here: Unlike time-lock puzzles [15, 12], the solution should not be known in advance to anyone, not even the puzzlemaker. And, although easy verification of the solution may be convenient, it is not a requirement here. Unlike pricing functions [8, 9], the cost of a security breach is high.

This paper is organized as follows. In section 2, we present definitions. Section 3 describes our solution including lottery registration, ticket purchase, critical purchase phase, and winning entry calculation. In section 5, we present related work. Section 6 presents some concluding remarks.

2   Definitions

Lottery terms and properties. We now present some terminology for describing lotteries. Our cast of characters include the *lottery agent* who runs the lottery, and a *client agent* who represents the *customer*. The client agent and lottery agent interact to generate a *lottery ticket*. Upon completion of this interaction a lottery ticket is said to be purchased. A lottery ticket is a record containing at least a *seed* and a *winning number parameter* of fixed size. Those two parameters may be the same, or be implicit (e.g., the winning number parameter may be a function of the ticket). For the purposes of this paper, we assume that a ticket's winning number parameter is the same as its seed.

The run of a lottery ends at a scheduled *closing time*, t. A distinguished interval called the *critical purchase phase* is represented as a pre-defined period, p. During the critical purchase phase, the required number of distinct client agents purchasing tickets is at least n. The number of lottery tickets sold is represented as L. The sum of what customers pay for tickets is called the *lottery revenue*. The fraction of the lottery revenue that is distributed to winners (according to pre-defined rules of the lottery) is called the *winnings pool*. The time to compute the winning ticket(s) is called the *calculation time*.

We now define properties of lotteries. First, the probability that a lottery is fair is represented using the notation P_f. The fairness of the lottery depends upon three factors: that the calculation used to compute the winning ticket gives all purchased tickets an equal chance of being selected; that the winning number calculation is highly unlikely to be computable before the lottery closes; and that at least one ticket purchased during the critical purchase phase is sold to a non-coalition member. Second, if anyone can calculate the winning ticket based on the parameters of purchased tickets, and the integrity of the winnings pool can be verified, we say the lottery is *publicly verifiable*. Third, if the calculation of the winning number depends only upon purchased tickets and not upon some extra information (such as a "trusted" third party selecting a random number), then we say the lottery is *closed*.

Delaying functions. We borrow complexity terminology from [8]. However, our definitions are not identical. We choose a calculation function that is *moderately hard* to compute, as opposed to *easy* or cryptographically *hard*. Depending on the lottery requirements, this may mean that the calculation time takes several hours, using the fastest known implementation. The critical purchase phase p is set at some fraction of that time, to place an additional measure of safety that the calculation could not be completed while the lottery is running.

A function f is a *delaying function* if:

1. f is moderately hard to compute. Given a minimum operation time p and an interval following p i.e. $[p, q]$, the probability that the function is computable on average is less than or equal to ϵ before the interval and the probability increases monotonically from $[\epsilon, 1]$ over that interval where ϵ can be made arbitrarily small.

   The function f is computationally secure. However, the attack complexity (the dominant of the data, storage, and processing complexities) that we must resist is much lower than that expected in a cryptographic function (i.e., hours of significant effort instead of years).

2. f preserves the information of its inputs. That is, given $Y = f(X)$, where Y and X are random variables representing the range and domain of the function f, $H(X) \simeq H(Y)$ (where $H(X)$ is the entropy of X).

The second requirement may be made clear by example. In our lottery scenario, the delaying function maps random numbers to random numbers and these are mapped to the winning ticket. We must ensure that the lottery is fair.

We caution the reader that delaying functions suffer from *practical* problems common to other cryptographic mechanisms: it is difficult to be sure that the problem can't be shortcutted (with a trap door or another mechanism) or parallelized. It is also difficult to count the number of sequential operations required to solve the problem–for example, people have discovered increasingly efficient ways to implement DES over the years. Finally, estimates of the cost of executing those operations are unlikely to be precise. For our purposes, however, we only require a conservative lower bound on the actual delay.

Syntax. We use "floor brackets" to indicate message authentication and curly braces to indicate message confidentiality. Thus, '$\lfloor X \rfloor_K$' might refer to data X signed with key K or a keyed one-way hash of X using K. '$\{X\}_K$' refers to X encrypted with key K. For our purposes, both of these are used to refer to mechanisms that also provide integrity within the indicated scope.

3   Approach

3.1   Problem and Overview

We place the following three requirements on our lottery design:

R1. The probability that the lottery is fair, P_f, can be made arbitrarily high:
1.1 The winning number calculation gives each purchased ticket an equal chance of being selected.
1.2 The calculation is unlikely to be computable in less than p time.
1.3 The calculation is resiliant to failures: It is random if even one argument is random, and the chance of one random argument is high.
R2. The lottery is publicly verifiable.
R3. The lottery is closed.

We are concerned with the type of lottery where if the entire pool is not distributed to winners, the balance rolls over into subsequent lotteries.

A solution must be secure against an adversary that can purchase tickets, selectively block communications, and control the order of purchased tickets. The lottery agent can be an adversary too. We make a simplifying assumption that the probability of a successful attack on cryptographic mechanisms is negligible.

Our approach divides the lottery into four phases. These phases are *registration, purchase, critical purchase,* and *winner calculation.* The purchase phase contains the entire critical purchase phase. The critical purchase phase defines the distinguished interval p. Prior to the purchase phase, the lottery agent commits to when the critical purchase phase begins.

We now present what happens in each of the lottery phases.

3.2   Registration

In order to detect whether a coalition of a particular size controls the lottery we must be able to count the number of different client agents participating in the lottery. How can we ensure that tickets are from distinct sources? The goal of registration is to enable this identification. This requires mapping between an individual and his client agent. The client agents may use certificates or hardware devices. The integrity of the mapping is maintained in two steps:

- It must be difficult for an individual to obtain more than a small number of certificates or hardware devices.
- Only the owner can prove that he is authorized to use the certificate or box.

We could use certificates issued by certificate authorities that can be trusted to issue only a single certificate to an individual. The individual could prove that he is authorized to use the certificate by signing some challenge with his private key. Anonymous purchases can be done in two ways: Certificate authorities could issue a blinded certificate along with an unblinded certificate. The lottery service may wish to issue its own certificates when presented with third party certificates to simplify ticket purchase. We refer to a client C's private key as K_C and to his certificate as $Cert_C$. This certificate includes the client's public key K_C^{-1}. Similarly, the lottery agent's private key is K_L^{-1}.

3.3   Purchase

During lottery transactions, tickets are only sold to registered client agents. Those are the only clients who know the private signature keys corresponding to the certified public keys. Ticket purchases by the same client agent can be correlated since the certificate is presented during each ticket purchase transaction. Ticket correlation is necessary, since we need to identify n distinct client agents. (This is discussed in further detail in sections 3.4).

Ticket purchase consists of the steps:

$$\text{Message 1} \quad C \to L: \quad \lfloor Seed \rfloor_{K_C}, Cert_C, Payment^3$$
$$\text{Message 2} \quad L \to C: \quad \lfloor \lfloor Seed \rfloor_{K_C}, seq_L, time_L, Cert_C \rfloor_{K_L}$$

In Message 1 the client agent requests a ticket by providing a signed seed, and possibly payment. In Message 2 the lottery agent returns a ticket, in which the lottery agent signs the already signed seed along with other fields: sequence number, timestamp, and purchaser's certificate. The client agent then verifies the ticket. The ticket's construction enables the proper purchaser to claim his winnings since the winner is the only one who can sign a challenge that can

---

[3] The payment must be structured so it cannot be applied with another seed. This level of detail is highly dependent on the payment mechanism and is not included in this description.

be verified with the same key as the winning ticket (e.g., K_C^{-1}). The lottery agent publishes the ticket. Publication enables observers to count distinct ticket purchasers and to calculate the winning ticket. Publication also allows individual ticket purchasers to ensure that their tickets are included in the calculation.

In Message 1, the space of the seed parameter must be at least as large as the number of tickets that can be sold. The client agent randomly generates the seed according to a uniform distribution over this range. Standardized implementations of the client agent can help insure that the client agent correctly generates the seed using available techniques for random number generation [1]. Clients not obeying the protocol specification may be defined to belong to the coalition attempting to control the lottery. (If the set is thus expanded, the number of distinct client agents required may need to increase to achieve the same probability of fairness.)

Message 2 represents the lottery ticket. Lottery tickets contain sequence number and timestamp fields. The goal is for the lottery agent to include information in tickets to incriminate himself if he cheats, while making the collection of this evidence easy for individual purchasers.

Sequence numbers allow the collection of small amounts of inconsistent information (i.e., two tickets with the same sequence number.) One could imagine that the lottery agent publishes both in the database, but not at the same time (in order to make the ticket appear yet make the winnings pool smaller). If sequence numbers are reused, audit queries from client agents searching for a single sequence number can eventually find two tickets with the same sequence number. In the analogous attack on tickets not containing sequence numbers, the lottery agent publishes only a subset of the database at any time. The evidence required to prove the absence of a ticket is to take a snapshot of the entire database without the ticket. (The lottery design may also require that sequence numbers be consecutive.)

Alternatively, tickets may contain signatures of previous tickets. This approach has similar virtues and limitations of the other approaches with the added feature that the lottery agent would not need to create a signature over the entire database after each ticket purchase. However, checking the integrity of the database would require numerous signature verifications. Of course, combinations of these approaches may prove suitable in practice.

The lottery agent includes the current time in message two of the signed ticket. This time is meant to be fairly accurate, and serves to identify which tickets are used in the winning calculation. The client agent can check this time locally, and if the lottery agent's clock is very fast (i.e., the lottery agent is trying to start the critical purchase phase early), then the client agent can immediately send the ticket to law enforcement, who will stamp it promptly. (For this purpose, it is reasonable to assume some form of communication channel, e.g., telephone, is available between a client agent and law enforcement.) Thus, the lottery agent has incriminated himself. The opposite attack, where the lottery agent slows the clock prior to the critical purchase phase, does not help the attacker, since shortening the critical purchase phase makes it even more unlikely that the

attacker can solve the delaying function in time.

At any time after the ticket purchase, the client may check to see that his purchased tickets are published properly. If they are not, the client can complain to lawful authorities and provide proof (the ticket or certificate or response to a query). Anyone can verify the number of distinct ticket purchasers.

3.4 Critical Purchase

Before the lottery starts the lottery agent commits to the time of the beginning of the critical purchase phase. The critical purchase phase is entirely within the purchase phase. Consequently, all steps of the purchase phase apply to the critical purchase phase. The critical purchase phase is an interval of size p. The critical purchase phase and the purchase phase end at the close of the lottery, t. We make the following assumptions:

A1. At least n distinct certificates appear in the set of purchased tickets during the critical purchase phase.

A2. The probability that n or more distinct client agents collude can be made arbitrarily small by increasing n.

The choice of n in assumption A2 depends in part on the likelihood that individuals have multiple certificates, and the likelihood of a client agent being a colluder. As discussed later, the values of n and p are chosen to make the probability of a fair lottery, (i.e., P_f) acceptably high.

3.5 Winning Entry Calculation

The winning entry calculation must select among all purchased tickets with equal probability. It operates under the minimal assumption that only a single seed parameter among all tickets sold during the critical purchase phase was chosen randomly. The timestamp within the ticket indicates which tickets fall within the critical period.

The winning entry calculation may be structured into the following steps:

1. Hash computation. Compute $h(s_1, s_2, ...)$ where $s_1, s_2, ...$ represents the concatenation of all seed parameters of tickets purchased during the critical purchase phase p. We require that $h()$ be preimage resistant and non-correlating [13, 19].
2. Delay calculation. Input the result of step 1 to a delaying function (as defined in section 2).

If the range of step 2 does not map directly to a winning ticket (e.g., sequence number) then the following step may be needed:

3. Winner computation. Map the result of step 2 to one or more purchased tickets.

The hashing of step 1 may be necessary if the delaying function does not take input of arbitrary size and the delaying function does not have the non-correlating properties of "distributing" localized randomness to all output bits.

The delay calculation is achieved through a moderately hard calculation that must also preserve the randomness in the input. We may choose the parameters of a delaying function for the calculation to take several hours, using the best known implementation. By choosing p to be some conservative fraction of that time, we can make it very likely that the calculation cannot be completed before the lottery ends. A consequence of our winning entry construction is:

A3. The probability that the winning calculation can be made within p time can be made arbitrarily small.

Where L is the number of lottery tickets sold, an implementation of these three steps may be:

1. Concatenate in some pre-defined order the seed parameters purchased in the critical purchase phase of the lottery. Hash the resulting string.
2. Use bits from the resulting hash as the key for a cipher with a very long period. Run the cipher in output feedback mode (OFB) to generate some pre-defined number of bits. Save the last $\lceil \log L \rceil$ bits.
3. Choose winning entries by computing the Hamming distance between the result of step 2 and the winning number parameter (expressed in binary) in the tickets. (The Hamming distance $d(x, y)$ from x to y is the number of positions in which the two strings differ [16].)

The use of a cipher in OFB mode may be a good delaying function. The cipher should have a large period, large linear complexity, and good statistical properties. Such a construction should be hard to short-circuit, because it is hard to predict how the choice of initialization vector indexes into the keystream. Also, the implementation of many ciphers have been carefully studied for minimizing the number of operations required. By conservatively estimating the cost of the dominating operations (bit permutations) one can predict how much delay a certain keystream length requires.

Notice that this implementation could choose several winners. The lottery policy may allocate the winnings using a tiered approach among the closest Hamming distances.

3.6   Denial of Service

Our lottery is subject to a denial of service attack. If fewer than n distinct sources are detected in the p interval, the winning calculation is considered invalid. A reasonable lottery policy may be to extend the lottery.

Measures to prevent, rather than detect, denial of service attacks are beyond the scope of this paper. But it is easy to imagine engineering approaches that complicate denial of service attacks, if the lottery agent does not participate in the coalition. For example, if sufficient independent network connections

connect the lottery agent to his possible customers, blocking communication entirely becomes more difficult. Of course, the lottery agent could be one of the colluders. Unmonitored, he can then deny service. This too may be detectable by monitoring the arrival of tickets during period p.

4 Evaluation

Our lottery operates in the following way: Assume a set of games that is too large to pre-compute. Each game produces a random result, and playing any game takes longer than some period of time, p.

The seeds in tickets purchased during the final p interval in the lottery together define the selected game. The game is played, and the result defines the winning ticket(s).

To fix the lottery, colluders must choose their seeds to select a game with favorable results. Since, any play of any game will extend beyond the close of the lottery, colluders must control all the final tickets sold (and precompute that set). But we assume one random ticket among the set of final tickets sold (contradiction).

In this analogy, the game is the delaying function. It is interesting to note that we could shift the delaying function to the function that amalgamates the seeds–that is, selecting the game could be the expensive operation, instead of playing the game. The resulting lottery would be fair too.

5 Related Work

We use a function of moderate complexity as the delaying function when computing the winner. To increase security, we minimize the requirements placed on that function (i.e., only complexity and randomness). In contrast, for the different problem of controlling access to a resource, Dwork and Naor present a technique requiring a user to compute a moderately hard, but not intractable (cryptographically hard) function which they call a pricing function [8]. They present solutions based on extracting square roots modulo a prime, the Fiat-Shamir signature scheme, and the Ong-Schnorr-Shamir (cracked) signature scheme. These solutions depend on the difficulty of extracting square roots mod p (no known method requires fewer than about log p multiplications), factoring large numbers, and an algorithm by Pollard for breaking the signature scheme based on quadratic equations modulo a composite, respectively. In a similar fashion, Franklin and Malkhi [9] use repeated hashing as evidence that a certain amount of time has elapsed.

These papers' motivating applications, preventing and/or detecting junk e-mail, and metering web clicks for advertising revenue, have two important properties that are not present in lotteries. The first is that, unlike lotteries, the cost of a security breach is low: some junk mail may get through or an advertiser may be changed for a bogus click. In contrast, the probability that a lottery may be

fixed must be insignificant. The second difference is that both the junk mail and Web click metering solutions require asymmetric functions: the calculation must be moderately hard, but verification must be cheap. This is because the moderately hard calculation is used to slow down attackers, but the recipient's filtering process must be fast. If the calculation is not done correctly, the recipient ignores the mail. If the verification was as hard as the calculation, this approach would be ineffective: for example, the denial of service attack that junk mail presents would be transformed into one that consumes cycles.

Can we use pricing functions as our delaying functions? If the pricing function must be built using a shortcut, and one entity knows the shortcut, then one entity can break the scheme. The security of the lottery would then rely both upon the hardness of the non-shortcutted solution, and the confidentiality of the shortcut. Because we want to decentralize trust in the lottery, and do not want to rely upon the confidentiality of the shortcut, such pricing functions would not be suitable.

Pricing functions must allow for easy verifications of solutions that are hard to calculate. We could use this functionality in the lottery solution, but it may complicate the lottery design. For example, if a single ticket's seed parameter was the only input to the winning calculation and the ticket was the winner if its winning number parameter was the result, then each ticket purchaser would face a quandary: He could verify whether his proposed seed and winning numbers parameters would be winners before purchasing a ticket! Since nearly all combinations would loose, purchasers would never purchase tickets.

We have two options: We can either simplify the selection of delaying functions by requiring that they only satisfy the moderately hard to compute property, or we can complicate their selection, but perhaps simplify the lottery design, by requiring them also to be hard to verify. As our lottery design is resistant to the dilemma proposed above, we do not place the hard to verify requirement on our delaying functions.

The class of delaying functions may include the functions of [8, 9]. Since delaying functions do not have the easy to verify or shortcut requirement, they can be easier to construct, simply by choosing functions that have a high operation count and cannot be parallelized.

Rivest et. al. [15] propose time-lock puzzles where a puzzlemaker can create a puzzle that requires a well defined amount of computation time to solve. The proposal is related to [12] but is not parallelizable. Unlike delaying functions, the solution is known to the puzzlemaker, but is then discarded. So the puzzlemaker must be trusted not to cheat. The related work is summarized in Table 1. (A blank space in the table means that the property on the left is not a requirement for the function at the top of the column.)

Beth and Desmedt [3] present a time based solution to the chess grandmaster problem. Their solution involves delay, not expensive computation. They solve the problem of how to prevent a man in the middle from playing two grandmasters against one another, and claiming a significant win. They reduce the problem to one of authentication, and show that if the game setup protocol re-

	Delaying Functions	Time-lock Puzzles	Pricing Functions
Preserve Entropy	*Requirement*		
Easy to Verify		*Requirement*	*Requirement*
Solution Known a-Priori	*Negative Requirement*	*Requirement*	*Requirement* (with shortcut)

Table 1. Required properties of the related functions.

quires each party to commit to an exact delay that will precede their moves, one of the legitimate players will discover the man in the middle. They assume that communication takes some time, and do not require parties to otherwise identify themselves. (This is in contrast to PGP-Phone [20], which requires parties to read the negotiated key–the right sounding voice reading shared keying material is hard to forge in real-time.)

Finally, we briefly touch on other work. Cai et. al use moderately hard functions in the design of uncheatable benchmarks [6]. Benchmarks use problems that take significant computation time. However, if the input data is known a-priori, machine architectures can be optimized for those inputs. A solution is to select inputs randomly for each test. However, this makes independent verification of the result difficult (perhaps the machine computed the wrong result quickly!). Cai et. al. propose functions that have shortcuts that make them easy to verify, and functions that are easier to verify than to compute. Rivest [14] defines several types of electronic lotteries. There has been much work on distributed coin flipping [2, 4, 5, 11]. The protocols are expensive for each participant and are multi-pass. Also, Kahn, Kalai, and Linial [10] prove that it is information theoretically impossible to design functions that cannot be controlled by some strict subset of their inputs. Finally, we note if registration certificates are blinded for anonymity, care must be taken that the communications channel not identify the customer [18].

6 Conclusion

This paper presented a publicly verifiable lottery scheme that can be made fair with arbitrarily high probability. The scheme relies on two assumptions: the existence of delaying functions that are unlikely to be controllable by colluders with limited computational resources and that preserve the entropy in their inputs; and, the likelihood that large coalitions are more difficult to assemble than smaller ones. A candidate delaying function may use a cryptographic cipher with a very long period to generate a sufficient amount of keystream.

Delaying functions are related to pricing functions, time-lock puzzles, and other work that uses calculation time as a barrier [8, 9, 15, 12]. However, none of the previously proposed functions possess all the required properties of delaying

functions: the function must preserve the entropy on its inputs, and the solution must not be known in advance to anyone, not even the party controlling the game.

Publicly verifiable lotteries decentralize the auditing of trusted function by enabling many individuals to validate the selected winner(s). To detect lottery fraud, individual customers can independently verify that their tickets were counted, anyone can compute the size of the winnings pool, and anyone with sufficient computational resources can calculate the winning entry. The use of time as a barrier may enable other interesting applications as well.

7   Acknowledgments

References

1. W. Aiello, S. Venkatesan, and R. Venkatesan. Design of practical and provably good random number generators. Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, NY, Jan. 1995.
2. M. Bellare, J. Garay, and T. Rabin. Distributed Pseudo-Random Bit Generators - A New Way to Speed-Up Shared Coin Tossing, PODC'96, ACM, 1996.
3. T. Beth and Y. Desmedt. Identification tokens-or: solving the chess grandmaster problem. Advances in Cryptology - CRYPTO '90 Proceedings, pp. 169-176.
4. M. Blum. Coin flipping by telephone-a protocol for solving impossible problems. Spring COMPCON 82, IEEE, 1982, pp. 133-137.
5. A. Broder. A provably secure polynomial approximation scheme for the distributed lottery problem. Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing, 1985, pp. 136-148.
6. J. Y. Cai, A. Nerurkar, M. Y. Wu. The Design of Uncheatable Benchmarks Using Complexity Theory, technical report 97-10, Department of Computer Science, SUNY Buffalo, 1997.
7. D. Chaum, "Security without Identification: Transaction Systems to Make Big Brother Obsolete", CACM (28,10), October 1985, pp. 1030–1044.
8. C. Dwork, and M. Naor. Pricing via processing or combating junk mail. Weizmann Technical Report CS95-20, 1995 (Also preliminary version in CRYPTO '92, pp. 139-147).
9. M. K. Franklin and D. Malkhi. Auditable Metering with Lightweight Security, Financial Cryptography 1997, LNCS Vol. 1318, Springer-Verlag, 1997.
10. J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions. 29th Annual Symposium on Foundations of Computer Science, IEEE Washington, DC, USA, 1988, pp. 68-80.
11. E. Kushilevitz, Y. Mansour, and M. Rabin, On Lotteries with Unique Winners, SIAM Journal on Discrete Mathematics, vol. 8, No. 1, pp. 93-98, February, 1995.

12. R. Merkle. Secure Communications Over Insecure Channels. Communications of the ACM, vol. 21, no. 4, April, 1978, pp. 284-299.
13. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.
14. R. Rivest. Electronic Lottery Tickets as Micropayments, Financial Cryptography 1997.
15. R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release Crypto. Unpublished manuscript, February, 1996. See http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps.
16. S. Roman. Coding and Information Theory. Graduate texts in Mathematics number 134, Springer-Verlag, 1992.
17. P. Syverson, S. Stubblebine, and D. Goldschlag. Unlinkable Serial Transactions, Financial Cryptography 1997, LNCS Vol. 1318, Springer-Verlag, 1997.
18. P. Syverson, D. Goldschlag, and M. Reed. Anonymous Connections and Onion Routing, *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, May 1997.
19. A. Webster and S. Tavares. One the design of S-boxes, Advances in Cryptology - Crypto 85 (LNCS 218), pp. 523-534, 1986.
20. P. R. Zimmerman. PGPfone Owner's Manual, Version 1.0 beta 7, July 8, 1996, p. 33.