Int. J. Inf. Secur. (2010) 9:33–50
DOI 10.1007/s10207-009-0094-1

REGULAR CONTRIBUTION

Temporarily hidden bit commitment and lottery applications

David M. Goldschlag · Stuart G. Stubblebine · Paul F. Syverson

Published online: 19 November 2009

Abstract This paper describes various types of commitment functions that maintain a secret for a predictable time delay or until a moderate and predictable amount of computation has occurred. The properties we set out for such functions are based on their usefulness for various applications, such as publicly verifiable lotteries, rather than for cryptologic investigation of the functions. In these lotteries, 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 or similar function, neither ticket purchasers nor the lottery organizer can take advantage of this information. We describe several such lotteries and the security requirements they satisfy, assuming that functions with the properties we state are used.

Keywords Timed commitment · Predictability · Lottery design · Public verifiability

D. M. Goldschlag
Trust Digital, Inc., McLean, VA, USA
e-mail: david@goldschlag.com

S. G. Stubblebine
Stubblebine Consulting, LLC, PO Box 212,
Madison, NJ 07940-0212, USA
e-mail: stuart@stubblebine.com

P. F. Syverson (✉)
Center for High Assurance Computer Systems,
Naval Research Laboratory, Washington, DC 20375, USA
e-mail: syverson@itd.nrl.navy.mil

1 Introduction

In this paper, we describe various mechanisms to commit to a value in such a way that anyone can confirm that commitment after a predictable amount of computation. After exploring various functions of this type and their properties, we describe how to use them 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 describe two mechanisms to control discovery of a committed value. *Delaying functions* take a predictable amount of computation to calculate an output from a given input. By setting the amount of computation, the amount of delay can be controlled. In secret bit commitment, a value is committed that cannot be discovered until the committer, or some other party(s), reveals the value or some other secret or performs a private computation. In *weakly secret bit commitment* (WSBC), no outside cooperation is needed to discover the hidden value; however, a predictable amount of computation is needed to compute the value without assistance. Intuitively, delaying functions calculate some previously unknown value from a given input, while WSBC is naturally thought of as hiding a value known to the committer. In fact, it is possible but not necessary that the committed value be known in advance. Delaying functions may be characterized as inverses of a certain class of WSBC functions. The scope of our descriptions is to adequately specify requirements for our intended applications, such as lotteries. We do not attempt to prove cryptologic results about such functions in general or to make claims about any particular instances of such functions. Such study is left for future work.

The lotteries presented in this paper, all use these kinds of functions to allow everyone to know the values that determine the outcome of the lottery, but the functions prevent anyone from using this knowledge to alter that outcome in a controlled and predictable manner—as long as at least one bit of input is unpredictable by them. Some lotteries make it difficult for anyone to control the entire input by providing mechanisms that require very large coalitions of players in order to do so. Other lotteries are designed so that anyone completely controlling the input gains no monetary advantage thereby. All the lotteries are one-pass; players need to make only a single purchase transaction for the lottery result to be complete. Winners may need to return to collect winnings unless delivery of any winnings is arranged in the ticket purchase. One design is much more efficient if most players cooperate in a second pass, but the outcome can eventually be determined even if none do.

This paper is organized as follows. In Sect. 2, we define useful properties of commitment and delaying functions. In Sect. 3, we give the basic requirements and assumptions of a high-level lottery design that is publicly verifiable. In Sect. 4, we describe a lottery where tickets have a fixed expected value. This prevents any size coalition from controlling the lottery outcome. If one can bound the coalition size of a lottery-controlling adversary, then it is possible to have a lottery with a growing jackpot, where funds from one lottery run can fairly roll over into the next. Such a lottery is described in Sect. 5. In Sect. 6, we describe a lottery using delays associated with individual ticket purchases rather than on the collected input from all purchased tickets. In Sect. 7, we present related work. Section 8 presents some concluding remarks.

2 Commitment and delaying functions

Bit commitment is a concept that has been around in cryptography for some time. (Cf. [19] for references.) The basic idea behind *classical secret bit commitment* (CSBC) is to allow a principal Alice to commit to a value so that another principal Bob will not know what the value is until Alice reveals it. At the same time, once Alice has committed, she cannot change the value without Bob detecting it. A simple example is as follows.

Message 1	$A \to B:$	$\{b\}_K$
Message 2	$A \to B:$	K

Alice uses key K to encrypt the value b to which she wishes to commit (possibly a single bit) and sends it to Bob. At this point, she is committed to b. Once Bob sees K, he will be able to confirm that she chose b. The second message would typically be sent much later, when the committed bit is to be revealed. (For example, in a game of chance, between these two messages Bob might send Alice a guess of the value to which she had committed, the winner being determined by whether or not he had guessed rightly.) The key and the encryption algorithm are obviously important here. A simple XOR encryption would allow Alice to change her commitment by changing keys. We now describe properties that a secret bit commitment scheme should satisfy.

2.1 Secret Bit Commitment

As the name implies the two main properties of a secret bit commitment scheme are that the bit remain secret and that it be committed. We can describe desired properties in terms of a secret bit commitment function f. We will be discussing weaker levels of secrecy later, but for standard secret bit commitment, we require

– **preimage resistance:** For any prespecified value y of f, it is *computationally infeasible* to compute any x such that $y = f(x)$

This is the same as a standard desired property of hash functions taken from [14] (*q.v.* for more properties and discussion). What 'computationally infeasible' means will depend on intended application; however, there has been lots of research into quantifying this for various cryptographic functions and algorithms.

Like secrecy, commitment also admits of degrees. As we noted before, a simple XOR encryption-based CSBC function implies no commitment at all. At the strongest, we can have

– **injectiveness (one-one):** For any given x and y such that $y = f(x)$, the inverse image $f^{-1}(y) = \{x\}$ (the singleton set containing x).

This is stronger than what can be offered by most standard methods of bit commitment. It is not surprising that CSBC based on well-known hash functions cannot meet this requirement. However, CSBC based on standard symmetric encryption, e.g., block ciphers, cannot meet it either. This is because such CSBC functions are parametric in the keys chosen. And, even though symmetric encryption is, by design, injective, this is not so once we allow the key to vary along with the input.

In practice, we will always be concerned with finite spaces. If there are probability measures attached to those spaces, then the function will be injective iff it is entropy preserving between the spaces.

At a slightly weaker level of commitment, we can have

– **collision resistance:** It is computationally infeasible to find two values $x' \neq x$ such that $f(x) = f(x')$

This is another standard property of hash functions. A commitment property that is weaker still is

– **2nd preimage resistance:** Given x, it is computationally infeasible to find $x' \neq x$ such that $f(x) = f(x')$

As noted earlier, the expression 'computationally infeasible' is application dependent. When it is either unimportant or clear from context which of these commitment properties a given function f satisfies, we will simply refer to it as a *commitment function*. This expression implies nothing about the secrecy of the commitment. Indeed, a digital signature that reveals what is being signed and by whom can be considered a commitment function. Similarly, if $y = f(x)$, where f is a commitment function, then we will say that y is a *commitment to* x.

Another property identified in [14] is

– **near-collision resistance:** It is computationally infeasible to find two values $x' \neq x$ such that $f(x)$ and $f(x')$ differ in only a small number of bits.

This is obviously stronger than collision resistance. How much stronger depends on the size of 'small': the larger the number of bits the stronger the property. Analogously, one can define

– **2nd-preimage-of-near-image resistance:** Given x, it is computationally infeasible to find $x' \neq x$ such that $f(x)$ and $f(x')$ differ in at most a small number of bits.

We actually have need to focus on the chance of finding a value such that the preimages are close. Thus, we define

– **near-preimage resistance:** Given any prespecified value y of f, it is computationally infeasible to find x' such that there exists an x for which $y = f(x)$ and x and x' differ in at most a small number of bits.

Near-preimage resistance is clearly stronger than preimage resistance. This property will prove useful for preventing spatial attacks on lotteries—in which the attacker attempts either to choose ticket numbers that are close to someone else's or to bound a portion of the space of ticket numbers, e.g., where no one else has a ticket, so as to gain an advantage. But for these purposes, near-preimage resistance is still not quite enough. We would like to prevent someone guessing values that share near-preimages, even if they cannot find some or all of those near-preimages. If there exist multiple preimages of a given value, y, of the function and these are near each other, then it is trivial to find a value of the function that has a near-preimage to one or more of these, namely y itself. This does not violate near-preimage resistance because it may not be feasible to find any of those preimages. To preclude nontrivial finding of a value y' of a commitment function that has near-preimages to a given value y, we need the following property.

– **2nd-image-with-near-preimage resistance:** Given any prespecified value y of f, it is computationally infeasible to find $y' \neq y$ such that there exist x and x' for which $y = f(x)$, $y' = f(x')$, and such that x and x' differ in at most a small number of bits.

See [21] for more motivation and discussion of these points.

2.2 Weakly secret bit commitment

The idea of WSBC is similar to that of CSBC. The difference is that we want the secrecy of the bit commitment to be breakable within an acceptable bound on time and/or computation.

Thus, for a WSBC function w, we replace the requirement of preimage resistance with

– **weak preimage resistance:** For any prespecified value y of w, it is *moderately hard* to compute any x such that $y = w(x)$.

The only difference from preimage resistance is that 'computationally infeasible' is replaced by 'moderately hard'. The latter term is borrowed from [6]. Here, we mean to imply both upper and lower bounds on the ease of computation. We will say more about this later.

2.3 Easy computation and easy verification of WSBC

For the applications we envision, it is important that it not be too easy to compute x from $y = w(x)$. Nonetheless, it would be nice if, given x and $w(x)$ one could quickly verify that x is in fact the argument of $w(x)$. This is related to the property of hash functions that they be easy to compute. In fact, basic ease of computation implies this property, although not vice versa.

– **easily computable:** Given any x, it is easy to compute $w(x)$.
– **easily verifiable:** Given any x and y s.t. $y = w(x)$, it is easy to verify that $y = w(x)$.

Under these basic definitions, an easily computed function is also easily verified. This is true if our goal concerns the WSBC function itself. However, it is possible to separate the verification of the commitment from the verification of the WSBC computation of that commitment. This is illustrated via a straightforward combination of CSBC and WSBC functions: Given a CSBC function s and a WSBC function w, we can construct a new WSBC function

$$w_s(x) = \langle s(x), w(s) \rangle$$

As long as s is easy to compute, then it is easy to verify that $w_s(x)$ is a commitment to x.[1] However, unless w is also easily verifiable, w_s will not be easily verifiable as that is defined earlier. We thus present a new definition to reflect this sometimes desirable property.

Let f be a commitment function (secret, weakly secret, or even not secret at all). Let w be a WSBC function such that $w(x)$ is a commitment to x iff $f(x)$ is a commitment to x in exactly the same way and to the same degree as $w(x)$. In other words, f is injective (resp. collision resistant, 2nd-preimage resistant) iff w is, and the mathematical interpretation of 'computationally infeasible', 'small number of bits', etc. is the same in both instances. Then, we will say that w is *commitment-equivalent* to f ($w \equiv_c f$).

– **easily commitment-verifiable:** Given any x and y s.t. $y = w(x)$, where $w \equiv_c f$, it is easy to verify that $y = w(x)$ or to find z s.t. $z = f(x)$ and verify that $z = f(x)$.

As our example mentioned earlier illustrates, while $w_s \equiv_c s$, it may still be that $w \not\equiv_c s$. A specific instance of commitment-equivalent functions such that one of the functions is easily commitment-verifiable is discussed in Sect. 2.4.1.

2.3.1 Shortcuts for computation and verification

Not all WSBC functions will be easily computable, or even just easily verifiable. However, it may be the case that there is some trapdoor or shortcut that makes the computation or verification easy. In fact, with respect to easy verifying of commitment, our example of w_s mentioned earlier is such a function. This also shows s to be a natural overloading of notation, standing for both 'shortcut' and 'secret-bit-commitment function'. If we let w_s in general be a WSBC function parameterized by a shortcut s, then we have the following definitions.

– **shortcut computable:** Given any x, and given s, it is easy to compute $y = w_s(x)$.
– **shortcut commitment-verifiable:** Given any x and $y = w_s(x)$, where $w_s \equiv_c f$ and given s, it is easy to verify that $y = w_s(x)$ or to find z s.t. $z = f(x)$ and to verify that $z = f(x)$.

Note that for the case of a CSBC function serving as a shortcut, $s = f$ in this definition. Here 'shortcut' is particularly apt, not just for notational convenience but also because its use is to speed up the computation or verification of the commitment. Another use of trapdoors with respect to commitments is in "chameleon commitment" and "chameleon signatures" [12]. In those, the trapdoor bypasses not the computation time but the commitment: the commit is collision resistant except to the holder of a private key. These can be used to allow a recipient of a signed commitment to confirm it in a way that he cannot abuse, e.g., by prematurely revealing it to third parties.

2.4 Temporarily secret bit commitment

As we have noted, many of the properties we have set out earlier are properties shared with cryptographic hash functions—except that hash functions are generally designed to be too strong for weak preimage resistance. A natural candidate for WSBC would then be some appropriately weakened hash function. For example, suppose we wish to show commitment to a random fixed-length binary value r from a relatively small space. A natural suggestion would be to use a hash of $s\hat{}r$, the concatenation of a fixed binary string s to r.[2] The length of r should be fixed so that the effort to find r by brute force search on $w_s(x) = h(s\hat{}x)$ is within the upper and lower bounds required by weak preimage resistance, and h should be some hash function believed to be preimage resistant, e.g., SHA1. This or a related approach may be appropriate for applications where the goal is simply to have some computational load involved in breaking the bit commitment. Perhaps, the earliest example is Merkle's puzzle approach to key establishment [15]. There communicants are able to establish keys by the following technique. Alice sends to Bob a large number of weak encryptions of keys and ID numbers (both keys and ID numbers should be randomly chosen). Bob breaks one and sends the ID number to Alice. They now have a session key. But, to get that key, an attacker would have to do work on the order of the product of the work each of them must do. This approach generally has the advantage of quick verifiability; however, it may not be suited to all applications. The problem is that it is easy to break up the search space. Thus, our bound cannot

[1] We mean here that a unique CSBC function s and WSBC function w must be given first and w_s defined by them. If one is allowed to vary the choice of s and/or w within the definition of a single w_s, then even if both choices are adequate as commitment functions, the constructed function might not be.

[2] A similar use of hashes is described in [13,1], ironically for strengthening rather than weakening secrecy.

be specifically related to time. Any accuracy we may have on bounds on computation time cannot be assumed to indicate real-time bounds: the more processors that run the computation, the faster they will find r. Also, we can only predict an average time to find r. There is no way to guarantee that it cannot be found earlier. This would suggest a more restricted class of WSBC in which the breaking of the bit commitment can be more accurately related to time.

Following [17], we suggest that the important feature is that it be calculated by an "inherently sequential" computation of predictable length. By 'inherently sequential', we mean a sequential computation for which the results of the preceding computations are needed to calculate the next value. In this way, we may speak of temporarily secret bit commitment (TSBC). For a TSBC function w, we replace the requirement of weak preimage resistance with the following more precise requirement.

– **temporary preimage resistance:** For any prespecified value y of w, to find any x such that $y = w(x)$, should require an inherently sequential computation of length that is within prespecified bounds.

2.4.1 TSBC via time-lock puzzles

This is a very brief summary of a technique set out by Rivest, Shamir, and Wagner in [17]. Roughly, they take a secret s and encrypt it with a strong key K. They then perform repeated squares of a value wrt a composite modulus, the two large prime factors of which are known to the committer. They then effectively encrypt K with the result of those squarings. Knowing how to factor the modulus allows the committer to get the result of the repeated squaring much more efficiently. Anyone else has no choice but to perform all the squarings. This does not appear to be parallelizable in any way. The only advantage to be gained is by doing it on a faster processor, and the range of available processor speeds can be readily determined. Assuming the speed of the processor, this has very predictable computation time. Once, the squarings have been performed, the key K is revealed and s can be uncovered.

Note that there is a straightforward generalization from this technique to quickly verifiable TSBC for any message M. First encrypt M with some strong key K. Then, given $\{M\}_K$, take any TSBC function w and commit to K by forming $w(K)$. The TSBC to M is thus, $\langle\{M\}_K, w(K)\rangle$. (This commitment depends on the standard property of some encryption algorithms that it is hard (or impossible) to find M, M', K, K' s.t. $\{M\}_K = \{M'\}_{K'}$. So, e.g., XOR-based stream ciphers would be inappropriate).

However, in general, there is no guarantee that it will be any easier to compute $w(K)$ than to derive K from it. The function chosen in [17] is especially nice because it allows quick calculation of $w(K)$ by the person who possesses the factors of the modulus. Thus, the TSBC is not only easily commitment-verifiable but also shortcut computable.

2.5 Delaying functions

Delaying functions are functions that inherently require a certain amount of computation time (sequential steps) to be computed. Roughly speaking then, delaying functions are simply the inverses of TSBC functions. However, these functions need not be one-one—hence need not necessarily be invertible. Also, TSBC is naturally viewed as commitment to some known value, but for some applications, we would like to commit to a value that is not known in advance by anyone. And, if the TSBC is not one-one, the output of breaking the secret will not yield a unique value. In this case, what is required is a delaying function. On the other hand, we may sometimes want to express commitment to a set of values rather than to a single value. This is most naturally expressed as a TSBC function that is not one-one.[3]

A function d is a *delaying function of length l* if:

– **temporary image delay:** For any prespecified value x in the domain of d, to find $d(x)$ requires an inherently sequential computation of fixed length l.

Like TSBC, it is often not enough to simply require an inherently sequential calculation. Other properties of the commitment may be desirable. One of these is that the delaying function maintain all the information of its inputs.

– **information preservation:** If $(x_i, p(x_i))$ gives the probability distribution for a random variable X, and $Y = d(X)$ where the probability distribution of Y is determined by d and the distribution of X, then $H(X) = H(Y)$ (where $H(X)$ is the entropy of X).

Temporary image delay relates to temporary preimage resistance for TSBC functions in the obvious way. Information preservation says that the information in the input to the delaying function X is maintained in the output Y. In fact, this relates to the other TSBC property: the commitment property. More specifically, let $w : X \to Y$ be a one-one onto function, and let Y be a random variable taking values in the range of w, with distribution given by $(y_i, p(y_i))$. Then, it is immediate from the definitions that w is a TSBC function iff its inverse w^{-1} is an information preserving delaying function (from Y to X).

[3] TSBC functions were introduced in [21]. Another approach to the same phenomenon was the use of delaying functions, introduced in [10]. The definitions of these and related concepts discussed herein may vary somewhat from the definitions in the papers where they were introduced.

2.5.1 Local unpredictability

Suppose one has a delaying function d that produces a 100 bit output. And, suppose that the given algorithm for calculating $d(x)$ reveals 99 bits of the output almost immediately; however, the entire inherently sequential calculation is necessary to reveal the last bit. This would be a perfectly good delaying function; albeit one that reveals most of what was supposed to be hidden. Note that this is a possibility even if the delaying function is information preserving.

A delaying function d of length l that prevents such an outcome is

– *locally unpredictable for length* l'. Given x, no algorithm running l' or fewer steps in sequence can distinguish $d(x)$ from a randomly chosen value in the range of d with probability significantly greater than $1/2$.

If $l' = l$, we say simply that d is *locally unpredictable*. Basically, this says that after a calculation of length $\leq l'$, one can do no better predicting the value of $d(x)$ than by simply guessing.

Which particular existing cryptographic functions are locally unpredictable delaying functions is as yet unexplored. However, this property would seem to be closely related to, and in some cases straightforwardly implied by other classes of long-studied cryptographic properties, e.g., non-malleability [5], plaintext awareness [2], and the strict avalanche criterion [22]. Exploring the relation between these is a more purely cryptologic issue that is outside the scope of this paper. For our purposes, it is enough to specify the property in order to show how functions that satisfy it can be used in the design of lotteries and other applications.

Local unpredictability would seem to imply that the probability of any output is uniformly distributed over the image of X under d. But, given x, the probability that the output of d is $d(x)$ is 1. So, we cannot say that the probability of the output after a calculation of length l' is anything but 1 or 0. Thus, in place of the probability of $d(x)$, it seems more natural to speak of the *predictability* of $d(x)$ after a calculation of length l'. We explore this concept briefly in the next few paragraphs.

Generally, the predictability of a value $f(x)$ after a calculation of i steps, $\text{Pred}(f(x) \mid i)$ is the probability that such calculation produces $f(x)$. Thus, let f be a delaying function of length l that is locally unpredictable for length l'. Then, for all x, $\text{Pred}(f(x) \mid l') = 1/\text{range}(f)$. We can also speak of the predictability of a value at a given time, $\text{Pred}(f(x), t)$ as meaning the predictability of $f(x)$ given any calculation done up to time t.

Restricting to locally unpredictable delaying functions (even locally unpredictable for length $l' < l$) may be too stringent. The function is ultimately expected to completely reveal $d(x)$. It would be nice if, similar to threshold cryptographic schemes, no information is revealed for calculations of length $\leq l'$. But, it may be that some information is leaked. More generally,

A delaying function d is ε *unpredictable for length* l, (ε, l)-unpredictable, if , for any x, $\text{Pred}(d(x) \mid l) < \varepsilon$.

For any delaying function d of length l, the predictability of outputting $d(x)$ after a calculation of length l' is $f(l')$ for some function f, where $f(l') = 1$ implies $l' \geq l$. In other words, the predictability of $d(x)$ given an initial calculation of length i is determined as follows:

$$\text{Pred}(d(x) \mid i) \begin{cases} < 1 & \text{for} \quad i < l, \\ = 1 & \text{for} \quad i \geq l. \end{cases}$$

2.6 Parallel and probabilistic speedup of verification

It may be somewhat surprising that, even for computations designed to resist parallelization, we can speedup the verification of a result by parallelizing it. This is obvious once we note that even an "inherently sequential" computation must have intermediate results.

Any sequential computation requiring S steps can be broken into arbitrarily many smaller pieces. For example, if $S = nk$, and c_i is the i^{th} intermediate result, the verification of a computation of c_S can be broken into n pieces of the form $\langle c_i, c_{i+k} \rangle$. The verification of this piece is then to confirm that k computations on c_i yields c_{i+k}. And, the verification time of calculating c_S can be thus reduced by a factor of n (assuming n processors working in parallel).

Using this method, it is possible to split the verification of the outcome of the lotteries described later. Individual ticket purchasers are unlikely to have many processors at their disposal; however, this would allow mutually trusting customers to split the duties.

Moreover, by splitting things up in this way, it is possible to check computations probabilistically. This could also ameliorate the limitation on easy verification. By what distribution should we choose segments to verify? If an error was equally likely to come in any segment, then the best strategy for checking would be to pick some number of segments at random. But an adversary might gain more advantage by, e.g., doing most of a computation correctly and then cheating somewhere near the end. In this case, it makes sense to check the last segments and then check backwards with increasingly less likelihood of checking as the initial segment is approached. Attaching probabilities to adversary behavior is tricky even when any sense can be made of it at all. For example, if the adversary can only construct an attack by splitting the computation into two roughly equal pieces, then it makes the most sense to check segments near the middle, perhaps with a gaussian distribution. Many other scenarios are possible. And, if we know nothing about the probability of an error, then all we can say is that testing more segments increases our assurance, although we cannot say by how much. Nonetheless, if we know some minimal things about the potential for errors, we need not necessarily know the adversary's error distribution strategy to say something meaningful about the probability that a check detects an error.

Some attacks may require introducing some number of errors not likely to be "near" each other. For example, to run a parallel attack to get an early result with the appearance of a sequential computation, one could split the computation into *n* roughly equal segments, much as in the checking technique we are discussing. The value at the beginning of each segment (other than perhaps the first) would be a made-up value, an error in the sequential computation. In this case, sampling has an even greater chance of catching the attack. So, for instance, suppose an attack requires splitting a calculation, resulting in errors in 10 disparate locations, and suppose we split the checking into 50 segments and verify 10 of them chosen uniformly at random. Then, no matter how the adversary distributes his errors among the segments, an attack is detected with over 92% chance, and the checker has much less work to do than the adversary. Checking involves doing one-fifth of the computation total, in ten parallel segments, each only one-fiftieth of the total length. But to mount the attack, the adversary must do the entire computation, split into 10 parallel segments, each comprising a tenth of the total length. This is just an example and may not be representative. In the case of collective delay, it is the lottery service itself that is the threat. So, risk of a single detection threatens the whole business. For this case, a 92% chance of detection may be overkill. On the other hand, if payoffs are large or inadequate checks are in place for attacks by insiders who may not concerned with the long-term reputation of the service, then it may be much too low. In general, the value of such probabilistic checks is thus closely tied to the accuracy of the threat model on which they are based.

This is all that we will say about various kinds of commitment and delaying functions. We leave for future work the cryptologic investigation of ones that are introduced herein. The remainder of the paper looks at various lottery designs that make use of such functions.

3 Abstract lottery

In this section, we will set out the assumptions and requirements of a high-level lottery design. All of the lotteries described in subsequent sections will be based on these assumptions and should satisfy these requirements. However, in future sections, we will outline additional requirements and assumptions of particular lotteries.

3.1 General design

The general design of our lotteries is as follows: Customer's will purchase tickets from a lottery agent. Numbers they submit will be used to make up a list of ticket numbers, effectively the list of all possible winning tickets. The list is regularly updated, signed, and posted during the purchase period by the lottery agent. The complete list is signed and posted shortly after the close of purchase. Also from numbers submitted in each ticket purchase during the purchase period (or a previously specified *critical purchase* subperiod) will be formed the argument for a locally unpredictable delaying function. Numbers used for the argument of the delaying function are called *seeds*. (Seeds and other relevant ticket information are also on the posted list of ticket numbers.) The output of that function will take long enough to calculate that no one could predict the output during the purchase period. Winners will be determined by a relation between the lottery function output and the numbers on the ticket list. This very general description is meant to give an idea of how the lotteries are intended to work. More specifics are given in the following assumptions and in the descriptions of the individual lottery designs.

As an example of the subtleties that arise, the listed ticket numbers are actually an appropriately chosen function of customer- supplied ticket-number inputs. Since the list must be publicly posted, this allows customers to choose unpredictable (adequately random) ticket numbers that prevent anyone else (including the lottery agent) from having a valid ticket with the same (or even a similar) number. One is not required to choose a random ticket-number input, but failure to do so on your part may mean that someone will be able to always choose so that you must split any winnings you get.

3.2 General lottery requirements

For convenience, a summary of the notation introduced in this subsection and the next can be found at the end of the next subsection.

R1. *One-pass.* Lottery purchase occurs in a single interaction. (Determining the results of the lottery does not require input from purchasers subsequent to their purchase of the tickets.)

Fairness requirements.

R2. *Fair selection.* At t_c, the time ticket purchase closes, the predictability that a purchased-ticket number wins is invariant over the set of purchased-ticket numbers.

R3. *Ticket unpredictability.* Given a provably valid ticket whose ticket-number input was chosen randomly and kept secret, it is computationally infeasible for anyone to choose a provably valid ticket whose number overlaps this one in more than a negligible number of bits.

Note that *provably-valid* means here that the owner of the ticket in question can prove possession of the ticket-number input. For simplicity, we will sometimes later use *well-chosen* to refer to tickets whose ticket-number input was chosen adequately randomly and kept secret at least until after the close of ticket purchase.

Verifiability and non-repudiability requirements.

R4. *Ticket verifiability.* Within some interval, I, after a ticket purchase, customers can verify whether they have a valid ticket. Also, the vendor cannot repudiate the validity of a purchased well-formed ticket.

R5. *Lottery outcome verifiability.* Anyone can verify a lottery result after the close of the lottery within time $t_c + I'$.

R6. *Lottery outcome non-repudiability.* The vendor cannot repudiate the results of a lottery run.

3.3 General lottery assumptions

These apply to all lotteries we will discuss in this paper. We offer brief explanation of them as they are presented. However, they are best understood in the context of their use in proving requirements.

A1. *Goodness of cryptographic mechanisms and parameters.* We assume that keyed cryptographic operations prevent any undetectable modification of fields to which those operations are applied. Also, we assume that cryptographic keys, nonces, and blinding factors are adequately randomly chosen from an adequately large space to prevent random collisions or revealing of secrets by cryptanalytic attacks.

A2. *Confidential secrets.* We assume that private keys and other secrets are not accidentally revealed or otherwise communicated out of band to unauthorized principals.

A3. *Clock synchronization.* Clocks are adequately (i.e., loosely) synchronized.
All times and intervals are assumed to take any skew into account.

A4. *Bounded rate of computation.* We assume that all sequential computing operations occur at a rate that is tightly bounded above (r_{ub}) and below (r_{lb}).

A5. *Ticket purchase fairness.* Purchasers pay for tickets iff they can prove ticket ownership by $t_c + I_0$ (time of purchase closing plus a fixed interval). (For example, we do not distinguish here between purchase by a fair exchange protocol and trust of the ticket vendor to be fair in this way. We also treat as contextually dependent the appropriate time by which payment must occur.)

A6. *Purchased implies listed.* Numbers of tickets that are paid for appear on the ticket number list. If distinct, customer inputs (seeds) for the lottery function are also listed.

A7. *Nontrivial.* At least one ticket is purchased.

A8. *Ticket-number function unpredictable and secret.* Valid ticket numbers are values of a 2^{nd}-image-with-near-preimage-resistant CSBC function. (Note that this only describes the ticket-number function. For purchased tickets, arguments for the function are chosen privately by the purchasing customer, and it is up to them to have well-chosen ticket-number inputs. If both of these are true, then spatial attacks are strongly curtailed [21].)

A9. *Lottery function.* The function that calculates the lottery output is composed of a, an amalgamation function, and d, a locally unpredictable delaying function of length l (where $(t_c - t_p)/r_{ub} < l < l_{ub}$). Here t_p is the time that (critical) purchase begins.

1. If the range of d is the same as the range of the ticket-number functions, then the lottery function is $d \circ a$ and a is a preimage resistant and 2^{nd}-preimage resistant hash function from the ticket seeds to the same range.
2. If the range of d is Y^n where Y is the range of ticket numbers and n is the number of tickets on the list, then the lottery function is $a \circ d$ and a is any function s.t. $H(a(X_1, \ldots, X_n)) \geq \max H(X_i)$.

A10. *Large ticket-number range.* The range of ticket numbers (and the lottery function) is too large to precompute more than an insignificant number of lottery outputs.

A11. *Winner calculation.* The winning ticket number(s) is a bounded or minimum Hamming distance(s) from the lottery output. Given the output and the list of ticket numbers, winners can be calculated in a negligible amount of time, I_ε.

The Hamming distance $d(x, y)$ from x to y is the number of positions in which the two strings differ [18]. There are other approaches that could be adopted to determine the winner. We limit ourselves to this one for convenience.

Publicly available information

A12. *Lottery parameters.* All parameters (purchase start time, purchase end time, means for determining winners, means for determining and distributing prizes (winnings), posting location for purchased ticket lists and winners, format for ticket purchases, public keys and associated certificates, lottery run identifier) are signed[4] by the lottery agent, and this is publicly available prior (by the same clock) to the purchase start time.

A13. *Listed tickets implies available.* The list of valid tickets (including ticket numbers and, if distinct, seeds for the lottery function) is labeled for the given lottery run and signed by the lottery agent, and this is available to interested parties by $t_c + I_0$.

A14. *Customer verification and storage.* Customers adequately collect and store available information to prevent repudiation by the agent.

A15. *Secure purse.* All funds are secured long enough after t_c to adequately verify and protect a complete lottery run. (This means, e.g., that either the vendor has inadequate incentive to abscond with the lottery funds (alone or via collusive detectable cheating) or that both winnings and vendor share are held by a trusted financial authority long enough for verification or customer demonstration of vendor impropriety to have occurred.)

3.3.1 Notation summary

For the reader's convenience, we summarize the notation introduced in the last two subsections.

t_c	the time at which purchase of lottery tickets closes
t_p	the time at which the lottery ticket critical purchase period begins (may be the same as the time purchase begins)
I	the maximum interval it should take a customer to verify the validity of a ticket after it is purchased
I'	the interval after t_c in which anyone should be able to verify the lottery result
I_0	the interval after t_c in which any purchaser should be able to provably verify ticket ownership (if he pays for the ticket)
I_ε	the interval after lottery output is public by which anyone should be able to calculate the winner(s)
r_{ub}	the fastest rate at which any lottery principal can perform computations
r_{lb}	the slowest rate at which any lottery principal can perform computations
l	the length of the locally unpredictable delaying function used in a lottery
l_{ub}	the maximum acceptable length for the locally unpredictable delaying function used in a lottery

[4] Signatures are assumed to be valid. Aside from issues addressed by A1, questions of, e.g., signature-key compromise or expiration etc., are assumed to be adequately addressed independently and are outside the scope of this paper.

4 Lottery with fixed expected value

In this lottery, the expected value of payoff is fixed regardless of the number of legitimately sold tickets or even regardless of whether the vendor simply gives large numbers of tickets away for free to his friends. This is both an advantage and a disadvantage. Unlike the lottery in Sect. 5, there is no need to assume mechanisms to prevent denial-of-service on ticket purchase, no need for customer authentication or for guaranteeing a minimum number of customers in a lottery run. So, there is automatic immunity to such attacks without adding security overhead. On the other hand, and also unlike the lottery of Sect. 5, (1) there is no rollover, so jackpots are limited by the number of tickets purchased in a single run, and (2) the delaying function must use input from the whole ticket list, effectively lengthening the time to calculate the winners. It may be helpful, although admittedly also misleading, to think of these as analogous to various US state lottery games. The lottery in this section is roughly comparable to the scratch-off ticket lotteries that have a fixed payoff expectation regardless of what happens with other tickets. The lottery in Sect. 5 is roughly comparable to lotto games, where jackpots can be rolled over to later runs of the lottery. Of course, all our lotteries are completely unlike existing lotteries in that outcomes are determined by customer chosen input rather than by an unrelated and random input.

Because the outcome is determined by the listed tickets, strictly speaking the expected value of all nonwinning tickets is zero once purchase has closed and all the tickets are listed. Similarly, the expected value of a single winning ticket is simply the payoff of the lottery. Thus, the correct notion we need is *predicted value* which is to expected value exactly as predictability (defined in Sect. 2.5.1) is to probability. The predicted value of a given value is thus the payoff times the predictability of that value.

Requirements

R7. *Constant predicted value.* The predicted value of a ticket with a randomly chosen and secret number input is invariant over any set of purchased tickets and from one set to another.

Assumptions

A16. *Simple pari-mutuel.* $W \leq B$, where W is the total value of the winnings in a given run of the lottery and B is the total purchase cost of all tickets. More specifically, we assume that $B = c \cdot W$, for some fixed c s.t. $0 \leq c \leq 1$. ('Pari-mutuel' means that the winnings are all from money bet in the lottery. 'Simple' means that all prizes are awarded; there is no rollover.)

A17. *Random input.* The seed (lottery function input) provided in at least one purchased ticket contains at least one bit of randomness.

4.1 Lottery design

As alluded to earlier, this lottery design has no registration and makes no specific assumptions about how ticket purchase is accomplished. It may or may not be anonymous. And, the lottery agent may even issue tickets to himself or his friends without any payment at all. Because of the simple pari-mutuel payoff, this will not affect the expected value of any sold tickets. (More on that below.) The only assumption we make about ticket purchase is that it is fair, as described in Assumption A5.

4.1.1 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 purchase phase was chosen randomly.

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

1. Input preparation. Compute $h(s_1, s_2, \dots)$ where s_1, $s_2, \dots$ represents the concatenation of all seed parameters of tickets on the purchased tickets list (in the order listed). We require that $h()$ be preimage and 2nd-preimage resistant. The preimage resistance property counters the adversary from using a "precomputed" lottery function calculation. The 2nd-preimage resistance property counters attacks associated with the reuse of previous hash computations. This may be interesting if the preimage resistant hash function effectively ignored a subset of input bits. Perhaps, the adversary could then attempt to control ticket purchases associated with significant bits.

2. Lottery function calculation. Input the result of step 1 to a locally unpredictable delaying function (i.e., as in Definition 2.5.1).
   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. The delay calculation is achieved through a moderately hard calculation that must also preserve the randomness in the input.

In Assumption A11, it was stated that the winning ticket number(s) is a bounded or minimum Hamming distance(s) from the lottery output. Since this lottery design is simple pari-mutuel, winners cannot be determined by fixed bounds on Hamming distance because there might not be any purchased ticket numbers within a fixed bound. Winners can therefore either be determined by a minimum distance or there could be a tiered calculation: first prize to everyone within the first bound to yield a winner, second prize to everyone in the next bound to yield additional winner(s), etc.

Example winning entry calculation. Where L is the number of lottery tickets sold, an implementation of these three steps might be:

1. Concatenate in some predefined order the seed parameters on the ticket list. 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 predefined 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 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. That said, whether or not such a function satisfies Assumption A9, in particular whether or not it is actually a locally unpredictable delaying function of appropriate length is a question that is left for future work.

4.2 Evaluation

One-pass requirement, R1:

The one-pass requirement is obviously satisfied since the lottery design requires no input from customers after their tickets are purchased.

Fair selection requirement, R2:

By assumptions A7, A6, and A17, there is at least one bit of randomness in the input to the lottery function. Thus, by Assumption A9, before t_c, values determined by the lottery function for a given input (run of the lottery) cannot be predictably distinguished from randomly chosen values in the range of the function. In other words, according to any calculation done until t_c, all outputs of the lottery function are equally predictable. And, by Assumption A8, the ticket numbers of valid tickets cannot be predictably chosen to be close to any other value in the range of ticket numbers, which is the same as that of the lottery function. Thus, by Assumption A11, as of t_c all ticket numbers are equally predictable to be a winning distance from the output of the lottery function.

Ticket unpredictability requirement, R3:

This is immediate from assumptions A8 and A2.

Ticket verifiability requirement, R4:

By assumptions A5, A6, and A13, any customer can determine by $t_c + I_0$ whether tickets that he can prove are his are on the ticket list, i.e., valid. Ticket purchase could be done no earlier than t_o (time of purchase opening). Thus, customers can determine within a fixed interval of length $(t_c + I_0) - t_o$ after a purchase whether or not they have valid tickets. As noted, by Assumption A5, the lottery agent cannot repudiate the purchase of the ticket. Thus, by assumptions A13 and A14, he cannot repudiate the validity of a purchased ticket.

Outcome verifiability requirement, R5:

By Assumption A13, the list of ticket numbers and perhaps redundantly, the input to the lottery function is available to anyone interested by $t_c + I_0$. By Assumption A12 any other ancillary information for calculating the lottery function output is available prior to purchase start time. By Assumption A9, the lottery function is a delaying function of length no more than l_{ub}. By Assumption A4, computation can be done at a rate no slower than r_{lb}. And, by Assumption A11, the time to calculate the winner(s) given the lottery output and the ticket-number list is I_ε. Thus, anyone will be able to verify the lottery result by t_c+I', where $I' = I_0+l_{ub}/r_{lb}+I_\varepsilon$.

Outcome nonrepudiability requirement, R6:

By assumptions A13 and A12, all of the information required to calculate the earlier mentioned verification is identified and properly signed by the lottery agent. And, by Assumption A14, enough honest customers record and store this signed information to prove its origin if needed. Thus, the agent cannot repudiate the results of a lottery run.

Constant predicted value requirement, R7:

By Assumption A16, the payoff of any lottery run is always the same constant fraction c of the total purchase cost of listed tickets in the ticket list. (Note that this is independent of whether or not the listed tickets represent legitimate purchases or were simply listed by the lottery agent.) If all tickets numbers were independently chosen at random from a reasonable large space, then, except with negligible probability, ticket numbers would not overlap except in a negligible number of bits. Thus, the predicted value of any ticket would be c times the cost of an individual ticket. We cannot assume that all ticket numbers are chosen at random. However, by the proof of requirement R2 above, the predictability that any listed ticket number wins is invariant over all listed ticket numbers in a run of the lottery. And, by property R3, the predictability that any provably valid ticket number more than negligibly overlaps another (winning) provably valid well-chosen ticket number is itself negligible. Thus, any well-chosen and provably valid winning ticket is (except negligibly) the sole winning ticket. Therefore, the predicted value of any winning ticket is c times the price of an individual ticket.

5 Lottery with rollover

In this lottery design, winnings may come from outside of the amount bet in a given lottery run. This has the obvious advantage of allowing for increasing jackpots if nobody wins, which may attract more players. It also has a less obvious advantage: since winnings can come from outside a lottery run, the lottery output function can be computed on a predetermined subset of the tickets purchased. By making this subset occur toward the end of the purchase period (i.e., in the critical purchase phase), we can have a shorter delay on the calculation of the lottery output function hence a quicker determination of the winner. We will not formally prove these requirements, but they will follow obviously and immediately from the design description. Nonetheless, to be explicit we state:

Requirements

R8. *Not simple.* The value of winnings may exceed the total amount bet in a given lottery run.

R9. *Shortened winning calculation.* The lottery output function has a (shorter) delay length governed by the (shorter) critical purchase phase rather than by the entire purchase phase.

Assumptions Most of the assumptions we make concerning this design may not be understood until the relevant part of the design is discussed. However, so that they are all in one convenient place, we state them here.

A18. *Minimum listed number of sources.* Let n be the actual number of distinct sources of tickets listed as purchased during the critical purchase phase. It is publicly verifiable from the list of purchased tickets, that $n \geq source\_min$, for any practical value of $source\_min$.

A19. *Sources approximate individuals.* For any practical maximum assumed coalition size $coalition\_max$, it is possible to set a practical minimum number of sources $source\_min$ such that the maximum number of sources obtainable by $coalition\_max$ individuals is less than $source\_min$.

A20. *Random input.* Given any collection of listed tickets purchased by more than *coalition_max* distinct individuals, the seed (lottery function input) provided in at least one purchased ticket contains at least one bit of randomness.

A21. *Accurate valid purchase time.* The time of any well-chosen ticket purchase can be determined from the published ticket list to an adequate degree of accuracy.

A22. *No denial-of-service.* Denial-of-service attacks do not occur. (We assume external mechanisms to detect and deter denial-of-service.)

Measures to prevent 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. For example, if the lottery agent does not participate in the coalition, and 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. However, this could easily be detected by an independent service that simply makes test purchases periodically. While such a third party is trusted on some level, it is not nearly as trusted or as intimately involved as one that would monitor the fairness of picking the winners. There are a number of services that currently monitor availability on the Internet; although usually for the direct benefit of the site being monitored rather than as a watchdog against the site. One could easily imagine one or more of these independent services being used to monitor a lottery agent for availability of (fair) ticket purchase.

5.1 Lottery design

In this lottery design, there are four phases: *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 as part of the lottery parameters. In the lottery of Sect. 4, the critical purchase phase was the entire purchase phase. Thus, there was no need to distinguish it. The same mechanisms that allow rollover of winnings from one lottery run to another allow us to fairly calculate the lottery output based only on seeds from the critical purchase phase.

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

5.1.1 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 differentiation. This requires mapping between an individual and a corresponding source that will be associated with ticket purchase. The sources might typically use certificates or hardware devices. The integrity of the mapping is maintained by the following assumption:

Limited obtainability of distinct valid identities. It is difficult or expensive for an individual to obtain more than a small number of certificates or hardware devices. Also, only the owner can prove that he is authorized to use the certificate or hardware device.

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. Hardware devices must be capable of producing output that is publicly verifiable to not have come from any other such device. This is necessary so that the number of ticket purchasers can be verified from the ticket purchase list.

Note that this assumption does not imply that it is impossible to obtain even many distinct valid identities. For example, an adversary could pay people 10 dollars each to register and give the adversary the resulting credentials. But, an adequate threshold for *source_min* would require both a large outlay of money and/or a large willing number of colluders to participate. If lottery payoffs are in the millions of dollars, *source_min* should probably be in the tens of thousands. Alternatively, one could simply pay large sums of money to the credential issuing authority to obtain the credentials. To some extent, there is a general threat to the validity of credentials that is beyond the scope of this paper; in any case, this is a parameter that can be tuned to the expected level of threat. Trust in an individual authority could be diminished by means of threshold registration, as described in [20]. Also as described therein, registration can be done in such a way as to yield a pseudonymous certificate in case customers do not wish to be identified whenever they play. Any of these mechanisms are compatible with anonymous purchase of tickets.

The most natural way to insure a minimum number of distinct purchasers of tickets is to have each ticket on the list be authentically bound to a certificate, e.g., via signature. However, we need not assume any particular mechanism. For example, it is conceivable that there is a publicly verifiable value that is computable from the ticket list only if a threshold number of distinct sources provided tickets. Thus, the assumptions we formally make in this regard are A18, A19, and A20, stated earlier.

5.1.2 Purchase

Unlike the lottery of Sect. 4, tickets are to be sold only to registered client agents. There is a valid purchase period as before, but also a subperiod which we now describe.

5.1.3 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_c. All purchased tickets are listed in the ticket list. But, only the seeds of those listed as purchased during critical purchase are used to calculate the output of the lottery function.

Unlike the fixed-expected-value lottery, we have a minimum number of purchasers and a critical purchase period. We thus need to be reasonably sure that, with few or no exceptions, tickets listed as purchased during the critical purchase phase were actually purchased then. Otherwise, we could take a number of ‘stooge’ valid distinct purchasers whose purchases were done early in the lottery run and combine them with the coalition to exceed *source_min* and still calculate the lottery outcome early enough to cheat effectively. Therefore, we assume that it is possible to determine with adequate accuracy the time of ticket purchase. A natural way to accomplish this is to assume adequately synchronized clocks, adequately fast purchase transactions, and the secure binding of time stamps to tickets, e.g., by signatures. The ticket list should thus include the time of purchase given by the customer and the time of posting to the list. In this context, we make Assumption A21, as stated earlier.

5.1.4 Winning entry calculation

The winning entry calculation for this lottery with rollover is almost identical to the lottery with fixed expected value presented in Sect. 4.1.1. The only difference in calculating the lottery output is that it is done using only the tickets purchased during the critical purchase period critical purchase phase p. 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 by any feasible coalition of dishonest individuals.

The other difference is that, since the lottery is not intended to be simple, prize money awarded for a given lottery run need not be a fixed proportion of the take in that round. In fact, there need not be any prizes awarded in some rounds. Therefore, we may use a tiered calculation as in Sect. 4.1.1, but where, e.g., no first prize is awarded if no purchased ticket’s number falls within the first bound.

5.2 Evaluation

Fair selection requirement, R2:

By assumptions A18, A19, and, A21, at least *coalition_max* distinct individuals purchased tickets listed as purchased in the critical purchase phase. Thus, by Assumption A20, there is at least one bit of randomness in the input to the lottery function. The rest of the proof is exactly as in Sect. 4.2.

The remainder of the general lottery requirements, R1, and R3 through R6, are shown exactly as in Sect. 4.2.

6 Lottery using individual delay

The two lottery designs given previously differed in how they managed the predictability and expected value of winning. However, they were the same in that they both computed the lottery output in essentially the same way: amalgamate seed values based on purchased tickets and then run the amalgamated value through a delaying function. The design to be given presently differs in that individual values are run through delaying functions and the result then amalgamated. This is independent of issues of rollover, etc. and is thus compatible with either of the previous designs in this respect. The advantage of the approach herein is that verification of lottery results is potentially much faster, and possibly even computation itself.

The lotteries mentioned earlier do not require anyone to trust the vendor or anyone else concerning the fairness of the lottery. Trust lies solely with the algorithms that transform lottery input to lottery output (and their implementations). However, there is still an element of trust vs convenience that must take place. Specifically, if anyone wishes to confirm the outcome of the lottery, she must reproduce the presumably time consuming and/or expensive calculation on her own trusted platform and wait for the results. A step on the way to overcoming this is to have an algorithm operate not on the entire conglomerate list of ticket entries but on each ticket number individually. The advantage here is twofold. First, the ticket holder can in this way know the result of applying the algorithm to his ticket number ahead of time. More importantly, the algorithm may be shortcut computable. Or, since it is the input to this algorithm that is made public, its inverse might be shortcut computable or even easily computable (without a shortcut). In this sense, the delaying algorithm is perhaps most naturally thought of as the inverting of a TSBC function.

We will call the input s the purchaser chooses for the TSBC function w the *private ticket number* and the output $w(s)$ the *public ticket number*. In other words, the input to the delaying function is the public ticket number and the output is the private ticket number. The public ticket number is what is given on the ticket list. The locally unpredictable delaying function of length l is in fact a vector of the length l functions used to compute each of the private ticket numbers. The final calculation of the lottery output can be a fast amalgamation of the private ticket numbers, presumably taken in the order that their corresponding public ticket numbers appear on the list. The only requirement on the amalgamation is that if any of the inputs are completely random the output can be anywhere in the range.

Perhaps, the most important advantage of the splitting of the TSBC (the delaying) is that it allows the individuals to precompute their own commitment value. This has several advantages, but the paramount one is that it can be based on a secret that the individual possesses. This means that the lottery result can be quickly and easily verified by anyone. It also means that the lottery result can potentially be determined more quickly without any threat to fairness.

With all these advantages, the reader may be wondering why one would bother with the previously given designs at all. There is an important potential trade-off using this design that may restrict its applicability in lotteries where there are many purchasers (perhaps more than a thousand or so). We will discuss this limitation in Sect. 6.2.

Requirements

R10. If all tickets are well-formed, the lottery outcome is easily commitment-verifiable.
R11. If all tickets are well-formed, the lottery outcome is shortcut commitment-verifiable.
R12. If all tickets are well-formed and if all shortcuts are revealed (after the close of purchase), then the lottery outcome is shortcut computable.

Assumptions We make one assumption specific to this design beyond those given in Sect. 3. Also, while the contributions of this design are independent of those of the earlier mentioned designs, there is some complementarity.

A23. *Tickets not all revealed.* At the end of ticket purchase, the shortcut to at least one well-chosen purchased ticket has not yet been revealed by its honest ticket purchaser. (If the lottery has a critical purchase phase, this ticket should be listed as purchased during that phase).

6.1 Lottery design

Registration and purchase will be as in the previous designs depending on whether the features of one design or the other are desirable. We therefore proceed to calculation of the winning entry.

6.1.1 Winning entry calculation

Calculation of the winning ticket(s) for this lottery is as follows:

1. Each public ticket number is input to a locally unpredictable delaying function.
2. The outputs of these (the private ticket numbers) are input into an amalgamation function that preserves the randomness of the most random input.
3. The output of the amalgamation function is compared via Hamming distance to the private ticket numbers. The policy for whether the winner(s) are minimum distance, fixed bound, or minimum tiered distance, etc. will depend among other things on whether the lottery is intended to be simple pari-mutuel.

Example winning entry calculation. Where L is the number of lottery tickets sold, an implementation of these three steps may be:

1. Public ticket numbers are determined via the time-lock approach of [17] (described above in Sect. 2.4.1). These are what is posted on the ticket list, private ticket numbers are thus computed by passing each public ticket number through the corresponding delaying function.
2. The list of private ticket numbers is amalgamated by concatenating them in their posted order and running the result through SHA1.
3. The winner is chosen by using Hamming distance by whatever the posted method is (e.g., depending whether rollover is intended).

Note that the suggested use of SHA1 assumes that the private ticket numbers are 160 bits, and that SHA1 preserves the entropy of the most random private ticket number in the input. If not, this implementation will not satisfy Assumption A9. As noted elsewhere, exploring which specific functions satisfy our design assumptions is outside the scope of this paper.[5]

6.2 Quick verification of lottery results

We can use the quickly verifiable technique of Sect. 2.4.1 to commit to each private ticket number. So, $w(s) = \{s\}_K$, the encryption of s with K. (More precisely, $w(s) = \langle \{s\}_K, w'(K) \rangle$, where w' is a TSBC function used to commit

[5] Also, please note that the last substantive revisions to this paper were made long before the first cracks in SHA1 were published in 2005.

to K.) Once K has been calculated for each of the tickets, all the keys can be published. Thus, anyone can quickly verify the lottery result by performing the encryptions and computing the lottery output. The lottery output may be quickly verified, but it is still necessary to wait for the result of breaking the secret bit commitments before the lottery outcome can be determined. However, depending on the cooperation of the lottery participants, this too can be accelerated.

Once the lottery has been publicly determined to be closed, individual ticket holders can reveal their private ticket numbers and the keys used to encrypt them. If everyone does this, then the lottery outcome can be determined—presumably more quickly than by breaking the bit commitments, which must include a temporal safety margin. There is no guarantee that everyone will cooperate. However, no one can know that they have won (or lost) until this is done. So, it is in the interest of the participants to cooperate. (The calculation is similar in this respect to rational exchange [21].) Even allowing for the irrational, sociopathic, or incompetent ticket holders, and the system errors that may occur, it is probably still in the interest of individual ticket holders to reveal their secrets. This is because the vendor probably has only so many processors to devote to the breaking of the bit commitments. Thus, the more people send in their secrets, the faster the outcome can be determined. This may even imply a solution (for this case only) to a classic coordination problem; when something requires the cooperation of lots of people, it is rational to think that someone will fail to do his part. Thus, it is not worth doing your part. Since doing your part here helps even if some others don't, it is a better bet to cooperate. This may lead to a higher number of lottery runs where everyone actually does send in his secrets. Nonetheless, as the number of purchased tickets increases, the probable number of private ticket numbers that must be revealed without the shortcut is bound to increase as well. In the end, how much of a problem this poses can only be determined empirically.

Note that even if no one reveals his secret, the lottery remains both one-pass and easily commitment-verifiable. It just will not be possible to quickly determine the results originally. However, there is still another potential limitation.

6.2.1 Malformed tickets

The earlier discussion all assumes that each of the tickets is valid. If any of the tickets was not produced via encryption using an appropriate key and the correct algorithm, this can be determined (provided there is enough redundancy to determine that a given message was encrypted using a given key, e.g., by including a hash of the message within the encryption). Such a ticket might be declared invalid, however; the technique for extracting the committed key will yield a unique key whether or not this is a key that was used to encrypt the private ticket number; thus, the private ticket number can be declared to be the decryption, using that key, of the public ticket number, even though that private ticket number was clearly not chosen and committed to in this way. The latter choice has the advantage of avoiding many, though not all, of the customer relations problems accompanying the first choice. For example, different lottery outcomes result depending on whether the malformed tickets are declared invalid or not. However, whatever choice is made, the existence of any such improperly formed tickets has the effect of destroying the quick verification of the lottery outcome. The lottery outcome can be quickly determined, ignoring the malformed tickets, but it is the determination that such tickets are malformed that cannot be quickly verified.

Malformed tickets might be relatively rare. Indeed, it is conceivable that many lottery runs will not contain any. Customers are not likely to submit malformed tickets since it will cost them money to do so. (Recall that we assume integrity protection of ticket purchase; so, the ticket chosen by a customer can assumed to be the ticket entered in the ticket list.) If necessary, ticket purchase can even be authenticated as in Sect. 5.1.2, as a means to reveal anyone who submits a malformed ticket. That individual can then be sanctioned by, at least, excluding him from any future lottery runs. And, assuming customer generated malformed tickets rarely occur, a vendor is not likely to choose invalid ticket numbers since this will make her lottery less appealing to the public than that of another vendor. Nonetheless, it is still possible that malformed tickets will occur enough to remove the advantage of quick verification. Obviously, the smaller and more familiar the set of ticket purchasers the lower the likelihood of malformed tickets. Thus, this design is likely to be useful for smaller scale lotteries, e.g., for choosing the winner of a door prize or for betting pools of no more than several hundred participants. Similarly, it can be useful in other situations where communal randomness is needed and sanctions can be expected to have effect or the numbers of participants are small. In [4], this design is used to provide randomness in the configuration of mix cascades. Mixes are anonymity building blocks that are usually most effective if several are used in sequence. A cascade is a static sequence of mixes. The concern in [4] is to allow the mixes to form up into cascades in such a way that no one has predictable control over which mixes are in which cascades. It is not expected that there would be more than some hundreds of mixes in a network (in the relatively near term more likely at most dozens). And, since mixes that submit malformed tickets or that do not reveal their secrets in a timely fashion are excluded from network configuration for a time, the number of malformed tickets and even the number of unrevealed secrets should be quite low.

Given registered purchasers, another way to minimize the effect both of malformed tickets and of failure to reveal private ticket numbers is to reduce the length of the delay. If one can reasonably assume a maximum degree of parallelization and enforce a corresponding minimum number of purchasers, then the individual delays can be reduced. This means that the time to verify that a ticket is malformed or to reveal a private ticket number can be commensurately reduced. One can of course also use the parallel and/or probabilistic verification speedup techniques of Sect. 2.6 to confirm that a ticket is malformed, once it has been so proven in the first place; in fact, the vendor should post the start-end pairs of parallel verification sequences for each malformed ticket once that ticket's whole computation is done. If it is possible to adequately predict the offsetting risks and advantages, this may yet be the most advantageous of the earlier designs. But, about that we can only speculate.

The problem of malformed tickets would not exist if there were an easy mechanism by which ticket purchasers could prove *verifiable recovery*, i.e., that the individual delay calculation for the ticket in question will reveal a commitment to a private ticket number. A mechanism to allow a proof of verifiable recovery was given in [3]. Obviously, the proof must not counteract the unpredictability properties of the delaying function. In fact, the mechanism goes further in that it is a zero-knowledge proof. The proof is relatively efficient for a zero-knowledge proof; however, at minimum several exchanges between prover and verifier are required, and several hundred exponentiations by the verifier are required, for the purchase of each ticket. As with the malformed ticket problem this mechanism solves, the computational and communication overhead required by the mechanism limits its scalability to lotteries with many thousands of purchasers. Unlike the malformed ticket problem, it is conceivable that the proof could ultimately be efficient enough for this approach to be feasible for large-scale lotteries. In fact, [9] recently extended the work of [3] both by allowing somewhat more efficient verification of correct commitment and by allowing temporarily secret commitment to standard (rather than "special-purpose" timed-release) signatures. The efficiency gain is based on a reuse of timelines, and we have not explored its usefulness in applications such as ours. In any case, the potential remains that many purchasers of (well-formed) tickets would not reveal their commitments after the lottery closed. If all shortcuts are revealed, the lottery outcome can be quickly computed. If few enough are revealed, the lottery outcome may take a very long time to compute (if not enough processors are available). As noted in the last paragraph, that would still need to be solved somehow for large-scale lotteries using this approach.

6.3 Evaluation

General requirements are shown essentially as they were for the previous lotteries, except that in order to guarantee fair selection, we must assume there is at least one well-chosen ticket used to compute the lottery outcome whose shortcut is not revealed during purchase, i.e., Assumption A23.

The requirements of being shortcut computable, shortcut commitment-verifiable, easily commitment-verifiable follow immediately from the lottery design.

7 Combined properties and related work

The lotteries of Sects. 4 and 5 made use of collected delays. Nonetheless, there was an individual secret that each ticket purchase implied, namely, the ticket-number input. This seemed especially important to deter spatial attacks on lotteries with run exogenous value, but it also played a role in the proofs of Sect. 4. There is a potential drawback to this in that an individual may purchase a ticket, and may even have independent proof of the purchase of *that* ticket, but if he has lost the ticket-number input, he cannot prove that the ticket is valid. On some level, this is akin to losing one's (physical) ticket in current games involving printed tickets. The answer generally is "too bad, not our problem". But, this funny status of being able to prove purchase but not validity may give rise to some awkward moments in public relations. And, one cannot simply be forgiving and generous on these occasions either, if the motivations for introducing the secret and random ticket-number input are taken seriously. A possible solution would be to make the CSBC function for the ticket-number inputs, a TSBC function. The time delay for recovering the ticket-number input can be quite a bit longer than that used for the calculation of the lottery outcome as long as it is adequately fast to still be able to collect winnings. This can actually be entirely up to the customer, who is choosing the delay based on his own expectation of carelessness or misfortune.

Various papers have appeared that make use of algorithms that require a bounded amount of computation (possibly sequential) to yield a value.

For the 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 [6]. They present solutions that depend on the difficulty of extracting square roots mod p (no known method requires fewer than about log p multiplications), on the difficulty of factoring large numbers, and other hard problems. In a similar fashion, Franklin and Malkhi [8] use repeated hashing as evidence that a certain amount of time has elapsed.

Dwork and Naor's pricing functions can be considered as algorithms that break (find the preimage of) easily computable WSBC functions. (Dwork and Naor also present a shortcut computable example.) The algorithms employed by Franklin and Malkhi also break WSBC functions, but the verification is probabilistic in nature. This is somewhat similar to our probabilistic verification of inherently sequential computations (Sect. 2.6), but using different techniques.

These papers' motivating applications are the prevention and/or detection of junk e-mail and the metering of Web clicks for advertising revenue, respectively. These have important properties that are not present in lotteries. One is that, unlike lotteries, the cost of a security breach is low: some junk mail may get through or an advertiser may be charged for a bogus click. In contrast, the value of a lottery payoff is presumably much higher and the probability of fraud or error must be proportionately lower. This is true both in the degree of secrecy and the degree of commitment in the WSBC.

There is an observation that applies both to these papers and our own work. For most applications, weak confidentiality is sometimes tolerated in practice because of a number of problems associated with strong cryptography: availability, cost (both monetary and computational), and legal or policy restrictions. Surprisingly, we see here several applications in which weak confidentiality is something not only to be grudgingly tolerated but also to be embraced as useful. This is also apparent in the password strengthening techniques of [1,13].

Related to password strengthening is the key stretching technique of [11]. Key stretching "adds a known number of bits to the expected difficulty of an exhaustive search attack". As noted earlier in Sect. 2.4, in password strengthening, the short salt is intended to allow a relatively small search by anyone who knows the password and to require a relatively large search by anyone who does not. For key stretching, there is no intent to produce a value that is recoverable with only moderate work. The intent is to produce from a short key, a key that is as resistant to some attacks (e.g., exhaustive search) as a longer one would normally be. Interestingly, one of the proposed algorithms for key stretching is to perform iterated hashings of the salted key, similar to TSBC techniques we have mentioned.

We have already noted the relation of [3,9,15,17] to the work in this paper. We further note that these involve commitment to known values, whereas our focus has been on applications where the value committed to is not known in advance by anyone. We did also present a lottery design in which the result was computed using delays on individually known commitments; however, it was subject to limitations that our other designs did not have. The collective coin flip example given in [3] is similar to the lottery scheme given in Sect. 6 (first presented in [21]) but with the addition of verifiable recovery, i.e., zero-knowledge proofs that submitted tickets are well-formed. The fair contract signing protocol in [3] is also similar to the "rational exchange" protocol given in [21]; however, with the addition of verifiable recovery, the possibility of a more generally applicable and fair (as opposed to rational) exchange is enabled.

Another approach to designing a lottery where the winner is determined by the (random) inputs from the players was presented in [7]. This paper introduces a homomorphic public-key scheme with threshold decryption. The lottery is run by combining the encrypted inputs of the players, e.g., summing the (encrypted) inputs modulo the number of players. A threshold number of servers is needed to decrypt the result and determine the number of the winning player. There have also been some cryptographic treatments of lotteries used as forms of payment mechanisms rather than to run actual lotteries (cf., e.g., [16,23]).

8 Conclusion

In this paper, we have described various properties of functions designed to control the work and/or time necessary to reveal a value—properties pertaining to the commitment, secrecy, and verification of the value. Various types of shortcuts and trapdoors may be possible if the value is known in advance of the commitment. Even when these are not possible or desired, shortcuts to probabilistic verification are possible. Unlike most previous work, we have primarily focused on functions and applications for which the committed value is not known in advance by anyone.

We used some of the functions we described in the design of several publicly verifiable lotteries. 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. Properties of specific presented lottery designs include (1) jackpot rollover and relatively short delay in calculating the lottery result, (2) authentication-free purchase and fairness of the lottery outcome independent of any size collusion set or denial-of-service for ticket purchase, and (3) quick and easy verification of the lottery outcome.

References

1. Abadi, M., Lomas, M.A., Needham, R.: Strengthening passwords. Technical note 1997-033, SRC, 1997. Published Sept. 4, 1997 (with minor revisions on Dec. 16, 1997) (1997)
2. Bellare, M., Rogoway, P.: Optimal asymmetric encryption. In: Advances in Cryptologoy—CRYPTO '94, pp. 92–111. Springer-Verlag, LNCS 950 (1995)

3. Boneh, D., Naor, M.: Timed commitments (extended abstract). In: Advances in Cryptology—CRYPTO 2000, pp. 236–254. Springer-Verlag, LNCS 1880 (2000)
4. Dingledine, R., Syverson, P.: Reliable MIX cascade networks through reputation. In: Matt Blaze (ed.) Financial Cryptography: FC 2002, 6th Interntaional Conference, Revised Papers, pp. 253–268. Springer-Verlag, LNCS 2357 (2003)
5. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. In: 23rd ACM Symposium on the Theory of Computing, pp. 542–552. Full version available from authors (1991)
6. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Technical Report CS95-20, Weizmann Institute, pp. 139–147. Preliminary version in CRYPTO '92 (1995)
7. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Frankel, Y. (ed.) Financial Cryptography: 4th International Conference, FC 2000, Proceedings, pp. 90–104. Springer-Verlag, LNCS 1962 (2001)
8. Franklin, M.K., Malkhi, D.: Auditable metering with lightweight security. In: Hirschfeld, R. (ed.) Financial Cryptography: FC '97, Proceedings, pp. 151–160. Springer-Verlag, LNCS 1318 (1998)
9. Garay, J.A., Jakobsson, M.: Timed release of standard digital signatures (extended abstract). In: Blaze, M. (ed.) Financial Cryptography: FC 2002, 6th International Conference, Revised Papers, pp. 168–182. Springer-Verlag, LNCS 2357 (2003)
10. Goldschlag, D.M., Stubblebine, S.G.: Publicly verifiable lotteries: applications of delaying functions. In: Hirschfeld, R. (ed.) Financial Cryptography: FC '98, Proceedings, pp. 214–226. Springer-Verlag, LNCS 1465 (1998)
11. Kelsey, J., Schneier, B., Hall, C., Wagner, D.: Secure applications of low-entropy keys. In: Information Security Workshop (ISW'97), Ishikawa Japan, September (1997)
12. Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. In: Proceedings of NDSS 2000, pp. 143–154 (2000)
13. Manber, U.: A simple scheme to make passwords based on one-way functions much harder to crack. Comput. Secur. **15**(2), 171–176 (1996)
14. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
15. Ralph, M.: Secure communications over insecure channels. Commun ACM **21**(4), 284–299 April (1978)
16. Rivest, R.L.: Electronic lottery tickets as micropayments. In: Hirschfeld, R. (ed.) Financial Cryptography: FC '97, Proceedings, pp. 307–314. Springer-Verlag, LNCS 1318 (1998)
17. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical Memo MIT/LCS/TR-684, MIT LCS, February (1996)
18. Roman, S.: Coding and Information Theory, volume 134 of Graduate Texts in Mathematics. Springer, New York (1992)
19. Schneier, B.: Applied Cryptography, Second Edition: Protocols, Algorithms and Source Code in C. Wiley, New York (1996)
20. Stubblebine, S., Syverson, P.: Authentic attributes with fine-grained anonymity protection. In: Frankel, Y. (ed.) Financial Cryptography: 4th International Conference, FC 2000, Proceedings, pp. 276–294. Springer-Verlag, LNCS 1962 (2001)
21. Syverson, P.: Weakly secret bit commitment: applications to lotteries and fair exchange. In: 1998 Computer Security Foundations Workshop (CSFW11), Proceedings, pp. 2–13, Rockport Massachusetts, June (1998)
22. Webster, A.F., Tavares, S.E.: On the design of s-boxes. In: Advances in Cryptology—CRYPTO '85, pp. 523–534. Springer-Verlag, LNCS 218 (1986)
23. Wheeler, D.: Transactions using bets. In: Lomas, M. (ed.) Security Protocols: 4th International Workshop, pp. 89–92 (1996)