A provably secure polynomial approximation scheme for the distributed lottery problem

(Extended abstract)

ANDREI Z. BRODER
DEC - Systems Research Center
130 Lytton Avenue
Palo Alto, CA. 94301

Abstract. It was shown by Michael Rabin that a sequence of random 0-1 values ("coin tosses"), prepared and distributed by a trusted "dealer," can be used to achieve Byzantine agreement in constant expected time in a network of n processors, despite the fact that some of the processors might behave in a malicious and collusive manner. A natural question is whether it is possible to generate these tosses uniformly at random within the network. The subject of this paper is a constructible, provably-secure, fully-polynomial approximation scheme to obtain such coin tosses. In other words, if the bias of the coin is allowed to be ϵ, the length of the messages required by this scheme is polynomial in n and $1/\epsilon$, and it can be shown that breaking the cryptographic devices used by this scheme is equivalent to solving a well-known hard number theoretic problem. Previous algorithms for such distributed coin tosses were provably secure and constructible, but exponential in the size the network (Yao, Awerbuch & *al.*), provably secure and and polynomial, but not constructible (Yao), and polynomial, constructible, but not provably secure in the above sense (Broder & Dolev).

1. Introduction

The subject of this paper is a protocol for distributed lottery agreement (DLA), that is defined as agreement on an unbiased random bit in a network of n processors, out of which up to t might be faulty and malicious. The interest in this problem was sparked by the work of Rabin ([Rabin83]) who showed that in such an environment it is possible to achieve agreement on a sender's value (Byzantine agreement[1]) in constant expected time, provided that at each round all processors have access to a common random bit,

[1] See next section for definitions.

not known in advance. It is natural to seek ways of generating this random bit *inside* the network, and algorithms achieving this goal were devised by Yao [Yao83], Broder and Dolev [BD84], Awerbuch, Chor, Micali, and Goldwasser [ACGM85], and Awerbuch, Blum, Chor, Micali, and Goldwasser. A lower bound for the number of rounds required for deterministic lottery agreement was proved in [BD84]. Other lower bounds are implied from the results in [KY85], [CSM85], and [FLM85].

Two immediate applications for such a distributed coin flipping protocol are the problem of choosing a leader at random and the fair allocation of resources in a network, where some of the processors might be faulty. It was also shown by Bracha [Bracha85] that such a protocol can be used as a subroutine to achieve Byzantine agreement in $O(\log n)$ expected time.

Although at first sight, this problem might appear simpler than agreement on a sender's value, it is not. For example if we assume that the only possible type of communication is *reliable broadcast,* (i.e. any message sent by a certain processor is "heard" by all the other processors), and if we assume that the majority of the processors are correct, then the majority of processors agree on every sent bit. However the majority vote on a random bit will be heavily biased by the vote of the faulty processors. The parity vote is also inadequate because a faulty processor might wait till it knows all the other votes before it casts its own vote, thus fixing the outcome. Another tempting solution is a trivial generalizations of the "telephone coin flipping" protocol [Blum82]; it fails for a subtler reason: a faulty processor might abandon the protocol if it perceives that the outcome will be unfavorable. The correct processors can restart the protocol, but the final outcome will be biased.

Yao's algorithm for DLA works if the number of faulty processors, t, is less than $n/3$. If $t < (1-\epsilon)n/3$ then the length of messages is polynomial in n, otherwise it is exponential. Yao's very elegant solution is based on the partition of the set of processors into a suitable family of subsets. The existence of such families can be proven by probabilistic methods, but their actual computation is apparently quite difficult. However if we allow exponentially (in n) long messages the algorithm is very simple: the n processors are partitioned into all possible $\binom{n}{2t+1}$ subsets, and each subset independently agrees on a 0-1 value, by majority vote; the DLA value is the exclusive or of all the subset values. Since one subset is composed only of correct processors, its associated value will be unbiased and hence the DLA value will be unbiased, provided that the faulty processors can not eavesdrop to the communication between correct processors. To allow for the danger of eavesdropping, it is possible to use the probabilistic encryption technique of Goldwasser and Micali [GM84] for all messages, and the resulting scheme is provably secure if the Quadratic Residuosity problem is hard.

The algorithm proposed by Broder and Dolev [BD84] is based on the idea of *shared secrets* (see section 4.2). It is constructible, requires a polynomial number of messages, but makes the assumption that a polynomial number of encryptions of a certain message, under different keys, using the RSA scheme [RSA78] with large exponent is secure. This assumption is an extra requirement beyond the presumption that RSA itself is hard to break, and it is not satisfied by RSA with low exponent [Hastad84].

The algorithm in [ACGM85] is constructible and uses a very ingenious novel cryptographic scheme for secret sharing to achieve provable security in the above sense. However it requires exponentially long messages and from the point of view of the DLA problem, does not have substantial advantages over Yao's constructible scheme. The algorithm in [ABCGM85] is constructible, provably secure, and also requires exponentially long messages. It works for $t < n/2$ versus $t < n/3$ for Yao's method.

From the above discussion, it should be clear that the ideal algorithm for DLA requires polynomially long messages, is constructible, and is provably secure. Such an algorithm has not yet been discovered, but in this paper we shall describe a fully polynomial scheme to approximate it: for any ϵ we show how to construct a DLA protocol that requires only messages of length polynomial in $1/\epsilon$ and n and produces a coin with maximum bias ϵ. The number of rounds is linear in n and independent of ϵ. An algorithm of this type is also described in [ABCGM85]; compared to the solution here, it has the disadvantage that number of rounds grows polynomially with $1/\epsilon$.

2. Model and terminology

The model used here is the standard model for synchronous Byzantine agreement protocols. We assume a fully interconnected network of synchronous processors. Each pair of processors is connected via a private line. (We do not assume reliable broadcast, unless otherwise specified.) Some processors might be faulty and they can behave in a malicious and collusive manner, but the communication between processors is perfectly reliable. These conditions allow us to assume that the computation proceeds in *rounds*, (or *phases*). During a round, each processor first sends a set of messages and then receives messages addressed to it by other processors. Thus, for correct processors, messages received during a phase do not affect messages sent during the same phase. Of course faulty processors might not abide by this rule.

The protocol described in this paper works both for broadcast communication networks, such as ETHERNET, and private communication networks. No secrecy assumption is made based on the privacy of communication, and hence the protocol is not vulnerable to passive eavesdropping.

A computational primitive in such a network is *Byzantine agreement* (BA), defined as follows: All the n processors in the network receive an initial value from a certain processor called the *sender*. Byzantine agreement is reached iff

1. All correct processors agree on the same value y.
2. If the sender sent the same value y' to every correct processor then $y = y'$.

A protocol for BA is called *t-resilient* if it is valid for any number of faulty processors not exceeding t, and it is called *authenticated* if it assumes that messages are signed with unforgeable signatures.

The following are known results [PSL80, LSP82, DS83]. (For additional references see [Fischer83].)

1. There is a t-resilient, authenticated, synchronous protocol that achieves BA for $t \leq n-1$ in $t+1$ rounds.

2. There is a t-resilient, non-authenticated, synchronous protocol that achieves BA for $t < n/3$ in $t+1$ rounds. (These bounds are tight.)

   Rabin's result, mentioned earlier, is that

3. There is a t-resilient, non-authenticated, synchronous protocol that achieves BA for $t < n/4$, in constant expected time (4 rounds), provided that at each round all processors have access to a common random bit, not known in advance [Rabin83].

3. Distributed lottery agreements (DLA)

A distributed lottery agreement belongs to the class of randomized agreement protocols, where each processor, i, has access to an unbiased and independent coin, C_i. The sequence of all the coin flips done by processor i (the "flip-vector") is denoted $\bar{C}_i$. For a precise formal description of randomized agreement protocols see [BD84].

A binary distributed lottery agreement (2-DLA) is achieved iff

1. Starting from any legal initial configuration, all correct processors decide on the same value $L \in \{0,1\}$.

2. For every legal initial configuration

$$\mathbf{Pr}(L=0) = \mathbf{Pr}(L=1) = 1/2,$$

   where the probability space is the probability distribution on the flip-vectors C_i for the correct processors. (In other words, whenever a new agreement protocol is started in any legal initial state, for any strategy of the faulty processors, the agreement value is equally likely to be 0 or 1.) We allow each $\bar{C}_i$ to have a different probability distribution, but they must be independent random variables.

Given a positive constant ϵ, called *permitted bias*, an ϵ-approximation of a distributed lottery agreement DLA(ϵ) is achieved iff

1. Starting from any legal initial configuration, all correct processors decide on the same value $L \in \{0,1\}$.

2. For every legal initial configuration

$$\left|\mathbf{Pr}(L=0) - \frac{1}{2}\right| \leq \epsilon,$$

   where the probability space is the probability distribution on the flip-vectors C_i for the correct processors.

4. Preliminaries

We shall make use of two cryptographic devices that are described below.

4.1. Probabilistic public key cryptosystems

For the purposes of this paper we can describe a probabilistic public key cryptosystem (PPKC) as consisting of two functions: an encryption function, E, which is public, and a decryption function, D, which is private. To encrypt a message x, an user chooses a secret random string R and sends $E(x,R)$. Hence any message, x, has many possible encryptions depending on the random string R used to encrypt it. However for every R, $D\big(E(x,R)\big) = x$, that is, the decryption function is unambiguos.

It is assumed that without knowing any of x, R, and D, the chance of guessing whether $x = y$ is increased only by an exponentially small amount from the knowledge of $E(x,R)$, if the cryptanalyst has only polynomially bound resources. In particular, if x is uniformly distributed over the set $\{0,1\}$, then from the knowledge of $E(x,R)$ it is not possible to guess x with probability more than $1/2$ plus an exponentially small amount.

For a more formal definition of PPKCs see [GC84] and for implementations see [GC84] and [BG84].

Theorem 1. *Let the functions $f_1, f_2, \ldots, f_s$ and g be all computable in polynomial time and $E_1, E_2, \ldots, E_s$ be encryption keys for PPKC's. Let x be a binary variable. Without the knowledge of any of x, $R_1, \ldots, R_s$, and $D_1, \ldots, D_s$, from the knowledge of $E_1\big(f_1(x), R_1\big), \ldots, E_s\big(f_s(x), R_s\big)$ the chance of guessing whether $g(x) = y$ is increased only by an exponentially small amount, if the cryptanalyst has only polynomially bound computational resources.*

Proof: Omitted. ∎

4.2. Sharing a secret

We want to divide a secret in such a way that from the knowledge of any subset of $k+1$ shares we can retrieve the secret, but the knowledge of only k shares offers no information about the secret. The following solution is due to Adi Shamir [Shamir79]

1. Agree on a prime number p (public).
2. Choose a random polynomial P over $GF[p]$ of degree k, $P = a_k x^k + \cdots + a_1 x + a_0$. Let a_0 be the secret.
3. A share of the secret is a pair $\big(x_i, P(x_i)\big)$, with $x_i \neq 0$.

The shares of the secret are pairs of the form $\big(x, P(x)\big)$. For any set of $k+1$ pairs $\big(x_i, P(x_i)\big)$, there is a unique interpolating polynomial, but for a set of k pairs, there are p interpolating polynomials, and therefore a_0 is completely undetermined.

Remark that the issuer of shares can issue any number of them. However it can also issue duplicate shares or inconsistent shares, in other words it might be the case that there is no polynomial of degree k that interpolates through all the shares, or that there are subsets of $k+1$ shares that do not uniquely determine a polynomial.

5. A protocol for ϵ-approximate DLA

We make the following assumptions:

1. The number of faulty processors is less or equal t, and there are at least $2t+1$ processors participating in the protocol.
2. Before the start of the protocol, $t+1$ processors, $A_1, A_2, \ldots, A_{t+1}$, are designated as "players" and $2t+1$ processors, $B_1, B_2, \ldots, B_{2t+1}$, are designated as "trustees." There is general agreement which processors are players and which processors are trustees. It is possible for a processor to be both a player and a trustee. (In fact this condition can be relaxed, at the expense of extra messages: all processors can be designated to be both players and trustees.)
3. Each trustee B_j can compute a set of probabilistic public key cryptosystem $\{(E_{i,j}, D_{i,j})\}$ as described in Section 4.1.
4. Before the start of the protocol there is general agreement on a prime $p > n$.

The protocol is composed of 5 parts described below; during each part Byzantine agreements on certain values are run concurrently. When we say that a processor G *makes public* a value v, we mean that a Byzantine agreement is run on the value v with the participation of all the processors (i.e. not only the players and the trustees). If the result of this Byzantine agreement is v = faulty or the result is an unacceptable (as defined by the protocol) value, then we say that G is *known to be faulty* after the Byzantine agreement. The processors not known to be faulty after a certain stage, are called *participant* after that stage. The protocol describes what correct processors should do. Faulty processors might of course not abide by it.

Most of the techniques used are based on the protocol PFLIP from [BD84], but the "verification" requirements that are the source of extra cryptographic requirements, are removed.

The protocol is described as depending on two parameters, m and r. Correct choices for these parameters to ensure that the bias is less than ϵ will be described in Theorem 12.

In the following description the indices are used as follows:

i	is used for "players,"	$1 \leq i \leq t+1$;
j	is used for "trustees,"	$1 \leq j \leq 2t+1$;
k	is used for "secrets,"	$1 \leq k \leq m$;
l	is used for renumbered secrets,	$1 \leq l \leq m - nr$.

Protocol PROBFLIP(m, r)

Part 1. Each trustee B_j, $1 \leq j \leq 2t+1$ computes a set of probabilistic public key cryptosystem $\{(E_{i,j}, D_{i,j})\}$, and makes public the set $\{E_{i,j}\}$. The set of all the participant trustees after this point is denoted PB_1.

Part 2. Each player A_i, $1 \leq i \leq t+1$, choses uniformly at random m independent values ("secrets") $C_{i,k} \in \{0,1\}$ and for each such value, choses uniformly at random a polynomial $P_{i,k}$ of degree t over the field GF[p], subject to the condition that its free term satisfies the condition $P_{i,k}(0) \equiv C_{i,k} \pmod 2$. For all $j \in PB$ the player A_i makes public m quadruples $(i, j, k, X_{i,j,k})$ with $X_{i,j,k} = E_{i,j}(P_{i,k}(j), R_{i,j,k})$. In other words, each quadruple represents the j'th share of the k's secret of the player number i, encrypted with the key of trustee number j using a random string $R_{i,j,k}$. The set of all the participant players after this part is denoted PA_2.

Part 3. Each trustee, B_j, makes public for each player $A_i \in PA_2$, a set of r *disclosure requests*, $REQ(i,j)$, each request being a random number between 1 and m. The set of all trustees not known to be faulty after this part is denoted PB_3.

Part 4. Each player, A_i, makes public for each request $k \in \bigcup_{j \in PB_3} REQ(i,j)$, all the shares $P_{i,k}(j)$ of its secret k, and all the random strings $R_{i,j,k}$ it used to encrypt the shares of the secret k. Hence at this point, if A_i obeys the protocol, it is possible to verify that all the shares of the k'th secret sent by A_i were consistent. (By design, it is impossible to send duplicate shares.) The set of participant players after Part 4 is denoted PA_4. (These players made public all the requested secrets and strings, and for each requested secret, all its shares were consistent.)

Part 5. Each trustee B_j makes public its decryption keys $\{D_{i,j}\}$. The set of trustees participant after Part 5 (this implies their decryption key were correct), is denoted PB_5.

□

At the end of the protocol each processor uses the following procedure to decide the lottery value L.

Procedure DECIDE

1. Let S be the set of the $t+1$ smallest index trustees in PB_5. This set is well defined because there are at least $t+1$ correct trustees.
2. For each $i \in PA_4$ there are at least $m - nr$ undisclosed secrets. To simplify notation, let's assume that these secrets are $C_{i,1}, C_{i,2}, \ldots, C_{i,m-nr}$. (If not, sort the the secrets in increasing order of the second index, and renumber them $1, \ldots, m-nr$. Discard extra secrets.) For each $i \in PA_4$ and for $l = 1, \ldots, n - mr$, determine the unique polynomial of degree t over $GF[p]$, $\tilde{P}_{i,l}$, that satisfies the system of $t+1$ linear equations

$$\tilde{P}_{i,l}(j) = D_{i,j}(X_{i,j,l}) \bmod p \qquad j \in S.$$

3. For $l = 1, \ldots, n - mr$, define the *global coin* G_l by

   (N.B. By the convention made above, G_l is the sum mod 2, of the l'th undisclosed secrets of the players participant after Part 4.)

4. The final lottery value is

$$\underset{1 \le l \le m-nr}{\mathbf{MAJ}} \{G_l\},$$

   where **MAJ** is majority function.

□

6. Proof of the protocol

Very informally the main idea of the proof is the following: if a faulty player makes public only consistent shares of a certain secrets, then that secret (as perceived by the correct processors) can not be modified after learning about the correct player's secrets. However if the faulty player made public inconsistent shares of a certain secret, then other faulty processors can modify the secret (as perceived by the correct processors) by choosing whether to be or not to be in PB_5. We shall show that any player that makes public too many inconsistent secrets, will be almost surely "found out" in Part 3 or 4. As a result most of the global coins G_l will be unbiased and the lottery value, L, will be almost unbiased.

Notice that we do not assume that an adversary must decide before the start of the protocol which processors will be faulty. The adversary can make up to t processors faulty, *during* the execution of the algorithm, with total knowledge of the global state. However the behaviour of the faulty processors is restricted; they can use only the knowledge they received via the protocol and do not know the global state. (For a a more formal description of these concepts, in terms of the allowed transition functions, see [BD84].)

What follows is only a sketch of the formal proof, with most of the details left for the full paper.

Lemma 2. *After the protocol* PROBFLIP *and the decision procedure* DECIDE, *for each* $i \in PA_4$, *all correct processors agree on* $\tilde{P}_{i,l}$ *for* $l = 1, \ldots, m - nr$. *Furthermore, if* i *is a correct processor then* $\tilde{P}_{i,l} = P_{i,l}$.

Proof: First note that by the definition of the protocol all correct processors agree on the sets PA_4 and S, and for every $i \in PA_4$ and every $j \in S$ they agree on $X_{i,j,l}$ and $D_{i,j}$. Hence all the correct processors solve exactly the same system of linear equations to determine $\tilde{P}_{i,l}$. If i is a correct processor then all the shares it sent in Part 2 are consistent and therefore $\tilde{P}_{i,l} = P_{i,l}$. ∎

Corollary 3. *After the protocol* PROBFLIP *and the decision procedure* DECIDE, *all correct processors agree on* L. ∎

Lemma 4. *If all the shares of the l'th secret published by player A_i in part of* PROBFLIP *were consistent, then the faulty processors can not influence the value $\tilde{P}_{i,l}(0)$ that the correct processors compute after part 5 of* PROBFLIP.

Proof: Obvious from the definition of the protocol. ∎

Lemma 5. *After the end of part 4 of the protocol* PROBFLIP *if A_i is a correct processor then no faulty processor can guess $P_{i,l}(0)$ mod 2 with probability greater than $1/2$ plus an exponentially small amount.*

Proof: In the worst case the faulty processors know, in cleartext, t interpolating pairs for the polynomial $P_{i,l}$, corresponding to t faulty participant trustees. However $P_{i,l}$ has degree $t+1$. Let $x = P_{i,l}(0)$. By construction, x mod 2 is uniformly distributed over the set $\{0,1\}$. Knowing the other interpolating pairs made public by player A_i, means knowing, for each correct trustee B_j, the value $E_{i,j}(\alpha x + \beta, R_{i,j,l})$, where α and β are some known values depending on i, j, and l and $R_{i,j,l}$ is an unknown random string. By Theorem 1, no faulty processor can guess x mod 2 with probability greater than $1/2$ plus an exponentially small amount. ∎

Lemma 6. *If A_i is a correct processor, then the secrets distributed by the faulty processors are independent of the secrets distributed by A_i.*

Proof: Sketch. Assume the contrary. Let B_j be any correct trustee. If the faulty processor $A_{i'}$ can distribute a secret that depends on A_i's secret, $C_{i,k}$, it means that $A_{i'}$ can, given the arbitrary encryption function $E_{i',j}$, produce the string $E_{i',j}(y)$, where y depends on $C_{i,k}$. If the faulty processors can produce such strings for arbitrary encryption keys, they can produce these strings for keys with known decryption, and therefore can produce strings y that depend on $C_{i,k}$. That means that the faulty processors broke the cryptosystem that protected $C_{i,k}$.

Lemma 7. *Let $X_1, X_2, \ldots, X_s \in \{0,1\}$ be random binary variables. If there exists an i such that X_i is uniformly distributed over the set $\{0,1\}$ and for all $j \neq i$, $1 \leq j \leq s$, X_i and X_j are independent then the random variable $L = \left(\sum_j X_j\right)$ mod 2 is uniformly distributed over the set $\{0,1\}$.*

Proof: Elementary, using conditional probabilities. ∎

Lemma 8. *If for a certain l, for all $i \in PA_4$, the secrets $P_{i,l}$ were consistent, then G_l is uniformly distributed over the set $\{0,1\}$, where the probability space is the set of choices made by the correct processors.*

Proof: Combine lemmas 4, 5, and 7 using the fact that there is at least one correct player. ∎

Lemma 9. *The probability that a there exists a faulty processor that made public more than b inconsistent secrets in Part 2 of* PROBFLIP *and is included in PA_4 (that is, it is not discarded in Part 3 or Part 4) is no more than*

$$t\left(1-\frac{b}{m}\right)^{(t+1)r} < te^{-btr/m},$$

where the probability space is the set of choices made by the correct processors.

Proof: Omitted. ∎

Lemma 10. *Let $X_1, X_2, \ldots, X_s \in \{0,1\}$ be random binary variables. If at least $s-\delta$ of them are independent, and uniformly distributed over the set $\{0,1\}$, then*

$$\left|\Pr\left(\underset{1\le i\le s}{\mathrm{MAJ}}\{X_i\}=0\right)-\frac{1}{2}\right| \le \frac{2\delta}{\sqrt{s}}.$$

Proof: Bound the central terms of the binomial distribution. The details are left for the full paper. ∎

Lemma 11. *If each faulty processor makes public at most b inconsistent secrets, than the number of biased global coins is at most bt.*

Proof: Immediate, from lemma 8. ∎

Lemma 12. *The protocol* PROBFLIP(m,r) *and the decision procedure* DECIDE *ensure that all correct processors agree on a lottery value L such that for any value b*

$$\left|\Pr(L=0)-\frac{1}{2}\right| \le te^{-btr/m}+\frac{2bt}{\sqrt{m-nr}}.$$

Proof: By lemma 9, the first term represents the probability that there exists a faulty processor that made public more than b inconsistent secrets. If this is not the case, then we apply Lemma 11 and Lemma 10. ∎

Theorem 13. *For any positive constant ϵ, the protocol* PROBFLIP(m,r) *and the decision procedure* DECIDE *with $m = 17^3 n^2 \left(\log(2t/\epsilon)\right)^2/\epsilon^2$ and $r = m/(17n)$, ensure that all correct processors agree on a lottery value L such that*

$$\left|\Pr(L=0)-\frac{1}{2}\right| \le \epsilon$$

where the probability space is the set of choices made by the correct processors, except for an exponentially small probability.

Proof: Denote $\lambda = 17\ln(2t/\epsilon)$. With this notation $r = n\lambda^2/\epsilon^2$, and $m = 17n^2\lambda^2/\epsilon^2$. Set $b = n\lambda/t$. Then

$$te^{-btr/m} = te^{-\lambda/17} = \frac{\epsilon}{2}$$

and

$$\frac{2bt}{\sqrt{m-nr}} = \frac{2n\lambda}{\sqrt{16n^2\lambda^2/\epsilon^2}} = \frac{\epsilon}{2}.$$

Hence by Lemma 12

$$\left|\mathbf{Pr}(L=0) - \frac{1}{2}\right| \leq \epsilon.$$

∎

Theorem 14. *The protocol* PROBFLIP *runs in* $5t+5$ *rounds.*

Proof: Each part can be organized as some number of parallel Byzantine agreements. ∎

Theorem 15. *The total length of the messages used by the protocol* PROBFLIP *is polynomial in* n *and* $1/\epsilon$.

Proof: It is clear that the total number of Byzantine agreements used is polynomial in n and m. The only question is how large are the values subject to agreement. The answer depends on the underlying PPKSs used. Assuming computational capability polynomial in n and $1/\epsilon$, it is enough to use keys that are exponential in n and $1/\epsilon$, and therefore the messages encrypted under such keys have length polynomial in n and $1/\epsilon$. ∎

The protocol PROBFLIP shares with the protocol PFLIP of [BD84] several properties that might be useful in certain implementations and generalizations:

- If several DLAs are run in a network, the first part of the second DLA can be superposed on the last part of the first DLA and so on. Also during part 1 of a given DLA it is possible to agree for each trustee on several encryption keys to be used in subsequent DLA's.
- During the protocol, the only usage for Byzantine agreement is to simulate reliable broadcast, and therefore if reliable broadcast is "built-in," then each Byzantine agreement reduces to one communication round. For similar reasons, passive eavesdropping by the faulty processors is irrelevant.
- One can use any subprotocol for Byzantine agreement, including randomized protocols, as long as the subprotocol ensures agreement on the termination of each part.
- For Bracha's $O(\log n)$ expected time Byzantine agreement algorithm, one needs $\epsilon = O(1/n)$ and hence one can use PROBFLIP with polynomially long messages.

7. Conclusions and open problems

The distributed lottery problem seems to gain importance in the theory of distributed systems, as it becomes increasingly clear that randomized distributed algorithms are in

many occasions more powerful than deterministic algorithms, and in some cases the only type of algorithm possible. (For example [Bracha85], or [BenOr83] vs. [FLP83])

Although we know from the work of Yao that an ideal (i.e. polynomial and cryptographically secure) algorithm for DLA exists, we do not know yet how to construct it. At least four alternatives avenues can be proposed now:

1. Find an explicit partition as required by Yao's algorithm.
2. Prove that a small set of encryptions of a message using RSA schemes with high exponent is as secure as one single encryption, or find another deterministic public key cryptosystem with this kind of property and plug it in the [BD84] protocol.
3. Use the idea of testing processors for consistency as done in this paper in a new manner and get only an exponentially small error with only polynomially many tests and secrets.
4. Use the cryptographic schemes in [ACGM85] but somehow avoid the need for exponentially long (double exponentially big) encryption keys.

The ideal security proof for all cryptographic protocols, is a simulation proof – we show that if it is possible to break the protocol, we can use the broken protocol as a subroutine to solve an intractable problem. The proof presented here is not a simulation proof, but a proof that attempts to identify all possible "dangers" (a task for which there is no formalism) and show that they do not compromise the protocol. For each such "danger" breaking the protocol is equivalent with solving a hard problem; unfortunately, there might be unforeseen dangers, and in fact such was the case with an earlier version of this algorithm. In future joint work with Benny Chor and Silvio Micali we propose to present a simulation proof of a variant of this protocol.

Acknowledgement

I am greatly indebted to Benny Chor for pointing out an overlooked security problem (the trustees *cannot* use the same key for every player) in an earlier version of this algorithm, and for several useful discussions.

References

[ACGM85] S. Awerbuch, B. Chor, S. Goldwasser, and S. Micali, "Provably secure coin flip in a Byzantine environment," unpublished manuscript, 1985.

[ABCGM85] S. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali, "How to implement Bracha's $O(\log n)$ Byzantine agreement algorithm," unpublished manuscript, 1985.

[BD84] A. Z. Broder and D. Dolev, "Coin flipping in many pockets," *Proceedings of the 25-th Symposium on Foundations of Computer Science*, Singer Island, Florida, 1984, 157–170.

[BenOr83] M. Ben-Or, "Another advantage of free choice: completely asynchronous agreement protocols," *Proceedings of the 2-nd Annual ACM Symposium on Principles of Distributed Computing*, Montreal, 1983, 27–30.

[BG84] M. Blum and S. Goldwasser, "An efficient probabilistic public key cryptosystem which hides all partial information," presented at CRYPTO 1984.

[Blum82] M. Blum, "Coin flipping by telephone – a protocol for solving impossible problems," *Spring COMPCON conference*, 1982, 133–137.

[Bracha85] G. Bracha, "A randomized Byzantine agreement with an $O(\log n)$ expected rounds," *Proceedings of the 17th ACM Symposium on Theory of Computing*, Providence, 1985, 316–326.

[CMS85] B. Chor, M. Merritt and D. Shmoys, "Simple constant time consensus protocols in realistic failure models," in these proceedings.

[DS83] D. Dolev and H. R. Strong, "Authenticated algorithms for Byzantine agreement," *SIAM J. Comput.*, **12**(1983), 656–666.

[Fischer83] M. J. Fischer, "The consensus problem in unreliable distributed systems (A brief survey)," Technical Report, Department of Computer Science, Yale University, 1983.

[FLP83] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Proceedings of the 2-nd Annual ACM Symposium on the Principles of Database Systems*, 1983, 1–7.

[GM84] S. Goldwasser and S. Micali, "Probabilistic encryption," *JCSS*, **28**(1984), 270–299.

[Hastad84] J. Hastad, "On using RSA with low exponent in a public key network," unpublished manuscript, 1984.

[KY85] A. R. Karlin and A. C. Yao, "Probabilistic lower bounds for Byzantine agreement," unpublished manuscript, 1985.

[LSP82] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM TOPLAS*, **4**(1982), 382-401.

[PSL80] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, **27**(1980), 228-234.

[Rabin83] M. O. Rabin, "Randomized Byzantine generals," *Proc. of the 24-th Annual Symposium on Foundation of Computer Science*, 1983, 403–409.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Comm. ACM*, **21**(1978), 120–126.

[Shamir79] A. Shamir, "How to share a secret," *Comm. ACM*, **22**(1979), 612–613.

[Yao83] A. C. Yao, "On the succession problem for Byzantine generals," Technical report, Computer Science Department, Stanford University, to appear.