Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation

(Extended Abstract)

Michael Ben-Or*
Hebrew University

Shafi Goldwasser†
MIT

Avi Wigderson‡
Hebrew University

Abstract

Every function of n inputs can be efficiently computed by a complete network of n processors in such a way that:

1. If no faults occur, no set of size $t < n/2$ of players gets any additional information (other than the function value),

2. Even if Byzantine faults are allowed, no set of size $t < n/3$ can either disrupt the computation or get additional information.

Furthermore, the above bounds on t are tight!

Introduction

The rapid development of distributed systems raised the natural question of what tasks can be performed by them (especially when faults occur). A large body of literature over the past ten years addressed this question. There are two approaches to this question, depending on whether a limit on the computational power of processors is assumed or not.

The cryptographic approach, inaugurated by Diffie and Hellman [DH], assumes the players are computationally bounded, and further assumes the existence of certain (one-way) functions, that can be computed but not inverted by the player.

This simple assumption was postulated in [DH] in order to achieve the basic task of secure message exchange between two of the processors, but turned out to be universal! In subsequent years ingenious protocols based on the same assumption were given for increasingly harder tasks such as contract signing, secret exchange, joint coin flipping, voting and playing Poker. These results culminated, through the definition of zero-knowledge proofs [GMR], their existence for NP-complete problems [GMW1] in completeness theorems for two-party [Y1] and multi-party [GMW2] cryptographic distributed computation. In particular the results of Goldreich, Micali and Wigdesrson in [GMW2] were the main inspiration to our work. They show, that if (non-uniform) one way functions exist then every (probabilistic) function of n inputs can be computed by n computationally bounded processors in such a way that: (1) If no faults occur, no subset of the players can compute any additional information, and (2) Even if Byzantine faults are allowed, no set of size $t < n/2$ can either disrupt the computation or compute additional information.

The non-Cryptographic (or information-theoretic) approach does not limit the computational power of the processors. Here, the notion of privacy is much stronger - for a piece of data to be unknown to a set of players it does not suffice that they cannot compute it within a certain time bound from what they know, but simply that it cannot be computed at all!

To facilitate the basic primitive of secret message exchange between a pair of players, we have secure channels. (For an excellent source of results and problems in the case no secure channels exist, see [BL]). Unlike the cryptographic case, very little was known about the capabilities of this model. Two main basic problems were studied and solved (in the synchronous case): Byzantine agreement [LPS,DS,...] and collective coin flipping [Y2].

*Supported by Alon Fellowship.

†Supported in part by NSF grant 865727-CCR, ARO grant DAAL03-86-K-017, and US-Israel BSF grant 86-00301, Jerusalem, Israel.

‡Supported by Alon Fellowship.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specfic permission.

© 1988 ACM-0-89791-264-0/88/0005/0001 $1.50

This paper provides a full understanding of the power and limits of this model, by proving a few completeness theorems. Comparing these results to the cryptographic case of [GMW2], one gets the impression that one-way functions are "more powerful" than secure channels. This should not be surprising, if one considers the case of $n = 2$. Clearly, here a secure channel is useless, and indeed two (non-faulty) players can compute the OR function of their bits using cryptography, while the reader can convince herself (it will be proven later) that any protocol will leak information in the information-theoretic sense. The lower bounds we provide show that the same phenomenon is true for any value of n. A similar situation arises in the Byzantine case where, using cryptography one can allow $t < n/2$ faulty players, but in the non-Cryptographic case one must have $t < n/3$.

As happened in the cryptographic case, the protocols are based on a new method for computing with shared secrets. Our constructions are based on Algebraic Coding Theory, particularly the use of generalized BCH codes.

It is important to stress here that our main protocols require only a polynomial amount of work from the players. (In fact, they are efficient enough to be practical!). Putting no bound on the computational power serves only to allow the most stringent definition of privacy and the most liberal definition of faultiness, both of which we can handle.

Essentially the same results we obtain here were independently discovered by Chaum, Crepeau and Damgard [CCD]. We briefly point out the small differences of this work from ours. The simple case of no faults is almost identical. Their solution in the case of Byzantine faults is elementary and requires no error correcting codes. The error correction is achieved using a clever scheme of zero knowledge proofs. This has two consequences: They have to allow an exponentially small error probability for both correctness and privacy (we can guarantee them with no errors), and the frequent zero knowledge proofs increase the complexity of their protocols. In the solution of [CCD] the simulation is of Boolean operations while our solution allows direct simulation of arithmetic operations in large finite fields. Thus, for example, computing the product of two n bit numbers using [CCD] calls for $O(\log n)$ communication rounds. This can be done in $O(1)$ rounds using our solution.

We mention that the above results already found application in the new, constant expected number of rounds protocol for Byzantine agreement of Feldman and Micali [FM].

We proceed to define the model, state the results and prove them. In the full paper we mention generalizations and extensions of our results to other tasks (playing games rather than computing functions), to other model parameters (synchrony, communication networks) and other complexity measures (number of rounds).

Definitions and Results

For this abstract, we define the model and state the results on an intuitive level. Since even the formal definition of the notions of privacy and resiliency are nontrivial, we give them explicitly in an appendix.

The model of computation is a complete synchronous network of n processors. The pairwise communication channels between players are secure, i.e. they cannot be read or tempered with by other players. In one round of computation each of the players can do an arbitrary amount of local computation, send a message to each of the players, and read all messages that were sent to it at this round.

We shall be interested in the computational power of this model when imposing privacy and fault tolerance requirements. For simplicity, we restrict ourselves to the computation of (probabilistic) functions f from n inputs to n outputs. We assume that player i holds the i-th input at the start of computation, and should obtain the i-th output at the end, but nothing else.

A protocol for computing a function is a specification of n programs, one for each of the players. We distinguish two kinds of faults: "Gossip" and "Byzantine". In the first, faulty processors send messages according to their predetermined programs, but try to learn as much as they can by sharing the information they received. In the second, they can use totally different programs, collaborating to acquire more information of even sabotage the computation.

A protocol is *t-private* if any set of at most t players cannot compute after the protocol more then they could jointly compute solely from their set of private inputs and outputs.

A protocol is *t-resilient* if no set of t or less players can influence the correctness of the outputs of the remaining players. For this to make sense, the function definition should be extended to specify what it is if some players neglect to give their inputs or are caught cheating (see appendix).

We can now state the main results of this paper.

Theorem 1: *For every (probabilistic) function f and $t < n/2$ there exists a t-private protocol.*

Theorem 2: *There are functions for which there are no $n/2$-private protocols.*

Theorem 3: *For every probabilistic function and every $t < n/3$ there exists a protocol that is both t-resilient and t-private.*

Theorem 4: *There are functions for which there is no $n/3$-resilient protocol.*

Proof of Theorem 1

Let $P_0, \ldots, P_{n-1}$ be a set of players, and let $n \geq 2t+1$. Let F be the function which this set of players wants to compute t-privately, where each player holds some input variables to the function F. Let E be some fixed finite field E, with $|E| > n$. Without loss of generality we may assume that all inputs are elements from E and that F is some polynomial (in the input variables) over E, and that we are given some arithmetic circuit computing $|F|$, using the operations $+, \times$ and constants from E.

To simplify our explanation we divide the computation into three stages.

Stage I: The input stage, where each player will enter his input variables to the computation using a secret sharing procedure.

Stage II: The computation stage, where the players will simulate the circuit computing F, gate by gate, keeping the value of each computed gate as secret shared by all players.

Stage III: The final stage, where the secret shares of the final value of F are revealed to one or all of the players.

Stages I and III are very simple and we describe them below, and delay the details of the computation stage to the next section.

The input stage

Let $\alpha_0, \ldots, \alpha_{n-1}$ be some n distinct non zero points in our field E. (This is why we need $|E| > n$.) Each player holding some input $s \in E$, introduces the input to the computation by selecting t random elements $a_i \in E$, for $i = 1, \ldots, t$, setting

$$f(x) = s + a_1 x + \cdots + a_t x^t$$

and sending to each player P_i the value $s_i = f(\alpha_i)$.

As in Shamir's [Sh] secret sharing scheme, the sequence $(s_0, \ldots, s_{n-1})$ is a sequence of t-wise independent random variables uniformly distributed over E, thus the value of the input is completely independent from the shares $\{s_i\}$ that are given to any set of t player that does not include the player holding the secret.

The final stage

To keep the t-privacy condition, we will make sure that the set of messages received by any set of t players will be completely independent from all the inputs. During the whole computation each gate which evaluates to some $s \in E$, will be "evaluated" by the players by sharing the secret value of s using a completely independent from all the inputs, random polynomial $f(x)$ of degree t, with the only restriction that $f(0) = s$. In particular at the end of the computation we will have the value of F shared among the players in a similar manner. If we want to let just one player know the output value, all the players send their shares to that particular player. This player can compute the interpolation polynomial $f(x)$ and use its free coefficient as the result.

Note that there is a one-to-one correspondence between the set of all shares and the coefficients of the polynomial $f(x)$. Since all the coefficients of $f(x)$, except for its free coefficient, are uniform random variables that are independent of the inputs, the set of all shares does not contain any information about the inputs that does not follow from the value of $f(0)$.

The Computation Stage

Let $a, b \in E$ be two secrets that are shared using the polynomials $f(x), g(x)$ respectively, and let $c \in E$, $c \neq 0$ be some constant. It is enough to show how one can "compute" $c \cdot a$, $a + b$, and $a \cdot b$.

The two linear operations are simple and for their evaluation we do not need any communication between the players. This is because if $f(x)$ and $g(x)$ encode a and b, then the polynomials $h(x) = c \cdot f(x)$ and $k(x) = f(x) + g(x)$ encode $c \cdot a$, $a + b$ respectively. Thus to compute for example $a + b$, each player P_i holding $f(\alpha_i)$, and $g(\alpha_i)$ can compute $k(\alpha_i) = f(\alpha_i) + g(\alpha_i)$. Likewise, since c is a known constant P_i can compute $h(\alpha_i) = c \cdot f(\alpha_i)$. Furthermore, $h(x)$ is random if only $f(x)$ was, and $k(x)$ is random if only one of $f(x)$ or $g(x)$ was.

As a corollary we immediately have

Lemma: (Linear Functional) For any t, $(t \leq n-1)$, and any linear functional

$$F(x_0, \ldots, x_{n-1}) = a_0 x_0 + \cdots + a_{n-1} x_{n-1}$$

where each P_i has input x_i and the a_i are known constants, can be computed t-privately.

From the lemma we have

Corollary: (Matrix Multiplication) Let A be a constant $n \times n$ matrix, and let each P_i have an input variable x_i. Let $X = (x_0, \ldots, x_{n-1})$ and define $Y = (y_1, \ldots, y_n)$ by

$$Y = X \cdot A,$$

then for any t, $(t \leq n-1)$, we can t-privately compute the vector Y such that the only information given to P_i will be the value of Y_i, for $i = 0, \ldots, n-1$.

Proof: Matrix multiplication is just the evaluation of n linear functionals. By the Lemma, we can compute each linear functional Y_i independently, and reveal the outcome only to P_i.

The multiplication step

The multiplication step is only a bit harder. Let a and b be encoded by $f(x)$ and $g(x)$ as above. We now assume that $n \geq 2t+1$. Note that the free coefficient of the polynomial $h(x) = f(x)g(x)$ is $a \cdot b$. There are two problems with using $h(x)$ to encode the product of a times b. The first, and obvious one, is that the degree of $h(x)$ is $2t$ instead of t. While this poses no problem with interpolating $h(x)$ from its n pieces since $n \geq 2t+1$, it is clear that further multiplications will raise the degree, and once the degree passes n, we will not have enough points for the interpolation. The second problem is more subtle. $h(x)$ is not a *random* polynomial of degree $2t$ (ignoring of course the free coefficient). For example, $h(x)$, as a product of two polynomials, cannot be irreducible.

To overcome these two problems we will, in one step, randomize the coefficients of $h(x)$, and reduce its degree while keeping the free coefficient unchanged. We first describe the degree reduction procedure and then combine it with the randomization of the coefficients.

The degree reduction step

Let

$$h(x) = h_0 + h_1 x + \cdots + h_{2t} x^{2t}$$

and let

$$s_i = h(\alpha_i) = f(\alpha_i) g(\alpha_i),$$

for $i = 0, \ldots, n-1$ be the "shares" of $h(x)$. Each P_i holds an s_i. Define the truncation of $h(x)$ to be

$$k(x) = h_0 + h_1 x + \cdots + h_t x^t,$$

and $r_i = k(\alpha_i)$ for $i = 1, \ldots, n-1$.

Claim: Let $S = (s_0, \ldots, s_{n-1})$ and $R = (r_0, \ldots, r_{n-1})$ then there is a constant $n \times n$ matrix A such that

$$R = S \cdot A.$$

Proof: Let H be the n-vector

$$H = (h_0, \ldots, h_t, \ldots, h_{2t}, 0, \ldots, 0)$$

and let K be the n-vector

$$K = (h_0, \ldots, h_t, 0, \ldots, 0).$$

Let $B = (b_{i,j})$ be the $n \times n$ (Vandermonde) matrix, where $b_{i,j} = \alpha_j^i$ for $i, j = 0, \ldots, n-1$. Furthermore, let P be the linear projection

$$P(x_0, \ldots, x_{n-1}) = (x_0, \ldots, x_t, 0, \ldots, 0).$$

We have

$$H \cdot B = S$$

$$H \cdot P = K$$

and

$$K \cdot B = R.$$

Since B is not singular (because the α_i-s are distinct) we have

$$S \cdot (B^{-1} P B) = R$$

but $A = B^{-1} P B$ is some fixed constant matrix, proving our claim.

The randomization step

As noted above the coefficients of the product polynomial are not completely random, and likewise the coefficients of its truncation $k(x)$ may not be completely random. To randomize the coefficients, each player P_i randomly selects a polynomial $q_i(x)$ of degree $2t$ with a zero free coefficient, and distributes its shares among the players. By a simple generalization of the argument in Shamir's [Sh] scheme, it is easy to see that knowing t values on this polynomial gives no information on the vector of coefficients of the monomials of $x, x^2, \ldots, x^t$ of $q_i(x)$.

Thus instead of using $h(x)$ in our reduction we can use

$$\tilde{h}(x) = h(x) + \sum_{j=0}^{n-1} q_j(x)$$

which satisfies $\tilde{h}(0) = h(0)$ but the other coefficients of x^i, $1 \leq i \leq t$, are completely random. Since each player can evaluate his point $\tilde{s}_i = \tilde{h}(\alpha_i)$, we can now apply the truncation procedure using the matrix multiplication lemma to arrive at a completely random polynomial $\tilde{k}(x)$ which satisfies both $\deg \tilde{k}(x) = t$, and $\tilde{k}(0) = a \cdot b$, and $k(x)$ is properly shared among all the players.

Thus (omitting many well known details, see [GMW]) we have proved

Theorem 1: *For every (probabilistic) function F and $t < n/2$ there exists a t-private protocol.*

Remarks:

(1) The complexity of computing F t-privately is bounded by a polynomial (in n) factor times the complexity of computing F.

(2) If F can be computed by an arithmetic circuit over some field using unbounded fan-in linear operation and bounded fan-in multiplication, in depth d, then F can be computed t-privately in $O(d)$ rounds of exchange of information.

(3) In our construction we have to reduce the degree of our polynomial only when its degree is about to pass $n-1$. Thus if $t = O(n^{1-\epsilon})$, for some fixed $\epsilon > 0$, and we start with polynomials of degree t, the players can simulate many steps of the computation before the degree comes close to n, by doing the computation each on their own shares, without any communication(!). When the degree does get close to n, we reduce the degree back to t in one radomizing, degree reducing step.

Two simple examples are:

a. Any Boolean function $F: \{0,1\}^n \rightarrow \{0,1\}$ can be represented as a multilinear polynomial over the field F. Thus if $t = O(n^{1-\epsilon})$ we can compute t-privately, in parallel, all the monomials of F in $O(1)$ number of rounds and then use a big fan-in addition to evaluate F. This procedure may use exponentially long messages but only constant number of rounds.

b. The Boolean Majority function has a polynomial size $O(\log n)$ depth circuit, and thus for $t = O(n^{1-\epsilon})$, this function can be computed t-privately using only polynomially long messages in constant number of rounds.

For completeness we state the following simple result

Theorem 2: *There are functions for which there are no $n/2$ – private protocols.*

Proof: It is easy to see that two players, each holding one input bit, cannot compute the OR function of their bits, without one of them leaking some information. This immediately generalizes to prove the theorem.

Sharing a secret with Cheaters:

Let $n = 3t + 1$ and let $P_0, \ldots, P_{n-1}$ be a set of n players among which we want to share a secret such that

(A) Any set of at most t players does not have any information about the secret and

(B) It is easy to compute the secret from all its shares even if up to t pieces are wrong or missing.

The following scheme achieves both requirements:

Let E be a (finite) field with a primitive n-th root of unity, $\omega \in E$, $\omega^n = 1$ and for all $1 < j < n$, $\omega^j \neq 1$. Without loss of generality we can assume that our secret s is in E.

Pick a random polynomial $f(x) \in E[x]$, of degree t such that $f(0) = s$. That is, set $a_0 = s$ and pick random $a_i \in E$ for $i = 1 \ldots t$ and set

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t.$$

Define the share of P_i, $i = 0 \ldots n-1$, to be $s_i = f(\omega^i)$. As in [Sh], the s_i-s are t-wise independent random variables that are uniformly distributed over E, and thus our first requirement (A) is met.

Note that setting $a_i = 0$ for $i > t$ makes our secret shares the Discrete Fourier Transform of the sequence $(a_0, \ldots, a_{n-1})$. Let $\hat{f}(x) = s_0 + s_1 x + \cdots + s_{n-1} x^{n-1}$. By the well known formula for the inverse transform

$$a_i = \frac{1}{n} \hat{f}(\omega^{-i})$$

and in particular $\hat{f}(\omega^{-i}) = 0$ for $i = t+1, \ldots, n-1$. Explicitly the s_i satisfy the linear equations

$$\sum_{i=0}^{n-1} \omega^{r \cdot i} \cdot s_i = 0 \quad \text{for} \quad r = 1, \ldots, 2t.$$

Thus the polynomial $g(x) = \prod_{i=t+1}^{n-1} (x - \omega^{-i})$ divides the polynomial $\hat{f}(x)$, which in the language of Error Correcting Codes says that the vector $s = (s_0, \ldots, s_{n-1})$ is a codeword in the Cyclic Code of length n generated by $g(x)$. By our choice of $g(x)$, this cyclic code is the well known Generalized Reed-Miller code. Such codes have a simple error correction procedure to correct $\frac{1}{2} \deg g(x) = t$ errors. See for example [PW, page 283].

Verifying a secret

Assume that player P has distributed a secret in the manner described above. Before entering this shared secret into a computation we wish to verify that the

secret shares we are holding are shares of a real secret and not some n random numbers. We want to do so without revealing any information about the secret or any of its shares. This is easily done using the following Zero Knowledge proof technique. We will later show how to verify a secret using a different technique that has absolutely no probability of error. We present this Zero Knowledge technique because it is simpler, and uses fewer rounds of communication.

Simple verification of a secret

Let f_0 be the original polynomial. Let $f_1, \ldots, f_m$, $m = 3n$ be random polynomials of degree t generated by P, and have P send to P_i the values $f_j(\omega^i)$ for $j = 1, \ldots, m$. Each P_i selectes a random $\alpha \neq 0$ from E and sends it to all the other players. After reaching agreement on the set of α-s the dealer broadcasts the set of polynomials $f^\alpha = \sum_{k=0}^{m} \alpha^k f_k$ to all players. Each player P_i checks that at the point ω^i, the shares he received satisfy the required equations, for all the α-s. If some P_i finds an error he broadcasts his complaint. If $t+1$ or more player file a complaint, we decide that the dealer is faulty and take some default value, say 0, to be the dealers secret, (and pick 0 for all the needed shares).

Claim: Let T be a set of good players that did not complain. Let f_i^T be the the interpolation polynomial through the points in T of the original polynomial f_i. Then with probability at least

$$1 - m2^n/|E|$$

all the polynomials f_i^T are of degree t.

Proof: Omitted.

Keeping in mind the (polynomial) complexity of the players computation, we can certainly allow $|E| \geq 2^{2n}$. This makes the error probability exponentially small. (The case of small $|E|$ is similar: Using a somewhat larger m, each player, using a different set of random polynomials, asks the dealer to reveal either f_i or $f_0 + f_i$.)

Note that if $n \geq 5t + 1$, then our secret sharing scheme can correct $2t$ errors. If a secret is accepted then at most t good players may have wrong values. This together with at most t more wrong values that may come from the bad players, gives altogether at most $2t$ errors. Thus in this case the secret is uniquely defined and there is a simple procedure to recover its value using the error correcting procedure.

To handle the case of $n = 3t+1$ we must make sure that all the pieces in the hands of the good players lie on a polynomial of degree t. To achieve this we ask the dealer of the secret to make public all the values that were sent to each player who filed a complaint. We now repeat the test, using new random α-s. Each player now checks at his point and at all the points that were made public, and if there is an error he files a complaint. If by now more than $t+1$ players have complained we all decide that the secret is bad and take the default zero polynomial. Otherwise,

Claim: With very high probability, all good players are on a polynomial of degree t.

Proof: Omitted.

Note that if the dealer is correct then no good player's value will become public during the verification process. This together with the fact that all the polynomials that the dealer reveals during this verification procedure are completely independent from the secret polynomial f_0, ensures that the bad players will not gain any information about the dealer's secret. (Detailed proof omitted).

Absolute verification of a secret

The verification procedure described above leaves an exponentially small probability of error. In this section we describe a secret verification procedure that leaves no probability of errors[1].

Instead of just sending the shares $\{s_i\}$, the dealer of the secret selects n random polynomials $f_0(x), \ldots, f_{n-1}(x)$, with

(1) $s_i = f_i(0)$ for $i = 0, \ldots, n-1$, and

(2) $\sum_{i=0}^{n-1} \omega^{r \cdot i} f_i(x) = 0$ for $r = 1, \ldots, 2t$

In other words, the dealer selects a random polynomial $f(x, y)$, of degree t in both variables x and y, with the only restriction that $f(0,0) = s$ (his secret). Then he sends the polynomials $f_i(x) = f(x, \omega^i)$ and $g_i(y) = f(\omega^i, y)$ to player P_i, for $i = 0, \ldots, n-1$. The real share is just $s_i = f_i(0)$, but for the purpose of its verification, the dealer also sends the polynomials $f_i(x)$ and $g_i(y)$. At this point each player P_i sends $s_{i,j} = f_i(\omega^j) = f(\omega^j, \omega^i) = g_j(\omega^i)$ to each player P_j.

Note that if the dealer is correct, then when a good player P_j is looking at the sequence $SS_j = (s_{0,j}, s_{1,j}, \ldots, s_{n-1,j})$, then all these points should be on his polynomial $g_j(y)$. Therefore P_j can compare the incoming values with his own computation and find out which values are wrong. Furthermore it is

[1] Our original protocol was simplified by Paul Feldman who independently observed that the verification procedure can be accomplished in a constant number of communication rounds.

clear that in this case no good player will have to correct any value coming from other good players.

On the other hand we have

Lemma: If no correct player has to correct a value given by a correct player, then there is a polynomial of degree t that passes through the interpolation points of all the correct players.

Proof: Simple algebra. Omitted.

To make sure that the condition of this lemma is satisfied, each player P_j broadcasts a request to make the coordinates (i,j) he had to correct public. If P_j detects more than t wrong incoming values, or had to correct his own value, the dealer is clearly faulty. In such a case P_j broadcasts a request to make both $f_j(x)$ and $g_j(y)$ public. At this point the dealer broadcasts the (supposedly true) values $s_{i,j}$ at all these points, and the polynomials that were to be made public. Note that making f_j and g_j public makes all the $s_{k,j}$ and $s_{j,k}$ public for $0 \leq k < n$, for that particular j.

Now if some player P_i observes that some new public $s_{i,j}$ contradicts the polynomials he is holding, or finds out the the public information already contradicts itself, he broadcasts a request to make all his information public. Here once more, the dealer makes public all the requested information, Finally, each P_i checks all the public and private information he received from the dealer. If P_i finds any inconsistencies he broadcasts a complaint by asking all his private information to be made public.

If at this point $t+1$ or more players have asked to make their information public, the dealer is clearly faulty and all the players pick the default zero polynomial as the dealer's polynomial. Likewise, if the dealer did not answer all the broadcasted requests he is declared faulty. On the other hand, if t or less players have complaint, then there are at least $t+1$ good players who are satisfied. These uniquely define the polynomial $f(x,y)$ and they conform with all the information that was made public. In this case the complaining players take the public information as their share.

Note that if the dealer has distributed a correct secret then no piece of information of any good player was revealed during the verification process. If however the dealer was bad, we do not have to protect the privacy of his information, and the verification procedure ensures us that all the good players values lie on some polynomial of degree t.

Some more tools

Before going into the computation stage, we need two more tools

(I) Generating (and verifying) a random polynomial of degree $2t$, with a zero free coefficient.

(II) Allowing a dealer to distribute three secrets, a, b, and c, and verifying that $c = a \cdot b$.

Both of these are not needed when $n \geq 4t+1$, but are required to handle the $n = 3t+1$ case.

(I) Generating polynomials of degree $2t$

Let each player P_i distribute t random (including the free coefficient) polynomials $g_{i,k}(x)$, $k = 1, \ldots, t$, of degree t. Define $f_i(x)$ by

$$f_i(x) = \sum_{k=1}^{t} x^k \cdot g_{i,k}$$

and let the players evaluate from their points on the $g_{i,k}$-s their corresponding point on $f_i(x)$.

After we have verified that indeed $\deg g_{i,k} \leq t$, it is clear that $deg f_i(x) \leq 2t$, and $f_i(0) = 0$. (It is also clear that the vector of coefficients of the monomials of x^i, $i = 1, \ldots, t$, in $f_i(x)$ are uniformly distributed and are completely independent from the information held by any set of at most t players that does not include P_i.)

Finally, as our random polynomial we take

$$f(x) = \sum_{i=0}^{n-1} f_i(x).$$

(II) Verifying that $c = a \cdot b$

Let the player P distribute a and b using the polynomials $A(x)$ and $B(x)$ respectively. We want P to also distribute a random polynomial encoding $c = a \cdot b$, in such a way that the players can all verify that indeed $c = a \cdot b$. Let

$$D(x) = A(x) \cdot B(x) = c + c_1 x + \ldots + c_{2t} x^{2t}$$

and let

$$\begin{aligned}
D_t(x) &= r_{t,0} + r_{t,1}x + \ldots + r_{t,t-1}x^{t-1} + c_{2t}x^t \\
D_{t-1}(x) &= r_{t-1,0} + \ldots + r_{t-1,t-1}x^{t-1} + \\
&\quad + [c_{2t-1} - r_{t,t-1}]x^t \\
&\vdots \qquad\qquad \vdots \\
D_1(x) &= r_{1,0} + \ldots + r_{1,t-1}x^{t-1} + \\
&\quad + [c_t - r_{t,1} - r_{t-1,2} - \ldots - r_{2,t-1}]x^t
\end{aligned}$$

where the $r_{i,j}$ are random elements from E. P selects the $D_i(x)$ and distributes their shares to all the

players. After verifying that $A(x)$, $B(x)$ and all the $D_i(x)$ are of degree t, define

$$C(x) = D(x) - \sum_{i=1}^{t} x^i \cdot D_i(x).$$

and verify that $C(x)$ is also of degree t. From the construction of $C(x)$ it is clear that $C(x)$ is a random polynomial of degree t with the only restriction that $C(0) = a \cdot b$.

Proof of Theorem 3

We separate again the computation to its Input, Computation and Final stages. At the input stage, we let each player enter his inputs to the computation using our secret sharing scheme, while verifying that each secret shared is indeed some polynomial of degree t. The secret verification assures that the inputs of any Byzantine player is well defined, but does not ensure that it is in the domain of our function. For example, in a 0-1 vote, we must verify that the input is 0 or 1. We defer this type of verification to the computation stage.

The final stage is exactly the same as in the proof of Theorem 1. When we have simulated the circuit, and the players are holding the pieces of a properly shared secret, encoding the final output, they send all the pieces to one or all the players. As at most t pieces are wrong, each player can use the error correcting procedure and recover the result.

The computation stage - Byzantine case

Let a and b be properly encoded by $f(x)$ and $g(x)$ respectively, where by "properly encoded" we mean that all the pieces of the good players are on some polynomial of degree t. Since $f(x)$ and $g(x)$ are properly encoded the polynomials $f(x)+g(x)$, and $c \cdot f(x)$, properly encode $a+b$, and $c \cdot a$, for any constant $c \in E$. The same argument of Theorem 1 implies that we can do the computation of any linear operation with no communication at all.

Here again, the multiplication step is more involved. To repeat the procedure of theorem 1, using the degree reduction step, via the Matrix Multiplication Lemma, we must make sure the all the players use, as input to this procedure, their correct point on the product polynomial $h(x) = f(x)g(x)$. To guarantee that this indeed happens, we use the Error Correcting Codes again.

Let $a_i = f(\omega^i)$, $b_i = g(\omega^i)$ and $c_i = h(\omega^i) = a_i \cdot b_i$ be the points of P_i on these polynomials. We ask each P_i to pick a random polynomial of degree t, $A_i(x)$, such that $a_i = A_i(0)$, and use this polynomial to distribute a_i as a secret to all the players. Similarly, P_i distributes b_i using $B_i(x)$. We also ask P_i to distribute c_i using the polynomial $C_i(x)$, while verifying that $A_i(x)$, $B_i(x)$, $C_i(x)$ are all of degree t, and that $C_i(0) = A_i(0)B_i(0)$.

We want to verify that the free coefficients of the polynomials $C_i(x)$ are all points on the product polynomial $h(x)$. It is enough to verify that all the free coefficient of the $A_i(x)$, and $B_i(x)$ are on $f(x)$ and $g(x)$ respectively. We do this as follows.

The free coefficient of the $A_i(x)$-s are a code word with at most t errors. By our assumption, all the $A_i(x)$ are properly distributed. We can therefore use them to compute any linear functional. In particular, using the same $A_i(x)$-s we can compute the polynomials

$$S_r(x) = \sum_{i=0}^{n-1} \omega^{r \cdot i} A_i(x)$$

for $r = 1, \ldots, 2t$. At this point all the players reveal their points on the polynomials $S_r(x)$, enabling all the players to recover the value of $s_r = S_r(0)$, for $r = 1, \ldots, 2t$.

Note that if all the $A_i(0)$ are correct (i.e. on a polynomial of degree t) then $s_r = 0$ for all r. Thus the computed value of the s_r are just a function of the errors introduced by the Byzantine players. In particular, this implies that the value of the s_r does not reveal any information that is held in the hands of the good players!

Since at most t of the $A_i(0)$ can be wrong, the value of the s_r-s, the so called Syndrome Vector, is the only information needed by the error correction procedure to detect which coordinates $A_i(x)$ encode a wrong $A_i(0)$, and give the correct value. Therefore, if some $s_r \neq 0$, all the players compute the wrong coordinates, the correct value of $f(\omega^i)$, and use the constant polynomial with this value, instead of $A_i(x)$.

In a similar way we can check and correct the $B_i(x)$. We can, therefore, also check (and correct) the $C_i(x)$, so we are sure that all the inputs to the linear computation we have to do in the degree reduction procedure are correct.

Note that much of this is not needed when $n \geq 4t+1$, because then we can still correct up to t errors on polynomials of degree $2t$. In this case we can do the error correction on the points of $h(x)$ directly.

As in the proof of Theorem 1, we have,

Theorem 3: *For every probabilistic function and every $t < n/3$ there exists a protocol that is both t-resilient and t-private.*

For completeness we state,

Theorem 4: *There are functions for which there is no $n/3$ – resilient protocol.*

Proof: Follows immediately from the lower bound for Byzantine Agreement in this model. We note that even if we allow broadcast as a primitive operation, theorem 4 remains true. This is because we can exhibit functions for three players that cannot be computed resiliently, when one player is bad. This generalizes immediately to $n/3$.

Remark: All the remarks following the statement of theorem 1 apply also to theorem 3.

References

[BL] M. Ben-Or and N. Linial, Collective coin flipping, FOCS86.

[CCD] D. Chaum, C. Crepeau and I. Damgard, Multiparty unconditionally secure protocols, These proceedings.

[DH] W. Diffie and M. E. Helman, New directions in cryptography, IEEE Trans. Inform. Theory, Vol.IT-22,pp.644-654, 1976.

[DS] D. Dolev and R. Strong, Polynomial algorithms for multiple processor agreement. STOC82.

[FM] P. Feldman and S. Micali, Optimal algorithms for Byzantine agreement, These proceedings.

[GMW1] O. Goldriech, S. Micali and A. Wigderson, Proofs that yield nothing but the validity of the assertion, and a methodology of cryptographic protocol design, FOCS86, pp. 174-187.

[GMW2] O. Goldriech, S. Micali and A. Wigderson, How to play any mental game, STOC87, pp. 218–229.

[GMR] S. Goldwasser, S. Micali and C. Rackoff, The knowledge complexity of interactive proof systems, STOC85, pp. 291-304.

[PSL] M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, JACM Vol. 27, pp. 228-234, (1980).

[PW] W. W. Peterson and E. J. Weldon, Error correcting codes, Second Ed., MIT Press, (1972).

[Sh] A. Shamir, How to share a secret, CACM, 22, pp. 612-613, (1979).

[Y1] A. C. Yao, How to generate and exchange secrets, STOC86.

[Y2] A. C. Yao, On the succession problem for Byzantine Generals, manuscript, (1983).

Appendix

Formal Notation

Let F be a field. Let $U = F^n$ denote the standard n-dimensional vector space over F and $M_n(F)$ the ring of $n \times n$ matrices over F.

Let R be a random variable with distribution D over F. Then R^k (R^*) denotes k (finitely many) independent draws from D.

Comment: Unless otherwise specified, F will be finite, and D the uniform distribution over F.

The Basic Model:

Fix $n > 0$ and a field F. Intuitively, an (n, F) – *network* is a complete synchronous network of n probabilistic machines (players) $P_0, P_1, ..., P_{n-1}$. At every round, each player can send one message (element of F) to each other player, receive a message from each other player, and perform arbitrary computation.

If we assume for convenience that players send messages to themselves too, a round of communication is neatly described by a matrix $M \in M_n(F)$, where each P_i sent the i^{th} row of M, and receives the i^{th} column of M. (This formalizes the security of private channels).

Formally, a T round (n, F) – *network* is a set of players $\{P_0, P_1, ..., P_{n-1}\}$. Each P_i is a tuple

$$P_i =< Q_i, q_i^{(0)}, R_i, \delta_i >,$$

where Q_i is a set of states, $q_i^{(0)}$ the initial state, R_i is a random variable over F (distributed like R) and

$$\delta_i : [T] \times Q_i \times F^n \times R_i^* \rightarrow Q_i \times F^n$$

is a transition function that given a round number, state, previous round input and private coin tosses computes the next state and this round's output.

A *protocol* is simply $\delta =< \delta_0, \delta_1, \ldots, \delta_{n-1} >$, the transition functions prescribing to each player what to do in each round.

A *run* M of a protocol δ is a sequence $(M_1, M_2, \ldots, M_T)$, $M_j \in M_n(F)$ of matrices describing the communication in rounds $j = 1, 2, \ldots, T$. Note that M is a random variable, depending on $\{q_i^{(0)}\}$, the initial states, and $\{R_i^*\}$, ($= R^*$), the random draws from D.