Multiparty Computation with Faulty Majority*

Extended Announcement

Donald Beaver
Aiken Computation Laboratory
Harvard University

Shafi Goldwasser
Laboratory for Computer Science
MIT

Abstract. We address the problem of performing a multiparty computation when more than half of the processors are cooperating Byzantine faults. We show how to compute any boolean function of n inputs distributively, preserving the privacy of inputs held by nonfaulty processors, and ensuring that faulty processors obtain the function value "if and only if" the nonfaulty processors do. If the nonfaulty processors do not obtain the correct function value, they detect cheating with high probability. Our solution is based on a new type of verifiable secret sharing in which the secret is revealed not all at once but in small increments. This slow-revealing process ensures that all processors discover the secret at roughly the same time. Our solution assumes the existence of an oblivious transfer protocol and uses broadcast channels. We do not require that the processors have equal computing power.

1 Introduction

Consider a network of n processors, each holding a private input x_i. Given a function $f(x_1, \ldots, x_n)$, the processors must compute f while maintaining the privacy of the local inputs.

The problem of correct and private multiparty computations in the presence of malicious faults has recently received much attention. Yao [36] gave solutions for the two-party case based on the intractability of factoring; Goldreich, Micali and Wigderson [22] gave solutions for the multiparty case where less than half the processors are faulty based on the existence of trapdoor functions and broadcast channels. Ben-Or, Goldwasser and Wigderson [4] and Chaum, Crépeau, Damgärd [10] gave solutions based on the assumption of private channels between pairs of processors, where less than a third of the processors are faulty. The case of less than half faults, with the additional assumption of broadcast channels, was solved by Ben-Or and Rabin [6] (see also Beaver [1], Kilian [32]). Other adversary models were discussed by Chaum [11]. A particularly relevant work is by Galil, Haber and Yung [26] in which an extension of Yao's method for two-party computation to n-party computation is stated. We discuss their protocol and model and compare its properties to our solution in §1.6.

1.1 Results

We consider the case that *more than half* the network consists of cooperating Byzantine faults. The faulty processors are allowed probabilistic polynomial time. We assume broadcast channels are available. Our main result is a completeness theorem for multiparty boolean protocols tolerating any number of faults.

Let n be the total number of players in the network and $t \leq n$ be the number of faulty players.

Let $f : D_1 \times \cdots \times D_n \rightarrow GF(2)$ be any polynomial-time boolean function.Our solution satisfies four essential properties (formal definitions in §2):

- **Independence of Inputs:** The faulty players choose and commit to their inputs x_i independently of the honest players inputs.
- **Privacy:** At the the end of the execution of the protocol, t Byzantine faults cannot to compute any more information about honest players inputs than already implied by the faulty players' private inputs and outputs.
- **Validity:** The honest players will either output the value CHEATING (if the number of active Byzantine faults is greater than $n - t$), or output v such that $v = f(x_1, ..., x_n)$.
- **Fairness:** The faulty players have "no advantage" over the honest players in "learning" the result of the computation at any time of the computation.

Theorem 1 *Let f be a boolean function of n variables represented by a polynomial size arithmetic circuit family. Let the number of faults t satisfy $t < n$. Assume that a protocol for two party oblivious transfer exists. Then there exists a protocol to compute f which achieves independence of inputs, privacy, validity and fairness.*

The assumption of an oblivious transfer protocol is necessary (see §1.4). Note that by [19],[21] a protocol for oblivious transfer exists if trapdoor functions exist.

In this abstract we consider only the special case of a single boolean function. In the full paper we show a solution also for the case that $\{f_i\}$ is a collection of probabilistic polynomial time boolean computations defined on the private inputs $\{x_i\}$ and processor i receives as output value $y_i = f_i(x_1, ..., x_n)$.

1.2 Cheating and Restarting

Our protocols allow only a correct value to be computed, if anything is computed at all. An unavoidable drawback is that the majority rules; the faulty players can always prevent a computation from completing.

We address this problem by presenting an extended solution (in the final paper) which allows the protocol to be restarted after a value CHEATING is output. We call this a

[1]The first author was supported in part under NSF grant CCR-870-4513. The second author was supported in part by NSF grant CCR-8657527 with IBM matching funds, and US-Israel binational grant.

CH2806-8/89/0000/0468/$01.00 © 1989 IEEE

RESTART model. Upon restarting, all faulty players which have been detected are cast out and given a default value. The remaining players are not allowed to change their input.

Though this extension may allow the faulty players some bias on the final output for some functions, at least the honest players are assured that they will ultimately obtain a correct computation based on the inputs of those processors who have remained nonfaulty. Various protocols for the restart model are in preparation for the general case in [8] and the parity function in [25].

1.3 The Method: Slowly Reveal The Output

In the works of [22, 4, 10, 6, 1], the private circuit computation essentially consists of three stages: an input stage in which the processors commit to their inputs via verifiable secret sharing, a computation stage in which the result of every circuit-gate is computed and kept as a shared secret by all processors, and finally an output stage in which each processor sends to all other processors his share of the final circuit output. Using the set of all shares received, the processors compute the output.

In all of these constructions, the condition $2t < n$ was unavoidable in order to achieve the following two conditions:

(1) the faulty processors should not be able to recover the secret output on their own; and

(2) the nonfaulty processors should be able to recover the secret on their own even if the faulty processors quit without sending their shares of the output.

We achieve condition (1) and a modification of condition (2), using a new technique to reveal secrets *fairly*. The faulty processors cannot obtain the secret while preventing the honest players from obtaining the secret, even when $2t \geq n$. Faulty processors which actively fail are identified and cast out.

Our solution relies on a new idea: how to reveal a shared secret slowly. Rather than simply broadcasting their shares of the secret, the processors use a randomized protocol to generate a sequence of coin flips biased toward the secret. If enough faulty processors should quit or misbehave, the nonfaulty processors will detect the faults and withdraw with as much information as the faulty processors, apart from the last coin flip. Thus, the faulty processors can choose to halt the computation, but they learn virtually the same as the nonfaulty processors about the secret, and they are identified as faulty.

The preceding discussion is adequate to describe the case of randomly generated secrets. When the secret v corresponds to the value of an arbitrary function computed on different inputs belonging to different processors, the situation becomes more complicated. Different processors may have different *a priori* probabilities to know the value of v. For example, let $f(x_1, x_2) = x_1$ AND x_2, where the initial distribution of x_1, x_2 is uniform over $\{0, 1\}$. Clearly, if $x_1 = 0$ and $x_2 = 1$ then processor 1 knows the result *a priori* with better probability than processor 2. Given a partial execution, the advantage that some processor may have over others in knowing the result must be discussed with respect to their initial advantage. One contribution of this paper is a definition for fairness which addresses this issue (see §2.5).

Luby, Micali and Rackoff [33] present a result related to our revealing method in which, based on the assumption that distinguishing quadratic residues from non-residues is a hard computational problem, two parties can exchange a single bit secret by flipping a symmetrically biased coin such that each player knows at most "one coin flip" more about the other player's secret than the other player knows about his own. There, the bias of the coin is toward the other player's secret. Our solution has a similar flavor: at the outset of the computation we prepare a sequence of secretly shared bits, each of which is the secret result of a global coin biased toward 0; during the output stage, we reveal the result of the computation exclusive-ored with these secret coins, one by one, to all the players. Our coin flip procedure works for any n and t, and relies on the general assumption that one-way functions exist.

1.4 Oblivious Transfer is Necessary & Sufficient

Our solution assumes the existence of a protocol for two-party oblivious transfer (see §3 for definition) and the existence of a one-way function. A recent result of Bellare and Cowen[8] shows that the existence of an oblivious transfer protocol implies the existence of a one-way function, and thus it is *sufficient* to assume that a two-party oblivious transfer protocol exists.

The assumption of two-party oblivious transfer is *necessary* since a completeness theorem for multiparty boolean computation implies the existence of a two-party oblivious transfer protocol. Let $\{b_0, b_1\}$ be the input of the first player and $\{r, c\} \in \{0, 1\}$ the input of the second player. Define $f(\{b_0, b_1\}, \{r, c\}, \ldots) = r \oplus \{b_c\}$. By computing f the network implements an oblivious transfer protocol between players 1 and 2. Because a multiparty protocol which tolerates half or more faults can be projected into a two-party protocol in which each party simulates half the network, a two-party protocol for OT is implied [2].

Impagliazzo and Rudich[30] show that it will be hard to prove that one-way functions alone are sufficient for implementing two-party oblivious transfer. Their proof does not apply directly to the setting of multiparty protocols, since there may be additional, nonfaulty players to assist the two parties performing the OT. In fact, when less than half the network is faulty, the techniques of [4, 10, 6, 1] imply a method for two-party OT based only on one-way functions (the latter assumption replaces the assumption of private channels). However, when half or more of the network is faulty, we adapt the proof in [30] to the multiparty setting by the projection argument of the previous paragraph [2].

1.5 Noisy Channels

Kilian and Crépeau [31],[18] have shown an information theoretic reduction from the existence of noisy channels (see [18] for definition) to two-party private circuit computation.

In final paper, we show in the multiparty case with faulty majority, an infromation theoretic reduction from the existence of broadcast, private, and noisy channels to a multiparty private function computation as follows. All processors must participate in an initialization phase. If no complaints are made during initialization, results similar to the cryptographic case can be achieved.

Theorem 2 *Let f be a boolean function of n variables represented by a polynomial size arithmetic circuit family. Let $t < n$ be the the number of Byzantine faults. Given broadcast channels, private and noisy channels between every pair of processors, there exists a protocol to compute f such that if all processors complete an initialization phase without broadcasting a complaint, then for any polynomial $p(n)$, with probability $1 - \frac{1}{p(n)}$, the protocol achieves independence of inputs, privacy, validity, and fairness.*

1.6 Comparison with Previous Work

Let us compare our solution to the work of [26] in which an extension of Yao's method for two-party computation to n-party computation (where half or more of the processors may be faulty) is stated. In recent personal communication [27] the protocol was outlined to us. Assume that trapdoor encryption schemes E_t exist. Obliviously compute an encryption of the global function being computed, $E_t(f(x_1, ..., x_n))$; then, reveal the trapdoor information t bit by bit to all the players. The claim is that if bad players quit, the good players are only one bit of the trapdoor behind, and thus the bad players "in some sense" have no computational advantage over the good players in computing the result. To formalize this, they require that the recovery procedure of the good players (after the bad players quit) relies on the bad players faulty program.

This requirement restricts the model in two ways:
1. It implicitly amounts to assuming equal or better computing power and knowledge of algorithms on part of the correct processors, since they can invoke the faulty programs as a subroutine. This is often unrealistic. Say player A, the Bank of America, and player B, a private investor, want to compute a function which tells whether a certain stock is a wise investment. Since the Bank has a thousand times the computing power of the investor, it can quit the computation once it has enough of the trapdoor to determine the result. Meanwhile, the investor cannot discover the result.

2. The faulty programs (including their decision of when to abort the computation) cannot depend on the nonfaulty recovery programs; thus an additional restriction to being polynomial time bounded is put on the faulty players.

The method presented in this paper makes no such restriction.

It is interesting to note that in the model when non-faulty players program is fixed in advance and does not depend on the program of the faulty players, perfect fairness is not achievable, by a generalization of a result of Cleve [15]. Let $n = 2$. If p and q are the maximal *a priori* probabilities of the players to know the value of $f(x_1, x_2)$ given x_1 or x_2 respectively, then for any two-party protocol running for k rounds there exists a quitting strategy of one player enabling him to predict f with probability at least $\frac{\min(1-p,1-q)}{2k}$ better than the other player [16]. Since a multiparty protocol tolerating a majority of faults can be projected as a two-party protocol (tolerating one fault), the lower bound applies in our model.

2 Definitions

In this section we give the background definitions needed to state the properties of our solution.

2.1 Interactive Proofs, Protocols

A *two-party protocol* is a pair of interactive probabilistic Turing machines each with a private input tape, a private random tape, a private output tape, and a public output tape, and sharing a common read-only input tape and two communication tapes (readable by one, writable by the other). See [24, 34] for definitions of interactive proof systems and auxiliary-input zero-knowledge proofs.

A *multiparty protocol* is a set of interactive probabilistic Turing machines $\{ M_1, \ldots, M_n \}$, each with a public input tape, a private input tape, a private random tape, a private output tape, and a public output tape, and sharing a common read-only input tape and a common readable and writable broadcast communication tape. The machines are synchronized by a common clock, and in each round exactly one machine is active.

Let 1^k denote the security parameter. Let x_i denote the input of the ith processors. Let D_i denote the domain in which x_i is selected. Let $f : D_1 \times D_2 \times ... \times D_n \rightarrow V$ be the polynomial time boolean function we want to compute

A *multiparty protocol for f* is a multiparty protocol whose common public input tape contains the security parameter 1^k and a circuit description C_f specifying the function which the machines are to compute. The private input tape for machine M_i contains $x_i \in D_i$.

Let T be a *coalition*, namely a subset of the processors of size at most t, where t is an upper bound on the allowable number of faults. The processors in T are replaced by arbitrary machines M_i^*.

In this paper we will discuss multiparty protocols consisting of three stages: input stage, computation stage and output stage. We assume that C_f is expressed using arithmetic gates over some fixed finite field E.

2.2 Input Independence

A protocol for f achieves *t-independence of inputs* if for all $i \in T$, for all probabilistic polynomial time predicates R, R_1, for all $c > 0$, for all k sufficiently large, $|prob(R_1(\{\hat{x}_i | i \in T\}, 1^k)) - prob(R_1(\{\hat{x}_i | i \in T\}, 1^k) | R(\{ x_j \mid j \notin T\}, 1^k))| < \frac{1}{k^c}$ where $\hat{x}_i$ is the value committed by the i-th faulty player in the input stage. This definition is an extention of the one in [13].

2.3 Privacy

The definition of privacy we apply is that of computational privacy, which is analogous to zero-knowledge: a faulty coalition must be able to generate a view of a run of the protocol according to the same distribution as one produced by an actual run of the protocol, given the faulty inputs and faulty outputs.

Let the complete transcript of the execution including all tapes and everything written on them during the course of the execution be called a *global view* of the protocol, denoted VIEW$(\vec{x}, \vec{\sigma})$. Here, $\vec{x} = <x_1, \ldots, x_n>$ is the list of inputs and $\vec{\sigma} = <\sigma_1, \ldots, \sigma_n>$ is a list of auxiliary inputs to each processor. The portion of the transcript corresponding to tapes that are readable by members of T is called the view of T, denoted VIEW$_T(\vec{x}, \vec{\sigma})$, or VIEW$_T$ for short.

We say that A protocol for f is *computational-t-private* if for any coalition T of size t, there is a probabilistic polynomial time simulator M_T such that for any $\vec{x}$ of inputs, for any output $f(\vec{x})$, for every auxiliary input $\vec{\sigma}$, the ensembles $\{$ VIEW$_T(\vec{x}, \vec{\sigma})\}$ and $\{ M_T(\vec{x}_T, \vec{\sigma}_T)\}$ are polynomially indistinguishable. Here, $\vec{x}_T$ means the set of x_i held by T.

2.4 Validity

We say that a protocol for f achieves *validity* if the nonfaulty players either all output the special value CHEATING or all output v such that for all $c > 0$, for all k sufficiently large, we have $prob(v = f(\hat{x}_1, ..., \hat{x}_n)) > 1 - \frac{1}{k^c}$ where $(\forall i)$ $\hat{x}_i$ was committed by player i in the input stage, and $(\forall i)$ nonfaulty $\hat{x}_i = x_i$ the initial input of the honest players.

2.5 Fairness

Our definition for fairness is motivated by definitions of likelihood and the weight of evidence frequently used in learning theory.

When the protocol terminates we think of every player i as writing its current guess of the value of the function on its

private tape. We denote the i^{th} player's guess at termination as y_i. We denote the guess of the coalition of faulty players at termination by y_T.

Let $\vec{X} = (X_1, ...X_n)$ be a vector of random variables distributed according to input distributions $D_1 \times ... \times D_n$. Let $\vec{X_T}$ be a random variable distributed according to the inputs of faulty players. Let Z be a random variable which takes on as a value the (possibly partial) history of an execution.

Define the following probabilities:

$$\begin{aligned} p_i(x_i, b) &= prob(y_i = b | X_i = x_i) \\ p_T(\vec{x_T}, b) &= prob(y_T = b | \vec{X_T} = \vec{x_T}) \\ p_i(x_i, b, z) &= prob(y_i = b | X_i = x_i, Z = z) \\ p_T(\vec{x_T}, b, z) &= prob(y_T = b | \vec{X_T} = \vec{x_T}, Z = z) \end{aligned}$$

Define $\text{odds}(p) = \frac{p}{1-p}$.

We say that a protocol for f with parameter $d > 0$ is *d-fair* if for all coalitions T, for every $i \notin T$, for all input values $\vec{x}$ committed in the input stage, for all partial histories z, and for all k sufficiently large:

$$(1+\delta)^{-1} \times \frac{odds(p_T(\vec{x_T}, f(\vec{x})))}{odds(p_i(x_i, f(\vec{x})))} \leq \frac{odds(p_T(\vec{x_T}, f(\vec{x}), z))}{odds(p_i(x_i, f(\vec{x}), z))}$$

$$\leq (1+\delta) \times \frac{odds(p_T(\vec{x_T}, f(\vec{x})))}{odds(p_i(x_i, f(\vec{x})))}$$

where $\delta \geq \frac{1}{k^d}$. We say that a protocol for f is *fair* if for every parameter $d > 0$ it achieves d-fairness.

3 Tools

Two Party Protocols. In [36], a protocol for two-party private circuit computation was given, based on the assumption that factoring is hard. Goldreich, Micali, and Wigderson [22] generalized that result to rely on the existence of any trapdoor one-way function. We give a new solution to two-party private circuit computation based on the existence of a protocol for two-party oblivious transfer.

Informally, a two-way party oblivious transfer protocol here is defined by two probabilistic polynomial time in k (the common security parameter) Turing machines A and B which each have a private random tape, private input tape, and private output tape. A has two private inputs $b_0, b_1 \in \{0,1\}$ and B has a private value $c \in \{0,1\}$. At the end of the protocol, B private output is b_c with high probability, and the probability that A's output is c is at most $\frac{1}{2} + \epsilon$ (where ϵ is negligible.)

Simultaneous VSS. We shall require a threshold scheme, namely a protocol which allows one processor to distribute a secret x so that (1) despite faults, a quorum of processors can reconstruct x; (2) any group of processors with less than a quorum cannot obtain any information about x. (to assure privacy, the quorum size must exceed the bound t on the number of possible faults); and (3) The protocol should be verifiable in the sense that if the dealer of the secret has not followed the protocol, then with high probability all nonfaulty processors will discover it immediately.

Shamir's method [35] for secret sharing involves selecting a random polynomial $f(x)$ of degree $q-1$ and free term s and to privately send the piece $\text{piece}_i(s) = f(i)$ to player i.Modification of this method making it a verifiable secret sharing (VSS) scheme were given in [12, 22, 6, 20, 7] for the case $2t < n$.

When $2t \geq n$, the minority of nonfaulty players cannot force the secret to be reconstructed. We present a new method for VSS which not only allows the nonfaulty players to verify that the secret is correct if it is in fact revealed, but which gives them equal knowledge of the secret if it is only partially revealed. When the dealer distributes pieces of a random polynomial, he must also broadcast encryptions of the pieces and the coefficients of the polynomial, including the free term. He gives zero-knowledge proofs that the pieces interpolate to a t^{th} degree polynomial.

Theorem 3 *Given a protocol for Oblivious Transfer, there exists a protocol for Simultaneous Verifiable Secret Sharing satisfying fairness, validity and privacy for any $t < n$.*

4 The Protocol for Correct Processors

In this section we describe a protocol which satisfies the properties of fairness, privacy, and correctness when the processors are restricted to fail-stop errors; namely, all processors produce messages identical with their given programs, but an adversary can cause them to halt at an arbitrary time. In the final paper, we add further requirements to the protocol in order to ensure that all processors behave or are detected and disqualified.

The overall protocol is divided into Input, Evaluate, and Output stages. During the Input stage, the processors agree on encryption keys $\{ E_i \}_{i=1..n}$ and $\{ E_{ij} \}_{i \neq j}$ (processor i knows the decryptions D_i, D_{ij} of E_i, E_{ij} for all j). They broadcast encryptions of their inputs and their random tapes and then verifiably secret share their inputs. We take the encrypted input values as the true inputs of all players. The advantage to using polynomial secret sharing is that in order to abort the protocol an adversary must cause sufficiently many Byzantine faults that every nonfaulty processor will output CHEATING and be able to identify a large number of faults.

4.1 The Circuit Evaluation Stage

Let the generic arithmetic circuit to be evaluated be C_f. We consider two types of gates, one computing linear combinations of inputs, the other computing products of inputs.

Linear combinations of secrets shared using polynomials are straightforward: if $f(x)$ and $g(x)$ have free terms u and v, then $af(x) + bg(x) + c$ is a polynomial of degree t with free term $au + bv + c$.

EVALUATE C_f.
For gates $j = 1, \ldots, \text{size}(C_f)$ do:

- If gate j is a linear combination $x_j = ax_k + bx_l + c$, set $\text{piece}_i(x_j) = a \cdot \text{piece}_i(x_k) + b \cdot \text{piece}(x_l) + c$.
- If gate j is a product $x_j = x_k x_l$, run the MULTIPLY-SECRETS protocol using $\text{piece}_i(x_k), \text{piece}_i(x_l)$.

Figure 1: Protocol to evaluate circuit C_f. (Code for player i.)

Unlike addition, the multiplication of secrets is not a simple as multiplying $\text{piece}_i(u) \cdot \text{piece}_i(v)$. The result would indeed be an evaluation point of a polynomial $h(x) = f(x)g(x)$ such that $h(0) = f(0)g(0) = uv$, but the degree of $h(x)$ would be $2t$, which is too large. As in [4], techniques to reduce the degree of the product polynomial come in handy. The techniques developed there do not apply here, since there are an excess of faulty processors. We present here a new method that solves the problem of multiplying secrets.

4.2 Synthesis of New Pieces

In order to provide each player with a piece of a random polynomial $H(x)$ of degree t and free term $H(0) = h(0) = uv$, the system computes and broadcasts the values $h(-1), \ldots, h(-t)$. It is then a matter of straightforward algebra for each processor to express its new piece $\text{piece}_i(uv) = H(i)$ as a linear combination of the publicized values and its private value $\text{piece}_i(u) \cdot \text{piece}_i(v) = f(i)g(i)$.

Our technique is to reduce the problem of synthesizing a new value $h(m)$ to the problem of multiplying two weighted sums of private pieces. This latter problem was solved by Galil, Haber, and Yung for the field GF(2) in [26]. We generalize their technique in the next section. First let us present the reduction from synthesis to multiplying sums.

Recall the LaGrange interpolation polynomial,

$$L_i(x) = \prod_{j \neq i} \frac{x - j}{i - j}$$

The key observation is that

$$f(m) = \sum_i L_i(m) f(i)$$

is a linear combination of private values, since each value $L_i(m)$ is publicly known, while each $p(i)$ is private.

We can now express $h(m)$ as the product of two sums, where the addends in each sum are weighted pieces of u and v:

$$\begin{aligned} h(m) &= f(m)g(m) = (\textstyle\sum_i L_i(m) f(i))(\sum_j L_j(m) g(j)) \\ &= (\textstyle\sum_i L_i(m) \cdot \text{piece}_i(u))(\sum_j L_j(m) \cdot \text{piece}_j(v)) \end{aligned}$$

4.2.1 Multiplying Sums

In [26], a scheme is presented for computing $(\oplus a_i)(\oplus b_i)$, where the a_i and b_i are bits. Let us address the general problem of secretly computing $(\sum a_i)(\sum b_j)$, where processor i holds a_i and b_i. The solution is to run a collection of pairwise computations for each i and j [26]. Each pair of processors, $i < j$, computes the values $(a_i b_j - r)$ for player i and r for player j, where r is a newly generated uniformly random field element. Clearly, this provides i and j with values which are each uniformly random but together determine $a_i b_j$.

MULTIPLY-SUM

- For $j = i+1, \ldots, N$ run MULTIPLY-PAIRWISE on a_i and b_j, where player j supplies b_j. Obtain $c_{ij} = a_i b_j - r_{ij}$. (Player j receives $c_{ji} = r_{ij}$.)
- For $j = 1, \ldots, i-1$ run MULTIPLY-PAIRWISE on a_i and b_j, where player j supplies b_j. Obtain $c_{ij} = r_{ji}$. (Player j receives $c_{ji} = a_i b_j - r_{ji}$.)
- Share the value $s_i = \sum_j c_{ij}$. Receive a piece $\text{piece}_i(s_j)$ from every player.
- Run the secret addition protocol using $\text{piece}_i(s_1), \ldots, \text{piece}_i(s_N)$. Use $\text{piece}_i(\sum s_i)$ as your piece of the result, s.

Figure 2: Protocol to multiply two sums of secrets. (Code for player i.)

The two-party subprotocol MULTIPLY-PAIRWISE used in MULTIPLY-SUM is based on two-party oblivious circuit evaluation of the following functions: player A supplies a and a random field element r_A; player B supplies b and a random field element r_B; compute $F = (ab - r_A - r_B)$ for player A; compute $G = (r_A + r_B)$ for player B.

4.3 Multiplying Secrets

Let $H(x)$ be the unique polynomial of degree t satisfying

$$H(x) \equiv h(x) \bmod x^{t+1}.$$

Then using the Chinese Remainder Theorem it is easy to express $H(i)$ as a linear combination of $h(i)$ and $h(-1), \ldots, h(-t)$. This will be the new piece for player i. For simplicity we omit a technical detail: a subprotocol which adds a random polynomial $r(x)$ of degree t and $r(0) = 0$, to randomize the coefficients of $H(x)$.

MULTIPLY-SECRETS(U,V).

- Privately compute $h(i) = f(i)g(i)$.
- Use $h(i)$ as input to a protocol to synthesize and obtain $h(-1), \ldots, h(-t)$.
- Compute $H(i)$ using the Chinese Remainder Theorem, and let $\text{piece}_i(uv) = H(i)$.

Figure 3: Protocol to multiply two secrets.

5 Slowly Revealing the Output

In this section we describe a technique for revealing a secret V without allowing the faulty processors to gain more than a negligible amount of confidence in the result in excess of the correct processors. We rely on the circuit evaluation protocols of the previous section.

Recall that d is a parameter of the protocol specifying the fairness constraint. The processors compute a polynomial number of secret coins biased to 0 with probability exactly $1/2 + 1/k^d$. By exclusive-or'ing the secret with a coin and then revealing the result, we obtain a sample of the secret's true value (cf. [33]). If the faulty processors prevent the good players from seeing the exclusive-or'ed result, the good players quit. At that point, the faulty players have at most one additional sample of a coin slightly biased toward V.

Sampling a polynomial number of times, the processors can deduce the value of V with high probability.

Let $R(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$. If any x_i is a uniformly random bit, then R is a uniformly random bit. Let $\phi(x_1, \ldots, x_{2k^d}) = 1$ if $x_1 + \ldots + x_{2k^d} \geq k^d + 1$, else $\phi = 0$. If the inputs to ϕ are uniformly random bits, then the output of ϕ is a coin biased to 0 with bias $\frac{1}{2} + \frac{1}{k^d}$. Figure 4 describes the protocol.

It is easy to see that if the revelation finishes, then the majority of the biased coin tosses is V with probability exponentially close to 1, so the nonfaulty players know the result. On the other hand, if the nonfaulty players output CHEATING, then the faulty players learn at most one additional sample of V. Their odds of guessing V are at most a factor $\frac{1}{k^d}$ greater than that of the good players.

The description of the underlying protocol for fail-stop faults is complete. As per [22], we transform the protocol into a protocol tolerating Byzantine faults by adding zero-knowledge proofs to ensure that faulty processors behave according to the protocol or are detected.

- For $l = 1 \ldots k^{3d}$ do sequentially:
  - For $m = 1 \ldots 2k^d$ do:
    - Each player i chooses a random bit b_{im}.
    - Run EVALUATE-R$(b_{1m}, \ldots, b_{nm})$. Let c_m be the output.
  - Run EVALUATE-$\phi(c_1, \ldots, c_{2k^d})$ to obtain a secret, biased coin, e_j.
  - Run EVALUATE-exor on V and e_j to obtain the masked result r_j.
  - Each player i broadcasts his piece of r_j.
  - Each player i interpolates r_j if $n - t$ pieces have been broadcast; otherwise he outputs CHEATING .
- Output MAJORITY$(r_1, \ldots, r_{k^{3d}})$.

Figure 4: Protocol SLOWLY-REVEAL(V).

6 Extensions?

In the case that less than half the processors are faulty, extending the solution for boolean functions to general functions is immediate. Because the nonfaulty majority rules, if the network computes the bits of the output in parallel, all of the output bits are guaranteed eventually to be output.

In the case of a faulty majority, however, the faulty processors can choose to abort the computation after discovering the result of a few of the output bits, which in some situations may give them a large advantage by virtue of their particular inputs and the particular function. This is best illustrated by the following example for $n = 2, t = 1$. Let x and y be drawn at random from $\{0,1\}^k$. Let $f(x,y) = y$ if x is even, and the bitwise complement $\overline{y}$ of y otherwise. Then, revealing the bits of the output $f(x,y)$ slowly in parallel will not work. At an intermediate point in the revealing process the second player can favor y if the majority of his current guesses for the outputs match y; otherwise he favors $\overline{y}$, while the first player has no corresponding strategy.

This objection also applies to the solution of [26] outlined in §1.6. For general trapdoor functions, when part of the trapdoor is revealed, it is no longer clear that the probabilistic encryption scheme utilized is still as secure as when none of the trapdoor was known. It is quite plausible that a polynomial time algorithm which knows half the trapdoor bits might be able to guess an encrypted bit with probability $\frac{3}{4}$. One can *assume* that revealing a fraction of the trapdoor does not affect the security of the encryption scheme, but this can be shown quite unlikely for encryption schemes based on RSA-like functions.

Acknowledgements Many friends helped us, especially in discussions on the nature of fairness. We are particularly grateful to Richard Cleve, Oded Goldreich and Yishai Mansour. The observations about the necessity of an oblivious transfer protocol were obtained with Yishai. Thanks also to Benny Chor (via Otto), Phil Klein, Nati Linial, and Ron Rivest.

References

[1] D. Beaver. "Secure Multiparty Protocols Tolerating Half Faulty Processors." Technical Report TR-19-88 (1988), Harvard University.

[2] D. Beaver, S. Goldwasser, Y. Mansour. Personal communication.

[3] D. Beaver, R. Cleve, S. Goldwasser. In preparation.

[4] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." Proc. of 20^{th} STOC (1988), 1-10.

[5] M. Ben-Or, O. Goldreich, S. Micali, R. Rivest. "Fair Contract Signing." Proc. of ICALP (1985).

[6] M. Ben-Or, T. Rabin. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." Proc. of 21^{st} STOC (1989).

[7] J. Benaloh. "Verifiable Secret Ballot Elections" Yale University PhD thesis.

[8] M. Bellare, L. Cowen, S. Goldwasser. "On the Power of Secret Key Exchange." In preparation.

[9] Blakely, "Security Proofs for Information Protection Systems." Proceedings of the 1980 Symposium on Security and Privacy, IEEE Computer Society Press, NY (1981), 79-88.

[10] D. Chaum, C. Crépeau , I. Damgärd . "Multiparty Unconditionally Secure Protocols." Proc. of 20^{th} STOC (1988), 11-19.

[11] D. Chaum. "Multi Party Protocols with Disruptors and Colluders." CRYPTO88 Rump Session.

[12] B. Chor, S. Goldwasser, S. Micali, B. Awerbuch. "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults." FOCS 1985.

[13] B. Chor, M. Rabin. "Achieving Independence in a Logarithmic Number of Rounds."

[14] B. Chor, E. Kushilevitz. "A 0/1 Law for Boolean Privacy." Proc. of 21^{st} STOC (1989).

[15] R. Cleve, "Limits on the Security of Coin Flips when Half the Processors are Faulty." Proc. of 19^{th} STOC (1986), 364-370.

[16] R. Cleve, personal communication.

[17] J. Cohen, M. Fischer. "A Robust and Verifiable Cryptographically Secure Election." Proc. of 26^{th} FOCS (1985).

[18] C. Crépeau , J. Kilian. "Achieving Oblivious Transfer Using Weakened Security Assumptions." Proc. of 29^{th} FOCS (1989).

[19] S. Even, O. Goldreich, A. Lempel. "A Randomized Protocol for Signing Contracts." Proc. of CRYPTO 1982, 205-210.

[20] P. Feldman, "A practical scheme for Noninteractive Verifiable Secret Sharing." Proc. of 19^{th} STOC (1987).

[21] Goldreich, O., Micali, S., A. Wigderson. "Proofs that Yield Nothing But Their Validity and a Methodology of Cryptographic Protocol Design." Proc. of 27^{th} FOCS (1986), 174-187.

[22] Goldreich, O., Micali, S., A. Wigderson. "How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority." Proc. of 19^{th} STOC (1987), 218-229.

[23] O. Goldreich, R. Vainish. "How to Solve any Protocol Problem – An Efficiency Improvement." Proc. of CRYPTO 1987.

[24] S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." Siam J. of Comp. Feb. 1989.

[25] Y. Mansour. "Unbiased Protocols for Parity". Personal communication.

[26] Z. Galil, S. Haber, M. Yung. "Cryptographic Computation: Secure Faulty-Tolerant Protocols and the Public Key Model." Proc. of CRYPTO 1989.

[27] S. Haber, M. Yung. Personal communication.

[28] S. Haber. "Multi-Party Cryptographic Computation: Techniques and Applications." Ph.D. Thesis, Columbia University, 1988.

[29] S. Haber, S. Micali. personal communication.

[30] R. Impagliazzo, S. Rudich. "Limits on The Provable Consequences of One-Way Permutations." Proc. of 21^{st} STOC (1989), 44-62.

[31] J. Kilian. "Founding Cryptography on Oblivious Transfer." Proc. of 20^{th} STOC (1988), 20-29.

[32] J. Kilian, personal communication.

[33] M. Luby, S. Micali, C. Rackoff. "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically Biased Coin." Proc. of 24^{th} FOCS (1983).

[34] Y. Oren. "On the Cunning Power of Cheating Verifiers: Some Observations about Zero Knowledge Proofs." Proc. of 19^{th} STOC (1987), 462-471.

[35] A. Shamir. "How to Share a Secret." CACM 22 (1979), 612-613.

[36] A. Yao. "How to Generate and Exchange Secrets." Proc. of 27^{th} FOCS (1986), 162-167.