Fair Computation of General Functions in Presence of Immoral Majority

Shafi Goldwasser[*]
MIT

Leonid Levin[†]
Boston University[‡], MIT

Abstract

This paper describes a method for n players, a majority of which may be faulty, to compute *correctly*, *privately*, and *fairly* any computable function $f(x_1, \ldots, x_n)$ where x_i is the input of the i-th player. The method uses as a building block an oblivious transfer primitive.

Previous methods achieved these properties, only for boolean functions, which, in particular, precluded composition of such protocols.

We also propose a simpler definition of security for multi-player protocols which still implies previous definitions of privacy and correctness.

1 Introduction

The problem of performing a distributed computation in a fault-tolerant and private manner has been addressed by many researchers in the past few years.

In a sequence of papers [Goldreich Micali Wigderson 87, Ben-Or Goldwasser Wigderson 88, Chaum Crepeau Damgaard 88, Ben-Or Rabin 89] it has been shown that when both private channels between pairs of players and broadcast channels are available, any distributed computation (e.g. function or game) can be performed privately and correctly, in spite of worst case behavior of the faulty players, if they are in minority.

When in majority, faulty players can be shown to be able to prevent the completion of certain computations by quitting early. Moreover, they may quit while being "ahead", i.e. having learned more about the output than non-faulty players.

A special computation problem where quitting early is especially harmful was addressed by [Luby Micali Rackoff 83]: the simultaneous exchange between two players of random secret bits. Each player must be protected against the case the other player

[*]Supported by ARO grant DAAL 03-86-K-0171 and NSF PYI grant 8657527-CCR with IBM matching funds

[†]*Supported by an NSF grant , and the MIT laboratory of computer science.*

[‡]Computer Science dept., 111 Cummington St., Boston, MA 02215; e-mail: Lnd@cs.bu.edu

quits early. The fairness notion they proposed (and achieved under the assumption the quadratic residue problem is hard) is that the probability that player A knows the secret bit of player B is within an ϵ of the probability that B knows the secret bit of A (the protocol is polynomial time in ϵ^{-1}).

[Yao 86] proposed (and showed how to achieve under the assumption that integer factorization problem is hard) the following notion of fairness for arbitrary two party boolean protocols. Suppose two players A and B want to compute a boolean function f privately and correctly. Informally, a protocol is fair if given any strategy of a faulty A, the non-faulty B has a strategy such that the probability that B will learn f, given that A will learn f is at any time during the protocol is as high as it is in the beginning of the protocol. The solution is based on the existence of trapdoor functions. These results were extended in ([Galil Haber Yung 89, Brickell Chaum Damgaard VanDeGraaf 87]) to the multi-player case.

The drawback of the above fairness definition is its severe limitation on the power of the faulty players. Since the strategy of the non-faulty players depends on the strategy of the faulty players, the faulty players program strategy must be chosen first and can not change depending on the program of the non-faulty players.

[Beaver Goldwasser 89] proposes a different notion of fairness, free from this limitation. A protocol to compute function f is said to be fair: if there exists a strategy for player B such that for any strategy of faulty subset of players A the ratio of the odds of B and A to compute the outcome of f is (about) the same at any time during the protocol as it is in the beginning of the protocol. It is shown how to achieve such fairness for multi-player protocols for boolean functions (as well as different boolean functions for different participants). The solution in[1] is based on trapdoor functions, and in [2] on the existence of an oblivious transfer primitive.

New Results. In this paper we show how to define and achieve fairness of any (not only boolean) function from strings to strings. This allows iteration and composition of protocols preserving fairness. In fact, we can achieve fairness for *any interactive probabilistic computation,* (i.e. games – to be defined in the journal version of this paper). The solution is based on the existence of an oblivious transfer primitive between every pair of players and a broadcast channel. The failure probability of the the protocol we propose is exponential while previously known was a polynomial.

We also propose a simpler definition of security for multi-player protocols which still implies previous definitions of privacy and correctness. [Kilian Micali Rogaway 90] have proposed independently another set of definitions of security. The relationship between the set of definitions has not been fully analyzed yet.

2 Conventions

Definition 1 • A *interactive* Turing machine is one equipped with a work-tape, input tape, random tape, output tape and several read-only and write-only communication tapes. The random tape contains an infinite sequence of random bits. *Flipping a coin* means reading the next new bit from the random tape.

- A *multi-player protocol* $\vec{P} = (P_1, P_2, \ldots, P_n)$ is a tuple of n interactive Turing machines where P_i starts up with x_i on its input tape and ends up with some output y_i on its output tape. We call $\vec{x} = (x_1, \ldots, x_n)$ the input vector to $\vec{P}$.

Inputs. We assume that the number n of players and identity i of each player P_i are included in its input. Initially coin-flips and inputs are independent. They may become correlated if their joint distribution became conditional on the information (say their sum) released by the player. Such coin-flips we call *relevant*. Players may want to keep them secret, to protect the privacy of their inputs. Other *irrelevant* coins flips may be released after the end of the protocol. The third type are *unused* coin-flips. They are kept, so that modifications of the protocol may use them and run with no extra random sources. The protocol must separate the three types before any communication starts and the unused flips must have at least constant density on the tape. We will treat the relevant coin-flips as part of the *complete* input, unless we talk of the *proper* input.

Outputs. Also, each player's output in non-faulty protocols consists of its input, relevant coin-flips, and only one more string $P(\vec{x})$, *common* to all players. This assumption does not limit generality, since we can always add one last step to any protocol $\vec{P}$ in which every player i uses a secret random string p_i and tells all other players the value of exclusive or: $y_i \oplus p_i$. Then, the common output is the concatenation of all $y_i \oplus p_i$. The new protocol is, clearly, equivalent to the old one and retains all its properties, like correctness, privacy, etc.

Notation 1 Let $\vec{x} = (x_1, \ldots, x_n)$ be the input vector to protocol $\vec{P} = (P_1, \ldots, P_n)$. Then, $\vec{P}(\vec{x})$ will denote the random variable which maps the (uniformly distributed) contents of random tapes α_i of the P_i's into the output vector $\vec{y} = (y_1, \ldots, y_n)$ where y_i is the output of machine P_i.

Let $F \subseteq \{1, \ldots, n\}$ be the set of (colluding) faulty players. An upper bound $t \geq |F|$ on their number is included in the inputs of all players. The inputs and outputs of faulty and non-faulty players we denote $\vec{x}_F$, $\vec{x}_{!F}$, $\vec{P}_F(\vec{x})$, $\vec{P}_{!F}(\vec{x})$, respectively.

Let us choose an arbitrary monotone unbounded *security* threshold $S(k) < k$ and call functions $k^{-S(k)/O(1)}$ *negligible*. If a family Y of random variables runs in expected polynomial time $\mathrm{E}_\alpha T_{Y_k(\alpha)} = k^{O(1)}$, we call it *samplable.*

Definition 2 A *test* is a probabilistic algorithm $t(\omega, y)$ which maps the tested string y into the result $\{\pm 1, 0\}$, using its internal coin-flips ω. Tests must run in expected polynomial time $\mathrm{E}_\omega T_{t(\omega,y)} = |y|^{O(1)}$. A test t *accepts* a family $Y_k(\alpha) \in \{0,1\}^k$ if its expected value $t_Y(k) = \mathrm{E}_{\omega,\alpha} t(\omega, Y_k(\alpha))$ is negligible. We call *indistinguishable*:

- two families of random variables, if every test accepts both or none of them.[1]
- protocols $\vec{P}, \vec{P}'$, if $\vec{P}(\vec{x})$, $\vec{P}'(\vec{x})$ are indistinguishable when $\vec{x}$ is generated by any samplable family of random variables.

2.1 Faulty Versions of Protocols

Versions of a protocol capture deviations from it by the faulty players.

Definition 3 A *version* of protocol P is any protocol P', with $P_{!F} = P'_{!F}$.

Note, that no restriction is put on P'_F. They may deviate from the P_F at any time and freely exchange messages among members of F. This raises two questions.

Question 1: How does a player become faulty and enter F?

Answer: We assume an adversary who points to a player and makes it faulty.

Question 2: How does such an adversary decide who to point to?

Answer: We consider two models for such adversary.

In the first model, the adversary, called the *static adversary*, chooses the set of faulty player before the beginning of the computation.

In the second model the adversary is called the *dynamic adversary*. In this model the adversary observes the broadcast messages and private inputs/communication of any (none at the start) players which already became faulty. Based on this information, the adversary may, at any time and repeatedly, choose new players to become faulty. Once a player is faulty it remains faulty and their number is limited by t.

2.2 Legal and Moral Faults

Some faults affect the input-output behavior of a protocol but, for trivial reasons, can never be prevented by the non-faulty players.

For example, players in F may choose to misrepresent their inputs x_F as x'_F and run P_F accordingly; also they may choose to replace their output y_F with entirely different strings y'_F. We refer to such faulty behavior as *immoral* but *legal*.

Definition 4 A *legal version* of a multi-player protocol $\vec{P} = (P_1, \ldots, P_n)$ is a protocol $\vec{P}' = (P'_1, \ldots, P'_n)$ where P'_i is identical to P_i except for

- Before any communication with non-faulty machines, the faulty players may pull together their inputs and random tapes and transform them arbitrarily.

[1]For non-samplable Y, Y' one should require negligibility of $t_Y - t_{Y'}$.

- Upon termination of $\vec{P}$, all $i \in F$ may pull together their inputs, outputs, and random tapes. Then the outputs of faulty players may be replaced by a function of this pool.

Note: The dynamic adversary in the legal version is active only during the input and output stages. In these stages he corrupts players choosing them on the basis of inputs (and at the end of outputs as well) of those players he previously corrupted.

When in majority, faulty players have other non-preventable ways to affect the protocol's input-output behavior. Namely, if players quit early, they can prevent the good players from completing the computation.

Definition 5 A *legal-minority* version of a multi-player protocol $\vec{P}$ is a legal version $\vec{P'}$ of a protocol identical to $\vec{P}$ except that

- At the start of the protocols the players broadcast whether their inputs are empty. If anyone's input is empty, the protocol is aborted and players output *error*.
- At any time $n - t$ players may broadcast "I am faulty". Then the protocol is aborted and non-faulty players append to their output the identity of the players who declared themselves faulty.

2.3 Robust and Fair Protocols

Definition 6 A protocol $\vec{P} = (P_1, \ldots, P_n)$ is *robust* (respectively *semi-robust*), if for every version $\vec{P'}$ of $\vec{P}$, there exists a legal (respectively legal-minority) version $\vec{P''}$, of $\vec{P}$, indistinguishable from $\vec{P'}$.

While robustness is a "complete" quality, semi-robustness requires additional feature: quit-fairness. It insures that interrupting the protocol does not give an unfair advantage in knowledge to the perpetrating majority.

In addition to players $1, \ldots, n$, we will speak of player 0 to mean the coalition of faulty players whose joint input is $x_0 = x_F$. (from here on i ranges from 0 to n).

For generality, we assume that not all output information has equal value. Some may be useless, as the players may somehow get it for free upon termination of the protocol $\vec{P}$. Suppose this free information for player i is $V_i(\vec{x})$. The function $\vec{V}$ may not even been known to i during the protocol, but could be known to the faulty players. (The reader may ignore this extra generality, assuming $\vec{V} = 0$.)

Let $\bar{\mu}_y(i)$ be the probability (over $\vec{x}, \vec{\alpha}$) of output $y = P(\vec{x})$ given $V_i(\vec{x})$ and x_i, and $\delta(i, \vec{x}, \vec{\alpha}) = \frac{1}{\bar{\mu}_{P(\vec{x})}} E_{y \neq P(\vec{x})}(\bar{\mu}_y) = \frac{1}{\bar{\mu}_{P(\vec{x})}} \sum_{y \neq P(\vec{x})} \frac{\bar{\mu}_y^2}{1 - \bar{\mu}_{P(\vec{x})}}$ be the ratio of the average (over y) probability of a wrong answer to the probability of a correct answer (from the point of view of player i).

Let $h_{\vec{x},i,t,\vec{\alpha}}$ be the history seen by player i upto step t on input $\vec{x}$, and coin tosses $\vec{\alpha}$.

Let $\mu_t(i,\vec{x},\vec{\alpha})$ denote the probability of the correct output $P(\vec{x})$ (taken over $\vec{x},\vec{\alpha}$) given $V_i(\vec{x})$ and $h_{\vec{x},i,t,\vec{\alpha}}$ (from the point of view of player i).

Let $r_t(i,\vec{x},\vec{\alpha}) = \frac{(1-\mu)}{\mu}$ be the ratio of the odds of wrong and correct values for $P(\vec{x})$ (from the point of view of player i)

Let $R_t(i,\vec{x})$ be the expectation (over $\vec{\alpha}$) of r_t and $D_t(i,\vec{x})$ be its standard deviation.

Definition 7 A protocol is *quit-fair* if

- for all $i,t,\vec{x},\vec{\alpha}$ either $\log\frac{r_{t+1}(i,\vec{x},\vec{\alpha})}{r_t(i,\vec{x},\vec{\alpha})} < \frac{1}{|\vec{x}|}$[2] or $h_{\vec{x},i,t,\vec{\alpha}} \in H_{\vec{x},i,t}$ where $H_{\vec{x},i,t}$ is a set of histories of exponentially small probability over α.
- $\frac{R_{t+1}(i,\vec{x})}{R_t(i,\vec{x})}$ does not depend on i.
- $D_t(i,\vec{x})$ is $O(\sqrt{\delta(i,\vec{x},\vec{\alpha})R_t(i,\vec{x})})$.

A protocol is *robust for minority* if it is semi-robust and quit-fair.

2.4 Stable Functions and Commitment Protocols

Definition 8 A function f is *stable* if $f(\vec{x'})$ is either nil or $f(\vec{x})$, for all $\vec{x}$ in its domain and $\vec{x'}$, s.t. $x'_{!F} = x_{!F}$.

Note 1 *Faulty players can not affect the value of stable functions by misrepresenting their inputs.*

By running a *commitment* protocol on inputs $\vec{x}$ we will transform any function f into a stable function f' (on possible outcomes $\vec{y}$ of the commitment protocol), such that $f'(\vec{y}) = f(\vec{x'})$ for some $\vec{x'}$, $x'_{!F} = x_{!F}$.

3 The Merits of the Definitions

Traditionally, several properties are required of a protocol such as privacy, correctness, independent choice of the inputs by faulty players, when a minority is faulty. And, additionally, quit-fairness, when majority is faulty.

All versions of robust protocol satisfy all these properties:

Proposition 1 *Any version of a robust protocol $\vec{P}$ satisfies the following properties:*

[2]This can be made arbitrarily small by padding the input $\vec{x}$.

- *Correctness:* In a legal version, by definition, the non-faulty players output $P_{!F}(\vec{x}')$, and $x'_{!F} = x_{!F}$. Now, since there exists a legal version which is indistinguishable from an illegal one and in that the good guys outputs are correct, we are practically guaranteed correctness for illegal version of $\vec{P}$.

- *Privacy:* Several definitions of privacy exist. We recall one of them and demonstrate it for a robust protocol. Let $VIEW_F$ be the random variable which takes on as value the entire history of communication and coin tosses as seen by the players in F. Call a protocol *private* if there is a polynomial time algorithm M, s.t. $(M(x_F, \alpha_F, y_F), \vec{x}, \vec{y})$ is indistinguishable from $(VIEW_F, \vec{x}, \vec{y})$.

  Now, any version $\vec{P}'$ of protocol $\vec{P}$, can be modified by making its faulty players to output the $VIEW_F$. There exists a legal version $\vec{P}''$ of $\vec{P}$, with an output distribution indistinguishable from $\vec{P}'$. In a legal version, the faulty players compute their output based only on their inputs/outputs for $\vec{P}$. It follows that $VIEW_F$ can be generated given only the inputs x_F and outputs y_F for $\vec{P}$ of a faulty coalition.

- *independent commitment to inputs:* By definition, in a legal version of the protocol the faulty players decide on which value x'_i to use independently of the values of non-faulty players. Since for every illegal version of the protocol, there exists a legal version with the same output distribution, the values that faulty players choose in the illegal version would have been chosen by faulty players in a legal version independently of non-faulty inputs.

Proposition 2 *Any version of a robust for minority protocol $\vec{P}$ satisfies privacy, correctness, independent commitment to inputs, and quit-fairness.*

Proof: Privacy and independent commitment to values are shown as above. The definition of correctness for a faulty majority is an extension of correctness in the faulty minority case. Namely, we allow non-faulty players to output the special "error" output when faulty players quit in the middle of the protocol. For this extended definition, the same argument used for correctness in above theorem will work. Fairness is guaranteed as part of the definition of robust for minority protocols.

Now previous theorems in the literature can be cast in this terminology:

Theorem 1 ([Ben-Or Goldwasser Wigderson 88, Ben-Or Rabin 89])
If $|F| < n/2$, any protocol can be modified into a robust one with same outputs.

Especially interesting is the case of $\vec{P}$ computing stable functions, since in all versions of robust protocols for a stable function, non-faulty outputs are the same.

4 Main Result: Robust for Minority Protocols

Theorem 2 (Main) *If an oblivious transfer primitive exists and a broadcast channel exists, any protocol can be modified into one robust for minority, with same outputs.*

Note 2 *No restriction is made on the number of faults in the theorem. The oblivious transfer condition has previously been shown necessary for a general protocol transformation preserving privacy for a majority of faults.*

In addition to players $1, \ldots, n$, we will speak of player 0 to mean the coalition of faulty players whose joint input is $x_0 = x_F$. (from here on i ranges from 0 to n).

The x_i's for player i are chosen at random with some (not necessarily easy computable) distribution. Recall that we assume the original protocol to compute one common output $P(\vec{x})$ (in addition to x_i and relevant coin flips). This is so since at the end of the original protocol each player can choose a random string p_i, and send all other players $y_i \oplus p_i$. The common output will the concatenation of all $p_i \oplus y_i$. Clearly, the same privacy properties hold. Thus, from here on we speak of a protocol to compute a single output.

The protocol consists of four stages: preprocessing, commitment, computation, and revelation.

Preprocessing

If the number of potential faults is in minority the preprocessing stage is skipped. If the number of potential faults is in majority, then first the entire network engages in a preprocessing phase, independent of the inputs. The outcome of the preprocessing phase is either *error* or the protocol proceeds to stages of commitment, computation and revelation. An *error* implies that the protocol is aborted. A majority of faulty players can always force an early abort, but their decision to cause an early abort is independent of the non-faulty players inputs.

Commitment

The commitment stage reduces the problem to computing a stable function $P(\vec{x})$. It also creates a sequence of (committed to but hidden) coin-flips α (each the sum mod2 of coin-flips of all users).

Computation

The computation stage reveals the sum (taken over $Z_2^{|P(\vec{x})|}$) of $P(\vec{x})$ with random password w (chosen based on α). Fairness is not an issue at this stage, because any player can (were the protocol interrupted) make this sum totally random by erasing her coin-flips.

Revelation

At the revelation stage w is revealed. Privacy is not an issue at this stage, since w has no information about the inputs (beyond what the function value reveals).

Let $\varepsilon = 1/|\vec{x}|$. The revelation protocol consists of $T < 2|w|$ *macrosteps* in which the protocol reveals next unused portion of α and interprets it as a vector $v_T \in Z_2^{|w|}$. It then reveals a sequence of ε^{-3} independent bits (one per *micro step*) $b_t(\alpha)$ chosen such that $b_t = (v_T \cdot w)$ (the inner product of v_T and w) with probability $1/2 + \varepsilon$. At the end of the macro-step the actual value of $(v_T \cdot w)$ is revealed.[3]

Clearly the logarithm of $r_t(i, \vec{x}, \vec{\alpha})$ (see Definition 7) cannot change by more than $O(\varepsilon)$ per micro-step. After going through ϵ^{-3} micro steps with an exponentially small probability the majority of the coin flips differs from $v_T \cdot w$. Thus, at the last step of the macro step when $v_T \cdot w$ is revealed, $r_{t+1} - r_t$ is changed negligibly uness this exponentially rare failure of majority has happened. This takes care of the first requirement of quit-fairness.

Assume for generality sake that at termination of the protocol player i may even be given an extra information $V_i(\vec{x})$. (The function $\vec{V}$ to be later handed out may not necessarily be known to the non-faulty players during the protocol but could be known to the faulty players).

One can easily show that R_t decreases with almost the same speed for all $\vec{x}, i, \vec{V}$. Indeed, let the computation stage output $y = P(\vec{x}) \oplus w$. At the outset of a macro-step T, let $S_T(y, \alpha)$ be the set of w' with $(v_{T'} \cdot w') = (v_{T'} \cdot w)$ for all $T' < T$. Let $p(Y)$ be the probability at the outset (over $\vec{x}$ with given x_i and $V_i(\vec{x})$) of $y \oplus P(\vec{x}) \in Y$. Then at the end of macrostep T, $r_t = \frac{1-\mu_t}{\mu_t} = \frac{(p(S_T)-p(w))}{p(w)}$.

Each w' has a 2^{-T} chance to satisfy all T of the above randomly chosen $v_T \cdot w$ boolean linear equations and so fall in S_T. Moreover, each u falls in or out of S_T pairwise independently of any $u' \not\in \{u, u \oplus w\}$. Thus, the expectation at the end of a macro step of $p(S_T) - p(w)$ (over α) is $(1 - p(w))/2^T$, and therefore the expected value of $\frac{1-\mu_t}{\mu_t}$ is $\frac{(1-p(w))}{p(w) \cdot 2^T}$. No change in the expected value occurs at a micro step. Thus R_t is independent of i and the second requirement of quit-fairness is satisfied.

The standard deviation of $p(S_T) - p(w)$ is smaller than the square root of the mean of $p(\{a\})$ by a factor of $2^{-T/2}/O(1)$, and therefore $D_T \leq O(\sqrt{\delta R_T})$. (Recall that δ (in the definition of quit-fairness) is the ratio of the average probability of wrong answer to correct answer.)[4]

[3]The purpose of not revealing $(v_T \cdot w)$ immediately is to assure that by quitting early the faulty players can only receive one coin flip more than non-faulty ones toward the value of $(v_T \cdot w)$. After $2|w|$ macrosteps, w itself can be revealed.

Sometimes the faulty coalition can be restricted to a polynomial number of possible combinations (known to all parties). Also the parties may be confident at the start that their inputs are random and completely secret. Then a more sophisticated procedure could be used to discriminate against possible coalitions, which "know too much". We ignore this issue for now.

[4]The fairness requirement does not prevent erratic behavior at the end of the protocol, thus in special cases when it is detectable that the players doubts are concentrated on a logarithmic number of outputs we can do better by tossing a cube of all possible answers slightly biases toward the correct one.

5 How to Use the Oblivious Transfer Primitive

5.1 The Oblivious Transfer Assumption

We assume that every two players can perform an oblivious transfer.

An *oblivious transfer* [Rabin, Blum, Fischer Micali Rackoff, Even Goldreich Lempel 82] between two players A and B denoted by $\frac{1}{2} - OT^{AB}(b_0, b_1, c)$ is a process by which player A who has bits b_0,b_1 transfers bit b_c to player B, where c is chosen by B. The transfer is done obliviously: player A can not distinguish between the case player B received b_0 and the case B received b_1; player B can not distinguish between $b_{\bar{c}} = 0$ and $b_{\bar{c}} = 1$.[5] An oblivious transfer of one bit, means that the other bit is 0 and the (random) order of bits is revealed after the transfer is performed.

An oblivious transfer can be implemented if trapdoor functions exist and A and B are computationally bounded [Even Goldreich Lempel 82]; or can be derived from the existence of noisy channel and other physical means even in the presence of infinitely powerful A and B [Crepeau Kilian 88].

We show how to use an $\frac{1}{2} - OT^{AB}(b_0, b_1, c)$ protocol between every pair of players A and B to implement the preprocessing, commitment, and computation stages specified in section 4. Thus, we start with a legal protocol and transform it to one which is robust.

Many of the ideas in the transformation which lead to semi-robustness property (not quit-fairness) are similar to ones used in previous results of [Goldreich Micali Wigderson 87, Galil Haber Yung 87, Ben Or Goldwasser Wigderson 88, Kilian 88, Beaver Goldwasser 89].

In [Beaver Goldwasser 89a] a version of a protocol achieving semi-robustness for boolena functions based on the existence of trapdoor functions, is described. Here, we describe a protocol based on the existence of oblivious transfer, in which the error probability is improved from the previously known 1/polynomial [Beaver Goldwasser 89b] to 1/exponential.

We let t be the number of potential adversaries, k denote the security parameter.

5.2 Preprocessing Stage

5.2.1 Global Commitment and Decommitment

Each player globally commits to a library of 0's and 1's. A global commit has properties similar to a commit between two players. In fact, many of the the ideas are similar to the two party bit-commitment of [Kilian 88].

In particular, preprocess-global-commit(A,v,J) is a protocol for player A to globally commit to a bit v such that if the preprocessing stage is completed successfuly, then there exists a unique value $\hat{v}$ associated with J such that

[5]This form of oblivious transfer was shown equivalent to the original one proposed by Rabin.

- $\hat{v} = v$ if A is non-faulty.
- At any time player A can decommit $\hat{v}$ (or $\hat{v}_1 \oplus \hat{v}_2$ where $\hat{v}_1$, $\hat{v}_2$ are two previously commited bits) to a subset S of players such that either all non-faulty players in S will receive the correct value, or all non-faulty players will broadcast that A is faulty, or an exponentially rare event will happen. In the case of decommiting $\hat{v}_1 \oplus \hat{v}_2$, the privacy of $\hat{v}_1$ remains intact for the entire network.
- If non-faulty A committed a v randomly chosen in $\{0,1\}$ then the probability that the faulty players guess the value of v before it is decommitted to one of them is negligible.

This is achieved as follows.
Notation: We let $rep(v)$ be a set of k boolean vectors $\{\overline{v}_i\}, 1 \leq i \leq k$ such that for each $\overline{v}_i = (v_{i1}, ..., v_{ik})$ the $\oplus_j v_{ij} = v$. We say that $rep(v) = (\vec{v}_1, ..., \vec{v}_k)$ is *invalid* if for some s,t the $\oplus_j v_{sj} \neq \oplus_j v_{tj}$. To choose a $rep(v)$ at random means to pick the $v_{ij} \in \{0,1\}$ as above at random. To broadcast or obliviously transfer $rep(v)$ means to broadcast or run oblivious transfer each of the v_{ij}'s. We let the function $all(\{b_i\}) = b_1$ if for all $i, j, b_i = b_j$, otherwise it assumes an *error* value.

Preprocess Global Commit(A,v,J):

Step 1. For $1 \leq i \leq k^a$: A chooses $v_i^L, v_i^R \in \{0,1\}$ at random and sets $v_i = v_i^L \oplus v_i^R$. A chooses a $rep(v_i^L) = (\overline{v}_1^{iL}, ..., \overline{v}_k^{iL})$ and $rep(v_i^r) = (\overline{v}_1^{iR}, ..., \overline{v}_k^{iR})$ at random; For every player B, A oblivious transfers to B $rep(v_i^L)$ and $rep(v_i^R)$.[6]

Step 2. The network chooses at random[7]a set I containing half of the i's. For all $i \notin I$, A broadcasts $rep(v_i^L)$ and $rep(v_i^R)$. If for some i, d player B gets an invalid $rep(v_i^d)$ or inconsistent with information B received then B broadcasts a complaint and the protocol is aborted.[8] Otherwise, A broadcasts a set $\{c_i = v_i^R \oplus v_i^L \oplus v | i \in I\}$.

Step 3.[9] Repeat for every player B k times: B broadcasts indexes i, j chosen at random in I; A broadcasts $b = v_i^L \oplus v_j^L$; B chooses $d \in \{L, R\}$ at random; A broadcasts $rep(v_i^d)$ and $rep(v_j^d)$; if $b \neq v_i^d \oplus c_i \oplus v_j^d \oplus c_j$, then B broadcasts a complaint and the protocol is aborted, otherwise $I = I - \{i, j\}$.

Step 4. Each player stores $I, \{c_i, i \in I\}$ and the information he received during the global commit of player A to $v_i^L, v_i^R, i \in I$ in $BIT-COMMIT(A,J)$[10] and the Jth bit is declared committed. (The value of this bit is $all(v_i^R \oplus c_i \oplus v_i^l), i \in I)$).

[6]each bits v_i is represented by a pair v_i^R, v_i^L such that $v_i = v_i^R \oplus v_i^L$.

[7]It suffices that players alternate in choosing elements in I

[8]if the protocol is aborted during an execution of preprocess-global-commit, then all non-faulty players output error.

[9]In this step A proves to each player B in turn that for all remaining $i, j \in I$ $v_i \oplus c_i = v_j \oplus c_j$.

[10]Clearly each player may have received different bits during the oblivious transfer and thus has different information.

At the outset of the preprocessing stage, every player runs the protocol preprocess-global-commit for a sufficient number of values $v = 0$ and $v = 1$ as will be necessary for A to commit bits during the life time of the protocol.

During the protocol player A globally commits to bit v by broadcasting index J, such that the bit committed during the preprocessing global commit stored in $BIT - COMMIT(A, J)$ is m. Once an index J is broadcast it is never reused.

To decommit to a subset S of the players, a committed bit stored in $BIT - COMMIT(A, J)$, A runs the following protocol.

Global Decommit(A,S,J):

Let v be the bit committed in $BIT - COMMIT(A, J)$ and S the subset to which it should be decommitted.

Step 1: A sends in private to each player in S, for all $i \in I$, $rep(v_i^R)$ and $rep(v_i^L)$. Players in S set $v = c_i \oplus v_i^R \oplus v_i^L$ for the smallest $i \in I$.

Step 2: If any player $B \in S$ gets for some i, d an invalid $rep(v_i^d)$ or inconsistent with information B received during the oblivious transfer stage, then B broadcasts a request that player A should broadcast $rep(v_i^R), rep(v_i^L)$ for all $i \in I$.

Step 3: If any player C detects that the information A broadcasts in step 2 is "inconsistent" or invalid, then C broadcasts that A is faulty, otherwise the value of v is taken to be the bit $c_i \oplus v_i^L \oplus v_i^R$ where v_i^L, v_i^R are defined by the information which A has broadcast at step 2.[11]

5.2.2 Decommitting Sums of Globally Committed Bits

During the protocol player A will need to prove that various bits globally committed are the same. Let v and u be two previously globally committed bits stored in $BIT - COMMIT(A, v, J_v)$ and $BIT - COMMIT(A, u, J_u)$.
Recall: Bit v has associated with it $I_v, \{c_{vi} | i \in I_v\}$, $rep(v_i^L), rep(v_i^R)$ for all $i \in I_v$, and bit u has associated with it $I_u, \{c_{ui} | i \in I_u\}$, $rep(u_i^L), rep(u_i^R)$ for all $i \in I_u$.

Protocol Prove-Equality(A,u, v)

Repeat for every player B k times: B broadcasts indexes $i \in I_v$ and $j \in I_u$ chosen at random; A broadcasts $b = v_i^L \oplus u_j^L$; B chooses $d \in \{L, R\}$ at random; A broadcasts $rep(v_i^d)$ and $rep(u_i^d)$ (if any C finds these invalid, then C broadcasts that A is faulty); if $b \neq v_i^d \oplus c_{vi} \oplus u_j^d \oplus c_{uj}$, then every player broadcasts that A is faulty, otherwise every player updates I_v to be $I_v - \{i\}$ and I_u to be $I_u - \{j\}$.

In fact, general properties of data globally committed can be proven in **zero-knowledge** using the protocols of [Kilian 89, Ben-Or et al [4]]. We chose the parameter a in the preprocess-global-commit protocol so to allow repeated zero-knowledge proofs about globally committed bits.

[11]By the properties of our global commit protocol all non-faulty players will either declare A faulty or agree that a bit of the same value has been decommitted.

5.2.3 Private Communication Lines

During the protocol players A and B will need to privately communicate and yet be able to prove to other players that the messages they send privately were computed correctly with respect to their committed inputs and previously received messages.

If encryption functions were available this would present no problem, however we only have the ability to perform oblivious transfers between every two players.

Thus, in the preprocessing stage every pair of players prepare and globally commit to a supply of 0's and 1's known to A and B alone which both can globally decommit. These bits will be used later for private communication.

This is done by running the following protocol.

Protocol Preprocess-Private-Communication(A,B,J)

A randomly chooses $b \in \{0,1\}$ and runs an identical protocol to $Preprocess - Global - Commit(A, b, J)$ with the exception that the information stored normally in $BIT - COMMIT(A, J)$ is stored in $PRIVATE - COMMIT(A, B, J)$. Next A decommits b to player B by running $Global - Decommit(A, \{B\}, J)$; (Note now that both A and B can decommit the bit store in $PRIVATE - COMM(A, B, J)$.)

During the protocol A sends private message $m = m_1, ..., m_k$ to player B by broadcasting indexes J_i such that for every $i = 1, ..., k$ the value of the bit committed in $PRIVATE - COMM(A, B, J_i)$ is m_i. (Once an index J is broadcast it is never reused.)

5.2.4 Global Oblivious Transfers

During the protocol every pair of players (A,B) will need to engage in an oblivious transfer in such a way that A and B can prove to to the rest of the network that indeed they have fed the correct inputs to the oblivious transfer process, and have received claimed outputs.

This is achieved by having every pair of players (A, B) prepare a supply of oblivious transfers in which the inputs and the outputs have been globally committed.

Let $i \in \{0,1\}^3$. Say that an oblivious transfer is of type $i = i_0 i_1 i_2$ if $b_0 = i_0$, $b_1 = i_1$ and $c = i_2$. To prepare at least L oblivious transfers of each type $i \in \{0,1\}^3$, every pair of players (A, B) execute the following protocol $O(L + k)$ times.

Protocol Preprocess-Oblivious-Transfer(A,B, J)

Step 1. For $j = 1, ..., k^a$: Player B globally commits to c^j randomly chosen in $\{0,1\}$; Player A globally commits to b_0^j, b_1^j, r^j randomly chosen in $\{0,1\}$. Players A and B run an $OT^{AB}(b_0^j \oplus r^j, b_1^j \oplus r^j, c^j)$; Player B globally commits to $\hat{r}^j$ which he received as a result of the oblivious transfer. (Clearly, if A and B are non-faulty, $\hat{r}^j$ should equal $b_{c^j}^j \oplus r^j$).

Step 2. The network chooses at random [12] a set I of half of the j's. For all $j \notin I$, A globally decommits b_0^j, b_1^j, r^j to the entire network; B globally decommits c^j and $\hat{r}^j$ to the network. If any player complains during these global commits or $r^j \oplus b_{c^j}^j \neq \hat{r}^j$, the protocol is aborted. Otherwise, player B globally commits to randomly chosen $c \in \{0,1\}$, broadcasts set $I_1 \subset I$ such that $I_1 = \{j | c^j = c\}$, and proves that $c^j = c$ iff $j \in I_1$ (using the Prove-Equality procedure defined above.) Player A globally commits to randomly chosen $b_0, b_1, r \in \{0,1\}$, broadcasts set $I_2 \subset I_1$ such that $I_2 = \{j | b_0^j = b_0, b_1^j = b_1, r^j = r\}$, and proves that $b_0^j = b_0$, $b_1^j = b_1$, $r^j = r$ iff $j \in I_2$ (using the Prove-Equality procedure). If for some $i, j \in I_2$ $\hat{r}^j \neq \hat{r}^i$ A broadcasts a complaint and the protocol is aborted, otherwise A globally commits to $\hat{r}$ such that $\hat{r} = \hat{r}^j$, for all $j \in I_2$ and proves this fact (using the Prove-Equality procedure). Each player stores I_2 and the information obtained during the global commits of $b_0^j, b_1^j, c^j, r^j, \hat{r}^j$, for $j \in I_2$ in its copy of $OT-COMMIT(A,B,J)$.

During the computation stage when A and B need to engage in an oblivious transfer with parameters B_0, B_1 known to A and parameter C known to B they run the following protocol.

Protocol Global-Oblivious-Transfer(A, B, B_0, B_1, C)

B selects a J such that the set $\{b_0^j, b_1^j, c^j, r^j, \hat{r}^j, j \in I_2\}$ stored in $OT-COMMIT(A,B,J)$ is such that $c^j = C$ for all $j \in I_2$ (a fact B proves to the network), and broadcasts J to the network. If $B_0 = b_0^j$ and $B_1 = b_1^j$ for all $j \in I_2$ (a fact A proves to the network), A decommits $r^j, j \in I_2$ to B, else J is cast out and the step is repeated. B sets $B_C = all(\{r^s \oplus \hat{r}^s, s \in I_2\})$.

5.3 Input Commitment and Computation Stages

At this stage every player A needs to globally commit to its input x_A and a sequence of coin flips α_A.

Let $x_A\alpha_A = y_A^1 ... y_A^m$ (in binary). Player A broadcasts[13] indexes $J_1, ..., J_k$ such that the bit committed in $GLOBAL-COMMIT(A, J_i)$ is y_A^i.

Set $\alpha = \sum_{\text{player } A} \alpha_A \bmod 2$.

5.3.1 Computation

Let C_P be the arithmetic circuit over field of elements F computing the legal protocol P (assuming that P has already been modified to output single output to all players). Let $\gamma_B \in F$ be a unique element associated with player B.

In order to allow the network to compute with inputs x_A (and α_a), player A secret shares x_A (and α_a) as follows. A selects a random polynomial p_{x_A} of degree t such

[12]It suffices that players alternate in choosing elements in I

[13]once an index is broadcast it will never be reused.

that $p_{x_A}(0) = x_A$. For every player B, A globally commits to $p_{x_A}(\gamma_B)$ and privately communicates $p_{x_A}(\gamma_B)$ to B.

Note now that because of the properties of PRIVATE-COMM every player B now knows $p_{x_A}(\gamma_B)$, and every other player has received a global commitment to the value of $p_{x_A}(\gamma_B)$ which can be decommitted either by A and or B.

A now proves in zero-knowledge to the network that the values privately sent $\{p_{x_A}(\gamma_B)\}_B$ interpolate to a unique t degree polynomial whose free term is x_A.

The arithmetic circuit C_P has two types of gates: addition $(+)$, and multiplication $(\times)$ over the finite field F (scalar multiplication is a trivial extension of $+$ gate).

The circuit is evaluated in a gate by gate fashion. The invariant during the computation stage is that each player holds a share of all inputs to the next gate to be computed, which is globally committed.

Suppose the inputs to a $+$ gate are u and v. Every player A holds $P_u(\gamma_A)$ and $P_v(\gamma_A)$ (where P_u and P_v are random polynomials of degree t with free term u and v respectively). To compute a share of the output $u+v$, player A computes $P_{u+v}(\gamma_A) = P_u(\gamma_A) + P_v(\gamma_A)$. A globally commits to $P_{u+v}(\gamma_A)$.

Suppose the inputs to a $\times$ gate are u and v. Computing $P_{u\times v}(\gamma_A)$ (where $P_{u\times v}$ is a random polynomial of degree t whose free term is $u\times v$.) can be reduced to the problem of every pair of players (A,B) computing semi-robustly a two-player function on the shares they hold $P_u(\gamma_A)$ and $P_v(\gamma_B)$, (see [Galil Haber Yung 87, Van Dem Graaph etal. 87, Beaver Goldwasser 88]). To compute a two-player function semi-robustly has been reduced to two-player oblivious transfer in [Kilian 88]. Instead of the two-player oblivious tarnsfert called for in [Kilian 88]'s construction, the $Commit-Oblivious-Transfer(A, B, ...)$ protocol which was set up in the preprocessing stage is used.

Every message of the player sent while computing the $\times$ gate must be accompanied by a zero-knowledge proof that it has been computed and sent correctly with respect to the inputs globally committed and the messages previously received from other players both in private and by broadcast. This is possible as all private messages sent during the commitment and computation stage have been globally committed (as all these messages were sent using the private communication lines set up in preprocess-private-communication.)

5.4 Acknowledgements

We are gratelful to Mihir Bellare, Joe Kilian, and Silvio Micali for very useful discussion.

References

[1] D. Beaver, S. Goldwasser. *Multi Party Fault Tolerant Computation with Faulty Majority*, proceedings of Crypto89, Santa Barbara, CA, August 1989.

[2] D. Beaver, S. Goldwasser. *Multi Party Fault Tolerant Computation with Faulty Majority Based on Oblivious Transfer*, proceedings of FOCS89, Duke, NC, October 1989, pp. 468-473.

[3] M. Ben-Or, S. Goldwasser, A. Wigderson. *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation.* Proc. of 20th *STOC* 1988, pp. 1-10.

[4] Ben-Or, Michael, Oded Goldreich, Shafi Goldwasser, Johan Hastad, Joe Kilian, Silvio Micali, Phillip Rogaway, "IP is in Zero-Knowledge," Proceedings, *Advances in Cryptology,* Crypto 1988.

[5] Rabin, T. and M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." Proc. of 21st STOC, ACM, 1989.

[6] Blakely,T.. Security Proofs for Information Protection Systems. Proceedings of the *1980 Symposium on Security and Privacy*, IEEE Computer Society Press, NY, pp. 79-88, 1981.

[7] Brassard, Gilles, Claude Crépeau, and David Chaum, "Minimum Disclosure Proofs of Knowledge," manuscript.

[8] Brassard, Gilles, Claude Crépeau, and Jean-Marc Robert. "Information Theoritic Reductions Among Disclosure Problems," *Proceedings of the* 27th *FOCS*, IEEE, 1986, 168–173.

[9] E. Brickell, D. Chaum, I. Damgaard, J. van de Graaf. Gradual and Verifiable Release of A Secret. *CRYPTO* 1987.

[10] D. Chaum, C. Crepeau, I. Damgaard. Multiparty Unconditionally Secure Protocols. Proc. of 20th *STOC* 1988, pp. 11-19.

[11] Chaum, David, Ivan Damgard, and Jeroen van de Graaf. "Multiparty Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output," *Proceedings of CRYPTO '85*, Springer-Verlag, 1986, 477–488.

[12] R. Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty. *STOC* 1986.

[13] Cohen, Fischer. A Robust and Verifiable Cryptographically Secure Election. *FOCS* 1985.

[14] C. Crepeau and J. Kilian. Achieving Oblivious Transfer Using Weakened Security Assumptions. *FOCS* 1988.

[15] Crépeau Claude, "On the Equivalence of Two Types of Oblivious Transfer", Crypto87.

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

[17] B. Chor, M. Rabin. Achieving Independence in Logarithmic Number of Rounds. *PODC* 1986.

[18] Even S., Goldreich O., and A. Lempel, *A Randomized Protocol for Signing Contracts*, CACM, vol. 28, no. 6, 1985, pp. 637-647.

[19] Fischer M., Micali S., and Rackoff C. *Oblivious Transfer Based in Quadratic Residuosity*, Unpublished.

[20] Z.Galil, S.Haber, M.Yung. Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public Key Model. Proc. *CRYPTO* 1987.

[21] O. Goldreich, S. Micali, A. Wigderson. How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority. Proc. of 19th *STOC* 1987, pp. 218-229.

[22] Goldreich, O., Vainish, R. "How to Solve any Protocol Problem: An Efficiency Improvement", Crypto 87.

[23] Goldwasser, Micali, Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. of Comp* 1989.

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

[25] Impagliazzo, Russell and Moti Yung, "Direct Minimum Knowledge Computations," Proceedings, *Advances in Cryptology,* Crypto 1987.

[26] Kilian, Joe, "On The Power of Oblivious Transfer," *Proceedings of the* 20th *STOC*, ACM, 1988, pp. 20-29. Also appeared in *Uses of Randomness In Algorithms and Protocols*, An ACM Distinguished Dissertation 1989.

[27] J. Kilian. S. Micali. P. Rogaway *Security Definitions for Multi Party Protocols.* In Preparation.

[28] Micali Luby Rackoff 83. *The Miraculous Exchange of a Secret bit*, Proc. of *FOCS* 1983.

[29] A. Shamir. How to Share a Secret. *CACM 22*, 612-613, 1979.

[30] Yao, Andrew C. "Protocols for Secure Computations," *Proceedings of the* 23rd *FOCS*, IEEE, 1982, 160–164.

[31] Yao, Andrew C. "How to Generate and Exchange Secrets," *Proceedings of the* 27th *FOCS*, IEEE, 1986, 162–167.