Multiparty Protocols Tolerating Half Faulty Processors

Donald Beaver*
Aiken Computation Laboratory
Harvard University

Abstract

We show that a complete broadcast network of n processors can evaluate any function $f(x_1, \ldots, x_n)$ at private inputs supplied by each processor, revealing no information other than the result of the function, while tolerating up to t maliciously faulty parties for $2t < n$. This improves the previous bound of $3t < n$ on the tolerable number of faults [BGW88, CCD88]. We demonstrate a resilient method to multiply secretly shared values without using unproven cryptographic assumptions. The crux of our method is a new, non-cryptographic zero-knowledge technique which extends verifiable secret sharing to allow proofs based on secretly shared values. Under this method, a single party can secretly share values $v_1, \ldots, v_m$ along with another secret $w = P(v_1, \ldots, v_m)$, where P is any polynomial size circuit; and she can prove to all other parties that $w = P(v_1, \ldots, v_m)$, without revealing w or any other information. Our protocols allow an exponentially small chance of error, but are provably optimal in their resilience against Byzantine faults. Furthermore, our solutions use operations over exponentially large fields, greatly reducing the amount of interaction necessary for computing natural functions.

1 Introduction

For fault tolerance and security in a distributed system, it is desirable to be able to execute secure multiparty protocols. Such protocols allow a system to evaluate a function f at private inputs $x_1, \ldots, x_n$, each supplied by a member of the system, and to reveal the result to a designated recipient. Nothing about the private information held by each participant is revealed, other than what could otherwise be computed solely from the value of the function.

A secret ballot, for example, requires a tally of private votes, each of which must be restricted to a 0-1 value. The system must produce the tally without revealing any individual votes, and without allowing any misbehavior to affect the result.

*This research was supported in part under NSF grant CCR-870-4513.

G. Brassard (Ed.): Advances in Cryptology - CRYPTO '89, LNCS 435, pp. 560-572, 1990.

In the case of a unanimous vote, for example, information about the inputs might inevitably be revealed by the value of the output; but no additional information is gained during the run of a protocol.

In [BGW88,CCD88] methods for secure, multiparty computations without using any unproven cryptographic assumptions were given. Those methods allowed for up to t Byzantine faults, where $3t < n$. It can be shown that, for information-theoretic, errorless security, those results were optimal. It was not clear, however, whether $2t < n$ could be achieved at the cost of allowing a negligible chance of error. For larger numbers of faults, of course, the faulty players constitute a majority and it becomes impossible merely to share a secret.

A natural question to ask, then, is whether multiparty computations can be achieved when $2t < n \leq 3t$. Recently [Rab88] made initial progress in this direction by demonstrating a method for verifiable secret sharing for $2t < n$, using a broadcast network, and having a small probability of error. Earlier methods for verifiable secret sharing with a faulty minority required cryptographic assumptions [CGMA85]. The extension to performing computations for $2t < n$, however, has remained open until now.

We present new and efficient methods for performing multiparty computations that tolerate $2t < n$. Our techniques utilize verifiable secret sharing for $2t < n$, and allow the field used for secret sharing to be of exponential size. Because we can simulate large-field arithmetic operations quickly and directly while alternative methods use bit-simulations or small-field arithmetic, our protocols have the practical advantage of using fewer rounds of communication for many natural functions.

Note that our methods are secure against adversaries with unbounded resources, while at the same time requiring only polynomial time to execute.

Methods for multiparty protocols tolerating a faulty minority were independently discovered by Ben-Or [BR89] and Kilian [Kil]. Whereas [BR89] use boolean circuit simulation (and implicitly require that the field used for secret sharing be polynomial size), our methods use a different and broader technique which allows direct computations in fields of exponential size, and are superior in terms of efficiency and flexibility.

In addition to proving that multiparty computations are possible when the honest players hold the majority, we show how zero-knowledge proofs can be implemented using a distributed system. Our new method for zero-knowledge proofs requires no cryptographic assumptions and allows up to half the participants to be faulty.

1.1 Secret Sharing and Computation

In Shamir's method for secret sharing [Sha79], a player Alice distributes a secret value s as follows. Let $p > n$ be prime and consider the field Z_p. Alice chooses t numbers $a_1, \ldots, a_t$ at random mod p, and sets $g(x) = a_t x^t + \cdots a_1 x + s$. Then she tells the "piece" $g(i)$ to player i, where i is a nonzero identifier. (In its most general form, secret sharing uses a unique evaluation point $\alpha_i \neq 0$ for each player, but for clarity and simplicity we take $\alpha_i = i$.) Clearly, $t + 1$ pieces are required to

reconstruct a secret, and t or fewer pieces provide no information about the secret.

The design of noncryptographic multiparty protocols is greatly facilitated by dividing the goal into three stages. First, all private inputs to the protocol are secretly shared. The second stage produces a new secretly shared value w which is equal to $f(x_1, \ldots, x_n)$. This stage is known as *oblivious circuit evaluation* or *secret computation*. Finally, w is reconstructed and revealed to the proper recipients.

This modularization ensures independence of the inputs (faulty players choose their inputs independently of the inputs of nonfaulty players) and makes analysis of privacy and correctness easier. It also provides a means to link various multiparty protocols, using the secret outputs created by one protocol as inputs to the next protocol.

Thus, given methods to share secrets and to verify that they have been shared correctly, we can focus our attention on creating new secretly shared values based on old ones.

1.2 Results

Two basic protocols form the basis for building protocols to evaluate arbitrary arithmetic circuits. Secret addition provides each player with a piece of a new secret whose value is the sum of earlier secrets. Similarly, secret multiplication creates a new secret whose value is the product of earlier secrets. These two fundamental tools provide the means to construct a protocol to perform any secure and fault-tolerant multiparty computation.

In the sequel, let E be a finite field of size $2^{O(n)}$, and consider a complete, broadcast network with private channels.

Our main result is based on the following theorem, which reduces the problem of secret multiplication to that of secret addition, even when $2t < n$. By *t-resilient* we mean a protocol that preserves correctness and privacy, in the face of up to t Byzantine faults coordinated by a dynamic adversary.

Theorem 1 *For $2t < n$, if there exists a t-resilient protocol to add secrets, then there exists a t-resilient protocol to multiply secrets.*

The verifiable secret sharing methods of [Rab88,BR89] are not directly suitable for secret addition unless the original secrets are shared by a single party. We extend those methods, however, with some additional techniques that make it possible to add secrets regardless of their origin.

Thus, the supposition of Theorem 1 is satisfied, showing that multiplication is indeed possible for $2t < n$. As a direct consequence we have our primary result:

Theorem 2 *(Main Result.) Let $\{C_n\}$ be a uniform polynomial size circuit family where each C_n computes a function f_n over a field E. For $2t < n$, there exists a t-resilient protocol to evaluate $f_n(x_1, \ldots, x_n)$, preserving the privacy of the inputs.*

Theorem 1 and Theorem 2 are based on a new and powerful technique whereby a single player shares secrets u, v, and w and can prove to the network that the value

of w is indeed the product of the values of u and v, without revealing any other information. This non-cryptographic zero-knowledge technique generalizes to allow a player to prove that a secret w holds the value $P(v_1, \ldots, v_m)$, for any function P computed by a polynomial size circuit C_P, without revealing any additional information about w or about any of the v_i. In particular, any player in the network can give a zero-knowledge proof to any other player; the network simply reconstructs w for the verifier. This result has value of its own right.

Theorem 3 *(Zero-Knowledge.) Suppose that Alice knows the values $v_1, \ldots, v_m$, which have been secretly shared. Let P be a publicly known function of m arguments which is described by a polynomial size circuit C_P. For $2t < n$, there exists a t-resilient protocol by which Alice can share a secret w whose value is $P(v_1, \ldots, v_m)$, and by which she can prove to all other players that $w = P(v_1, \ldots, v_m)$, without revealing w or any other information.*

Lemma 4 *Any language in IP is provable in zero-knowledge in the presence of a network as described above, without using unproven cryptographic assumptions.*

Our protocols are time and communication efficient. All of the protocols in this paper require message sizes that are polynomial in the number n of players, the size of the arithmetic circuit C_f to be simulated, and a unary security parameter k ensuring correctness and privacy with probability at least $1 - \frac{1}{2^k}$. Let T be the number of rounds needed for Verifiable Secret Sharing (see section 2.2). The number of rounds of interaction is proportional to dT, where d is the depth of an algebraic circuit C_f with polynomial fanin $+$-gates and bounded fanin $\times$-gates over a field of size up to $2^{O(n)}$. The methods of [BB88] can be applied to reduce the number of rounds to $O(dT/\log n)$, or even to $O(T)$, the latter at a possible increase in message size.

Furthermore, we use no unproven cryptographic assumptions.

In section 2 we introduce some definitions and discuss protocol properties. Section 3 describes the overall design for multiparty protocols. In section 4 we discuss how secrets can be added, and in section 5 we give the novel result that if secrets can be added then secrets can be multiplied (Theorem 1). Using our methods for secret addition, secret multiplication, and zero-knowledge proofs, we show the main theorem in section 6. Finally, non-cryptographic zero-knowledge proofs of properties of secrets (Theorem 3) are discussed in section 7.

An earlier version of this paper appeared as [Bea88b].

2 Preliminaries

In this section we list some definitions, including desired properties of verifiable secret sharing and of protocols to compute functions. For purposes of this extended abstract, however, proofs of the formal properties listed here will not be given in detail.

2.1 Protocol Properties

Definition 1 *A protocol is a set of n interactive Turing machines* $\{ M_i \}$ *with inputs* $\{ (x_i, 1^k) \}$, *communication tapes* $\{ t_{ij} \}$, *broadcast tapes* $\{ b_i \}$, *and outputs* $\{ y_i \}$, *where k is a security parameter expressed in unary.*

Our protocols will be resilient against Byzantine faults coordated by a dynamic adversary. That is, we allow an adversary to choose to substitute M_i' for an arbitrary M_i at any time, based on information held by processors whose programs have already been corrupted. The adversary is restricted to t corruptions, where $2t < n$.

In order to consider the correctness and other properties of the protocol, we let P_i denote the probability distribution according to which x_i is selected. Let $f : P_1 \times P_2 \times ... \times P_n \rightarrow \{0,1\}$ be the function we want to compute. If T is a coalition of at most t processors, $w = f_T(x_1, ..., x_n)$ denotes the correct value of the function, where f_T is the function obtained from f by substituting default inputs for the processors in T who are disqualified during the Input stage.

Let VIEW_i be a random variable describing a transcript of everything written on the input, output, and communication tapes that are readable or writable by processor i. For a coalition T, let VIEW_T be the set of views of all processors in T.

1. (Independence of Inputs) During the Input stage, faulty processors share an input value x_j which is independent of those shared by the good processors.

2. (Privacy) For any coalition T of at most t faulty processors, no additional information is obtained from the protocol. More formally, for any input values $\vec{x}_T$ held by the coalition and for any output w, there is a machine which (probabilistically) generates a string V, whose distribution is nearly identical with that on VIEW_T :

$$\sum_{\text{VIEW}_T} |\Pr[M(\vec{x}_T, w) = \text{VIEW}_T] - \Pr[\text{VIEW}_T \mid \vec{x}_T]| < \frac{1}{2^k}.$$

3. (Fairness) All good players can compute the correct output whenever the faulty players can:

$$\Pr[\, y_j = w \mid j \text{ is faulty}] \leq \Pr[\, y_i = w \mid i \text{ is non-faulty}].$$

4. (Correctness) All good players compute the correct output with high probability:

$$\Pr[\, y_i = w \mid i \text{ is non-faulty}] \geq 1 - \frac{1}{2^k}.$$

2.2 VSS

Verifiable secret sharing (cf. [CGMA85]) is a fundamental tool for our protocols. In *secret sharing,* one player holds a secret value, which he distributes among the other players, giving each player a quantum of information called a "piece." Two properties must be satisfied:

1. (Privacy) The pieces held by any coalition of fewer than $t+1$ players are independent of the secret.

2. (Reconstructability) Any coalition containing $n-t$ honest players can reconstruct the secret in full from their information.

In this paper we shall explicitly require the following property as well:

3. (Verifiability)

   - If the secret is not shared correctly and uniquely reconstructible, all honest players output CHEATING, with probability at least $1-\frac{1}{2^t}$.
   - If the secret is uniquely reconstructible, all honest players agree on its validity.

We shall utilize the solution to VSS for $2t < n$ given by [Rab88]. Briefly, it uses Shamir's method for sharing a secret, and requires that each piece be reshared using a weak form of sharing. The weaker form of sharing includes information called "check vectors," which allow verification of the pieces. We refer the reader to [Rab88, BR89] for details.

2.3 Disqualification

We consider a processor *disqualified* if $t+1$ or more players have broadcast impeachments of it. When player i is disqualified, the remaining players run a recovery protocol to reshare all the secrets using polynomials of smaller degree (since there are fewer faulty players left to participate). This protocol involves synthesizing and broadcasting one new piece of each secret, in order to reduce the degree of the secret polynomial by one; we omit it from this abstract. (During the Input stage, however, a faulty player's information is replaced by a default value.) Thereafter the good players ignore player i and continue to evaluate the circuit C_f, using a new $t\prime = t-1$ as the resiliency parameter.

3 Compiling Multiparty Protocols

Given a function $f(x_1, \ldots, x_n)$ and a polynomial size circuit for it, C_f, we wish to construct a protocol to compute f.

The three-stage paradigm [GMW87,BGW88,CCD88] will serve as our model for protocols. In the Input stage, the inputs x_i are verifiably shared. In the Evaluation

stage, the circuit C_f is evaluated, producing a new secret $w = f(x_1, \ldots, x_n)$. In the Output stage, the secret w is reconstructed and revealed.

The Input and Output stages are easy to describe. During the Input stage, each player i uses VSS to share his input x_i. Any honest player who detects misbehavior impeaches the misbehaving player by broadcasting a vote against him. If a player is disqualified, substitute a default value. (As an aside, we may require that the default value be secret and selected according to some samplable distribution. In that case, use a circuit C'_f which has additional, random inputs supplied by each player, which are used to compute the desired default value.)

The Output stage operates exactly like the reconstruction stage of a VSS protocol; following the Evaluation stage, the players hold a set of information about w exactly as though it were shared using VSS. The reconstruction of w involves a broadcast of pieces and "check vectors" to verify the pieces before interpolating.

The Evaluation stage forms the crux of our protocols, and contains our new methods for multiparty computations. We follow the outline of [BGW88, CCD88], in which we simulate the computation of a circuit gate by gate using algebraic operations over the field used for secret sharing. Our protocols work for larger numbers of faults, however.

4 Addition of Secrets

It is possible though not trivial to extend [Rab88] to a method for linearly combining secrets.

Lemma 5 *Let u and v be secretly shared values, and let a, b and c be publicly known field elements. Then for $2t < n$, there exists a t-resilient protocol $w \leftarrow au + bv + c$ to provide each player with a piece of a new secret w whose value is $au + bv + c$, without revealing any information about u, v, or w.*

Each player i uses $af(i) + bg(i) + c$ as his piece of $h(x) = af(x) + bg(x) + c$, the desired new polynomial which represents the secret w.

In order that w be a verifiable secret according to the VSS protocol, each $h(i)$ must be reshared in a weak fashion. This involves the use of check vectors for the pieces of $h(i)$. In the full paper we describe in detail how to generate new check vectors for $h(i)$; because the method involves examining in detail the methods used by [Rab88], in this abstract we only sketch the idea. If $f(x)$ and $g(x)$ were generated by different sources, the check vectors associated with them will not be compatible for checking that $h(i) = af(i) + bg(i) + c$. On the other hand, player i can create new check vectors for $f(i)$ and $g(i)$ that will be compatible for checking linear combinations; and there is a protocol revealing nothing about $f(i)$ and $g(i)$ that ensures that player i creates new check vectors which are consistent with the previous check vectors.

5 Multiplication of Secrets

We come now to one of two important results.

Theorem 6 *Let u and v be secretly shared values. Then for $2t < n$, there exists a t-resilient protocol to provide each player with a piece of a new secret w whose value is uv, without revealing any information about u, v, or uv.*

By Lemma 5 it suffices to show Theorem 1. We prove Theorem 1 by describing the protocol.

Our solution follows a few brief steps (cf. [BGW88]). Let u be shared using $f(x)$ and v be shared using $g(x)$.

Step 1.
Each player i secretly shares the value $f(i)g(i)$ and "proves" that he has in fact shared this value (see section 5.1). If his proof fails, he is disqualified (see section 2.3).

Step 2.
From the collection of secret products, the polynomial $f(x)g(x)$, of degree $2t$ and free term uv, is determined. Using a protocol to truncate the polynomial to degree t (see section 5.2), and then to add a random polynomial of degree t and free term 0, each player i is supplied with the value $h(i)$ for the resulting polynomial $h(x)$ of degree t and free term uv.
□.

In section 5.1 we describe the method by which player i can share $f(i)g(i)$ and prove with respect to the secrets $f(i)$ and $g(i)$ that he has in fact shared the correct secret product. The protocol for reducing the degree of the polynomial is based on linear combinations of secrets, and is presented in section 5.2.

5.1 Verifiable Multiplication

In order to accomplish Step 1, the verifiable sharing of $f(i)g(i)$ and zero-knowledge proof of its correctness, let us first define and solve a more general problem, which forms the basis for the results of this paper.

The ABC problem. Let Alice know the values of secrets a and b. Alice must share a new secret c and prove to the other players that the secret value of c is indeed ab. No other information about a, b, or c must be revealed (unless Alice is faulty).

Given a protocol for the ABC problem, Alice will be able to prove to the network that a new secret which she shares is indeed $f(i)g(i)$.

Lemma 7 *(ABC Lemma.) If there exists a t-resilient protocol for linear combinations of secrets, then there exists a t-resilient protocol to solve the ABC problem.*

We prove the ABC lemma by exhibiting the protocol. First, an overview: In the first phase, Alice shares several triples of secrets $(\mathcal{R}, \mathcal{S}, \mathcal{D})$ satisfying a simple equation (of the form $\mathcal{D} = (a + \mathcal{R})(b + \mathcal{S})$), which will be used to ensure that Alice does not misbehave. In the second phase, the players select and reveal combinations of some of these triples in order to confirm that every triple satisfies the simple equation. Finally, each unrevealed triple of secrets gives rise to a simple linear combination of secrets that should equal the desired product ab. The third phase checks that the linear combinations are consistent.

Protocol ABC.

Phase 1.

Let a and b be verifiably secret shared. Alice verifiably shares a third value $c = ab$. Then Alice chooses and shares several random secrets $r_1, \ldots, r_{2k}$ and $s_1, \ldots, s_{2k}$ chosen uniformly from the field used for sharing. (The chance of incorrectness will be bounded by $\frac{1}{2^k}$.) For simplicity we take $k > n$ and k a power of two. Alice also shares secrets $d_1, \ldots, d_{2k}$ having (secret) values $d_j = (a + r_j)(b + s_j)$.

Phase 2.

Next, the system confirms that in fact each $d_j = (a + r_j)(b + s_j)$. The system selects and announces a set $Y = \{j_1, \ldots, j_k\}$ of k random indices. (Selecting indices at random can be achieved by the following method: each player selects a random secret in the field $GF(k)$ and shares it over $GF(k)$; together, they secretly compute the sum; and finally, the sum is revealed.) For each $j \in Y$, the system secretly computes the sums $(a + r_j)$ and $(b + s_j)$, then reconstructs each of them, along with the value of d_j. Every processor checks that the product of the sums matches the value of d_j; if not, Alice is disqualified.

Phase 3.

If Alice passes the test in Phase 2, then the system reconstructs and reveals the values of r_j and s_j for every index $j \notin Y$. For each index $j \notin Y$, the system then secretly computes (but does not reveal) the linear combination $c_j = d_j - r_j b - s_j a - r_j s_j$. (This is a linear combination since r_j and s_j are now public.) If Alice has behaved properly, then in fact each secret c_j will contain the value ab.

The system now computes and reveals the differences $(c - c_j)$ for every $j \notin Y$. If any difference is nonzero, Alice is disqualified. Otherwise, if Alice has passed all the tests, then the system accepts Alice's sharing. □.

In Phase 2, the sums $(a + r_j)$ and $(b + s_j)$ are independent of a and b, respectively, since r_j and s_j are uniformly random field elements. Their product d_j is also independent of a and b. Therefore, revealing the sums and d_j values for $j \in Y$ reveals nothing about a and b. If Alice has behaved, then regardless of the values of a and b, the secrets $(c - c_j)$ computed in Phase 3 will all be zero, so that revealing them will give no information about a or b.

How may Alice cheat without being detected? We shall show that she must behave properly on exactly those indices chosen in Phase 2 and she must misbehave

on all the others. Let X be the set of indices j for which Alice shares d_j correctly, that is, for which Alice shares d_j having the value $(a + r_j)(b + s_j)$. In Phase 2, the set Y of indices chosen by the system must be a subset of X, or else Alice's misbehavior is detected. In Phase 3, the remaining indices $j \notin Y$ must all satisfy $c = d_j - r_j b - s_j a - r_j s_j$ or else Alice is caught. If $c \neq ab$ (Alice is cheating), no index $j \notin Y$ is in X. Hence $X = Y$, and since Y is chosen randomly after Alice has shared all her secrets, the probability that Alice can cheat without being detected is no more than $\frac{1}{2^t}$.

5.2 Degree Reduction

Lemma 8 *Let $p(x)$ be a polynomial of degree $2t$ with hidden coefficients, having $c = p(0)$ as its free term, and suppose each player i knows the secret value $p(i)$. Let each value $p(i)$ itself be verifiably shared among the players. Assuming a protocol for linear combinations of secrets, there is a protocol to verifiably share $c = p(0)$ using a random polynomial $q(x)$ of degree t.*

One method for interpolating polynomials, given a list of n values $p(1), \ldots, p(n)$, is to use LaGrange polynomials:

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

$$p(x) = \sum_{i=1}^{n} L_i(x) p(i).$$

If we denote by $\overline{f}(x)$ the truncated polynomial $f(x)$ mod x^{t+1}, then

$$\overline{p}(x) = \sum_{i=1}^{n} \overline{L_i}(x) p(i).$$

The polynomials $L_i(x)$, and hence $\overline{L_i}(x)$, are publicly known; they depend only on the fixed, known interpolation points (which we have taken to be $1, \ldots, n$ for simplicity.)

We shall provide each player m with the value $\overline{p}(m) + r(m)$, where $r(x)$ is a random polynomial of degree t and free term 0. The reason for including $r(x)$ is to ensure that coefficients of the new polynomial $q(x)$ are indeed random. (The generation of $r(x)$, along with the providing of $r(m)$ to player m, is left as an exercise to the reader.)

Define $q(x) = \overline{p}(x) + r(x)$. Recall that each value $p(i)$ is a secret known only to player i, and can itself be shared. For each m, the value $q(m)$ can be written as a linear combination of *secrets* $p(1), \ldots, p(n)$:

$$q(m) = (\overline{p} + r)(m) = r(m) + \sum_i \overline{L_i}(m) p(i).$$

(Recall that each $\overline{L_i}(m)$ is publicly known, and serves as a weight for the secrets.) By performing this linear combination of secrets and revealing the result to player m, it is possible to provide each player m with the value $q(m)$, solving the problem at hand.

5.3 Proving the Multiplication Theorem

As in [BGW88,CCD88], given the power to reduce the degree of the product polynomial $f(x)g(x)$ and to prove (in zero-knowledge) that the new pieces are correct, we have completed the proof of Theorem 6, that there exists a t-resilient multiplication protocol.

6 Multiparty Protocols

Addition and multiplication protocols (Lemmas 5, 7, and 8) make the Evaluation stage possible. In order to evaluate C_f, the protocol simulates the circuit layer by layer, producing a new set of secrets each time.

The overall protocol consists of verifiably secret-sharing the input values $x_1, \ldots, x_n$, evaluating a circuit C_f on those values to produce a new secret w, and finally reconstructing w. This completes the proof of our main result, Theorem 2.

7 Zero-Knowledge Proofs

For definitions and background concerning interactive proof systems and zero-knowledge proofs, we refer the reader to [GMR89]. In the full paper we formally define and extend these concepts to the network setting.

The method of section 5.1 for proving that a secret contains the product of two others extends immediately to proving that a secret w contains the value of some polynomial size arithmetic circuit C_P applied to secrets $v_1, \ldots, v_m$. Alice secretly shares the output of each gate g_l when C_P is applied to $v_1, \ldots, v_m$. The network must verify that the output of each g_l is correct with respect to its two inputs. Let w_l be the output of gate g_l, and u_l and v_l represent its inputs. If g_l is a linear combination gate, the network secretly computes the linear combination $w_l - (au_l + bv_l + c)$ and reveals it to verify that it is zero. If g_l is a multiplication gate, Alice uses the ABC protocol to prove that $w_l = u_l v_l$.

Theorem 3, which states that zero-knowledge proofs are possible in the presence of a partly-corrupt network, follows from these observations. Notice that this technique for zero-knowledge proofs of predicates on secretly shared values uses only a *constant* number of rounds and a message complexity proportional to the size of the circuit C_P.

In order to show Lemma 4, stating that any language which has an interactive proof system is also provable in zero-knowledge in the presence of a network, we must demonstrate how such a zero-knowledge proof runs. Let $L \in$ IP and let $<P, V>$ be a proof system for L. Without loss of generality assume V runs for exactly $p(n)$ steps on all inputs of length n, where $p()$ is a polynomial. The verifier is a deterministic machine, given its random tape. The trick to showing the Lemma is to note that the network can generate secret random bits for V, and then simulate the operation of V, ensuring not only that the state of V and its tapes is never revealed, but also that the final output of the simulation is a correct assessment of what a true

verifer would answer. Alice takes the position of the prover, sharing her messages to V among the network, while the network simulates a message from V to P by reconstructing it only for Alice.

8 Conclusion

We have shown the new and powerful result that multiparty computations can be performed securely and secretly despite Byzantine faults by up to half the parties. No unproven assumptions are needed. Furthermore, our techniques require only a small polynomial number of message bits and a small constant number of rounds of interaction per basic operation (multiplication, addition, logical "and," logical "or").

We have also shown how a member of a network can share two secrets along with their product, proving to the network that the secret product is correct. More generally, we have demonstrated that a player can prove anything "provable" (in the sense of interactive proofs) about a set of *secrets* which she knows, without revealing anything else about the values of the secrets. She can also prove any IP-statement about *known* values without revealing anything other than that the statement holds. This new method for zero-knowledge proofs in the presence of a network of processors requires no unproven assumptions and is correct with exponentially small chance of error.

In comparison to other solutions, the methods in this paper are more practical and efficient. They allow field operations over an exponentially large field, which permits direct secret computations of many naturally occurring operations, such as arithmetic. This is a great advantage over the requirements and inefficiencies of bit-simulations, on which all other methods rely.

Each of our results allows an exponentially small chance of error, but it is easily proven that for $3t \geq n$ no protocol can tolerate t Byzantine faults without error. Since it is impossible even to *share* a secret when more than half the parties are faulty, our results are optimal.

References

[BB88] D. Beaver, J. Bar-Ilan. "Non-Cryptographic Fault-Tolerant Computing in a Constant Expected Number of Rounds of Interaction." Proc. of 21st STOC (1989), 201-209.

[Bea88b] D. Beaver. "Secure Multiparty Protocols Tolerating Half Faulty Processors." Technical Report TR-19-88 (September, 1988), Harvard University.

[BGW88] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." Proc. of 20th STOC (1988), 1-10.

[Bla81] G. R. Blakley, "Security Proofs for Information Protection Systems." Proceedings of the 1980 Symposium on Security and Privacy, IEEE Computer Society Press, NY (1981), 79-88.

[BR89] M. Ben-Or, T. Rabin. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority ." 21^{st} STOC (1989), 73-85.

[CCD88] D. Chaum, C. Crépeau, I. Damgård. "Multiparty Unconditionally Secure Protocols." Proc. of 20^{th} STOC (1988), 11-19.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, B. Awerbuch. "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults." Proc. of 17^{th} STOC (1985), 383-395.

[GMR89] S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." SIAM Journal on Computing 18, no. 1 (1989), 186-208.

[GMW87] 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.

[Kil] J. Kilian, personal communication.

[Kil88] J. Kilian. "Founding Cryptography on Oblivious Transfer." Proc. of 20^{th} STOC (1988), 20-29.

[Rab88] T. Rabin. "Robust Sharing of Secrets When the Dealer is Honest or Cheating." Masters Thesis, Hebrew University, 1988.

[Sha79] A. Shamir. "How to Share a Secret." CACM 22 (1979), 612-613.

[Yao86] A. Yao. "How to Generate and Exchange Secrets." Proc. of 27^{th} FOCS (1986), 162-167.