Asynchronous Secure Computations with Optimal Resilience

(Extended Abstract)

Michael Ben-Or
Hebrew University

Boaz Kelmer
IBM Israel Science & Technology

Tal Rabin
Hebrew University *†

Abstract

We investigate the problem of multiparty computations in a fully connected, asynchronous network of n players, in which up to t Byzantine faults may occur. It was shown in [BCG93] that secure error-less multiparty computation is possible in this setting if and only if $t < n/4$. We show that when exponentially small probability of error is allowed, this task can be achieved even when the number of faults is in the range $n/4 \leq t < n/3$. From the lower bounds of [BCG93] for the asynchronous fail-stop model it follows that the resilience, $t < n/3$, of our protocol is optimal.

We describe an $(\lceil \frac{n}{3} \rceil - 1)$-resilient protocol that securely computes any function $\mathcal{F}$. With overwhelming probability all the non-faulty players complete the execution of the protocol. Given that all the honest players terminate the protocol, they do so in time polynomial in n, in the boolean complexity of $\mathcal{F}$, and in $\lceil \log \frac{1}{\epsilon} \rceil$, where ϵ is the error probability.

Our protocol follows the scheme of [BGW88, RB89] for multiparty computations in synchronous networks, in which the intermediary results of a circuit for $\mathcal{F}$ are always kept shared among the players as a verifiable secret. As the asynchronous network makes it impossible to use a regular Verifiable Secret Sharing scheme for computations, we introduce a new secret sharing scheme called *Ultimate Secret Sharing*. This scheme guarantees that all the honest players will obtain their share of the secret, and it enables the players to verify that the shares are genuine.

*Supported by the Eshkol Fellowship of the Israel Ministry of Education.

†e-mail: talr@cs.huji.ac.il

PODC 94 - 8/94 Los Angeles CA USA

1 Introduction

Secure Multiparty Computation is a versatile and very powerful tool in the design of cryptographic protocols. It allows any computational task, that can be computed securely by the system with the help of a trusted third party, to be securely computed within the system itself when no such trusted third party exists.

The Problem: Consider a network of n players (processors), where every two players are connected via a secure and reliable communication channel. The network is *asynchronous*, meaning that any message sent on these channels can have arbitrary (finite) delay. Each player has a private input value x_i, and the goal is to collectively compute some function $\mathcal{F}(x_1, \ldots, x_n)$.

Players who conform with their protocols are called *honest*. Some of the players may deviate from their protocols, and even collaborate in order to disrupt the computations. These are called *faulty*. There is an upper-bound t on the number of faulty players.

The meaning of multiparty computations in this model is delicate. We follow the definitions given by [BCG93]. Roughly speaking, an asynchronous multiparty computation consists of three stages. In the first stage, each player P_i *commits* himself (in a sense to be made precise later) to his input. Even if P_i is faulty, if he completed this step, then he is committed to some value (not necessarily x_i). Denote by x'_i the value committed by player P_i. For each honest player P_i, $x'_i = x_i$. Then the players agree on a subset *CompSet* of size at least $n - t$ of committed inputs. The subset *CompSet* will be the same for all honest players. In the last stage the players will compute the value $\mathcal{F}(y_1, \ldots, y_n)$, where $y_i = x'_i$ for $P_i \in$ *CompSet*, and $y_i = 0$ otherwise.

A protocol is *t-resilient* if *every* honest player will complete the computation and output the value $\mathcal{F}(y_1, \ldots, y_n)$. Furthermore, the faulty players have no additional information beside what they can infer from their own inputs and the final output.

It is clear that the above definition captures the maximal information about the value $\mathcal{F}(x_1, \ldots, x_n)$ which can be obtained in our setting, where as many as t of

the players may be faulty, and where messages may be delayed for an arbitrary long time.

Related Work and Our Results: The exact conditions under which secure multiparty computation is possible for *synchronous* distributed systems have been studied quite extensively. Given a synchronous distributed system of n players, where any two players are connected via a private channel, and where at most t of the players may become faulty, we know that: Secure error-free computation is possible exactly under the same conditions needed for the Byzantine Agreement Problem, namely, $t < n/3$ [PSL80, BGW88]. Furthermore, secure computation is possible, with an exponentially small probability of error, even for $t < n/2$ if we add a broadcast channel to the system [RB89].

Recently, Ben-Or, Canetti and Goldriech [BCG93] initiated the study of secure computation for *asynchronous* systems. Generalizing the "trusted party" model of Micali and Rogoway [MR], they provide exact definitions of secure computation for asynchronous systems, and prove that asynchronous secure error-free computation is possible if and only if the number of faulty players t is less than $n/4$. The asynchronous Byzantine Agreement problem can be solved by a randomized error-free protocols for $t < n/3$ [Bra84]. Thus the conditions needed for asynchronous error-free secure multiparty computation are more stringent than those required for the error-free asynchronous Byzantine Agreement problem.

Inspecting the conditions for secure multiparty computation in the synchronous case, it is natural to ask whether by allowing an exponentially small probability of error we can obtain asynchronous secure computation with optimal resilience. That is, when the number of faulty players t is in the range $n/4 \leq t < n/3$. In this paper we answer this question affirmatively, proving:

Theorem 1 *In a completely asynchronous network of n players, where any two players have a private communication channel, and where at most $t < n/3$ players are faulty, secure multiparty computation with an exponentially small probability of error is possible.*

Furthermore, The complexity of the protocol achieving this, given that the players complete the protocol, is polynomial in the number of players n, the boolean complexity of the computational task, and in $\lceil \log \frac{1}{\epsilon} \rceil$, where $\epsilon > 0$ is the error probability.

Obstacles to over come: The first step in the general direction of Theorem 1 was accomplished by Canetti and Rabin [CR93]. They gave an asynchronous Verifiable Secret Sharing Scheme (AVSS). Informally, AVSS allows a dealer to share a secret that can be reconstructed by the players at a later stage. AVSS protects the honest players from a faulty dealer, by forcing him to commit to a specific value that is guaranteed to be the reconstructed value. The protocol of [CR93] tolerates up to $t < n/3$ faults, and has an exponentially small probability of error. With this AVSS, following the lines of Feldman and Micali [FM88] and Feldman [Fel89], they obtained a fast t-resilient asynchronous Byzantine Agreement protocol for $t < n/3$. Our secure protocol builds on this AVSS protocol. However, the AVSS of [CR93] by itself is not sufficient for implementing secure multiparty computations.

The main difficulty is the following: If $n = 3t+1$, the scheme in [CR93] assures that at least $n-t$ players will obtain a share of the secret. Of these, only $n-2t = t+1$ are guaranteed to be honest, and that is exactly enough to enable the system to reconstruct the secret at a later stage. When we try to add (or multiply) two secrets that have been shared using this AVSS scheme, each secret has its own set of $n-t$ players with valid shares. Therefore, we can only guarantee that $n-2t$ players will hold a share from both secrets. Of these, t may be faulty, leaving us with only a single honest player that holds a correct share of both secrets.

Another difficulty is the problem of erroneous shares. As the models of both [BGW88, BCG93] ensure the participation of at least $3t+1$ players, they are able to use methods of error correcting codes to correct faulty shares. In our model, due to the asynchronicty of the network and the number t of faulty players, we can only count on the participation of $2t+1$ players. Thus, we can not employ the methods of error correcting codes.

Overview: Our protocol for computing a function $\mathcal{F}$ follows the general method of Ben-Or, Goldwasser and Wigderson [BGW88]. The players simulate an arithmetic circuit for $\mathcal{F}$, gate after gate, in such a way that the intermediate results are always kept as verifiable secrets distributed among the players.

To handle secure computation for $t < n/3$, we must make sure that all the honest players will eventually obtain their share of each secret. We achieve this using a new secret sharing scheme which we call *Ultimate Secret Sharing* (USS). At first glance this task appears to be impossible, since in an asynchronous environment there is nothing to prevent the adversary from delaying t honest players, and forcing the system to end the secret sharing protocol without their participation! We circumvent this by requiring that each share be distributed among all players. Thus, a late player can obtain his share by asking the system to reconstruct it. In addition, our secret sharing scheme encompasses a built-in self correcting property. A definition and an implementation of a USS scheme are presented in Section 3.

Having a stronger verifiable secret sharing scheme is not enough. In order to carry out the computation, we must provide protocols to add and multiply such shared secrets. Here, following the lines of Rabin and

Ben-Or [RB89], we introduce asynchronous protocols for the addition and multiplication of ultimately shared secrets [1]. These protocols are presented in Section 5.

Following [BCG93], in the case of asynchronous secure function evaluation, the players need to agree on a subset of inputs on which the function will be evaluated. In Section 4 we provide a protocol to agree on a common subset of at least $n-t$ players. Our protocol is a simple constant expected round protocol. This should be compared with the previous $O(\log n)$ round protocol of [BCG93] to solve this same problem.

Finally, our protocol, as well as the verifiable secret sharing protocol of [CR93], have the following annoying property: The exponentially small error probability of the protocol includes an exponentially small but non zero probability of not terminating. This should be contrasted with the asynchronous Byzantine Agreement problem where the randomized protocol terminates with probability 1. We prove that this is unavoidable. That is, if $n \leq 4t$ then any t-resilient asynchronous verifiable secret sharing protocol A must have some probability $q_A > 0$ of not terminating. A sketch of the proof appears in Section 6.

2 Definitions

The Model

Our model consists of a fully connected, asynchronous network of n players. Every two players are connected via a secure and reliable channel of communication. That is, each message sent on such a channel cannot be heard by other players, and is guaranteed to arrive at its destination. However, each message may have an arbitrary (finite) delay.

Up to t of the players may become faulty at any time during the run of the protocol. The faulty players may deviate from their protocols, and collaborate in order to reveal private data of honest players, and even to disrupt the computation.

To model the faulty players' behavior, we postulate a computationally unbounded, dynamic adversary. This adversary may choose, at any stage of the protocol, additional players to corrupt, as long as he does not exceed the limit of t faulty players. Once a player is picked by the adversary, he hands over to him all his private data, and gives the adversary the control on his next moves. To model the network's asynchrony, we further assume that the adversary can schedule the communication channels, i.e. he can determine the time delays of all the messages (however, he cannot read nor change those messages). We call such an adversary a *t-adversary.*

We assume that a finite field E, with $|E| > n$, is used by all players. With out loss of generality, the function to be computed is an n-variable polynomial $\mathcal{F}$ over E. An arithmetic circuit computing $\mathcal{F}$ is known to all the players [2]. The circuit consists of binary addition and multiplication gates, and unary gates for multiplication by non-zero constants from E.

Asynchronous Computations

Each player P_i holds a private input x_i, and the players wish to securely compute the exact value of a function $\mathcal{F}(x_1, \ldots, x_n)$. However, since the network is asynchronous, and t players may be faulty, the players can never even wait for more than $n-t$ of the inputs to be entered to the computation. Furthermore, the missing inputs are not necessarily of the faulty players.

To define secure computations in our model, we use the notion of a *trusted party.* In an ideal trusted party scenario, the players send their inputs to a trusted party (of course the faulty players may substitute their inputs arbitrarily). An adversary chooses a subset *CompSet* of the players of size at least $n-t$, and delivers to the trusted party only the messages sent by players in this set. Then the trusted party computes an approximation of $\mathcal{F}$, by computing $\mathcal{F}$ after substituting all the missing inputs with 0. Finally the trusted party sends the value he computed back to the players. The good players output this value, and the faulty players output an arbitrary function of the information they gathered during the execution of the protocol. We say that an asynchronous protocol *computes* a function $\mathcal{F}$ in our model if the output of the players is equivalent to their output in the ideal trusted party scenario. We refer the reader to [BCG93, Can93] for complete formal definitions.

Tools

In our protocols we shall use some tools devised for asynchronous networks. Following are references to the origin of these tools, with a few words of explanation (a full definition of each one of these tools appears in [Rab94]).

A simulation of an *asynchronous broadcast channel* (A-Cast) was devised by [Bra84]. *Fast Asynchronous Byzantine Agreement* (BA) was devised by [CR93]. *Asynchronous Verifiable Secret Sharing* (AVSS), as well as a weaker scheme of secret sharing (AWSS) are also from [CR93]. *Two&Sum Secret Sharing* [Rab89, CR93] is a variation on any secret sharing scheme that enables the sharing of three secrets, in a manner that the play-

[1] We note that a simple adaptation of the error-free protocol of [BGW88] using our new ultimate secret sharing scheme does not work, since the asynchronous system must be able to compute when only $n-t = 2t+1$ players are participating. The correctness of the multiplication protocol of [BGW88] uses the fact that $2t+1$ *honest* players are participating, while here we must proceed with only $t+1$ honest players.

[2] The players need not know this circuit, nor $\mathcal{F}$, until the input phase has ended.

ers are guaranteed that the sum of two of them is equal to the third. *Multiple Reconstruction of Secrets* [Rab89] makes it possible for each secret to be reconstructed twice - once privately to some player, and then publicly. A *ZK technique for showing that a set of shares define a polynomial of degree at most t* was devised by [CCD88]. A *Product ZKP* protocol [RB89] enables a dealer to prove that three secrets which he has shared satisfy that one of them is the multiplication of the other two, without revealing any additional information. We note that all these protocols are $(\lceil \frac{n}{3} \rceil - 1)$-resilient, and of polynomial complexity.

3 Ultimate Secret Sharing

As stated in the introduction, the AVSS scheme of [CR93] falls short of our needs in order to be able to compute. We need to enhance the secret sharing scheme so that *all* the honest players will have proper shares of each secret, and to provide an error detecting method to sieve out faulty shares.

We make our requirements formal in the following definition:

Definition 1 *For a dealer D sharing a secret s, let* (Sh, Rec) *be a pair of protocols. We say that* (Sh, Rec) *is a* $(1-\epsilon)$-correct, t-resilient *USS scheme for n players if the following hold, for every t-adversary.*

- **Termination.** *With probability $1-\epsilon$ the following requirements hold.*
  1. *If the dealer is honest, then each honest player will eventually complete protocol* Sh.
  2. *If some honest player has completed protocol* Sh, *then each honest player will eventually complete protocol* Sh.
  3. *If all the honest players have completed protocol* Sh, *and invoke protocol* Rec, *then each honest player will complete protocol* Rec.
- **Secrecy.** *If the dealer is honest and no honest player has begun executing protocol* Rec, *then the faulty players have no information about the shared secret.*
- **Correctness.** *Once an honest player has completed protocol* Sh, *then there exist a unique value r, shares $\beta_1, ..., \beta_n$, and protocols $L(\cdot)$ and $G(\cdot)$, such that with probability $1-\epsilon$ the following hold.*
  1. *If the dealer is honest, then $r = s$.*
  2. *If player P_i invokes protocol $L(i)$ then at the end of the execution he locally outputs the value β_i.*
  3. *If $G(i) \neq null$ then $G(i) = L(i) = \beta_i$.*
  4. *If all the honest players invoke protocol* Rec *with inputs $G(1), ..., G(n)$, then each honest player outputs r.*

This definition complies with our requirements in the following way: The protocol $L(\cdot)$ ensures that each player can access his share of the secret. The self correcting property is embedded in the protocol $G(\cdot)$. The output of this function may be *null* (for faulty players), but if it is not *null* then it is guaranteed to be a proper share.

Our implementation of USS-Share and USS-Rec is described in Figures 1 and 2, respectively. In our scheme, a dealer shares a secret s using a polynomial $f(x)$ (i.e. $f(x)$ is a random polynomial over E except that $f(0) = s$), and each player P_i will have at the end of the sharing phase the share $f(i)$. The idea is that the dealer who shares the secret, doesn't hand each player his share, but rather shares each share using AVSS-Share. Due to the properties of AVSS-Share, he is now committed to n values. He proceeds to prove that these values are proper shares. He does this using the cut and choose zero knowledge technique [CCD88] for proving that n values define a polynomial of degree $\leq t$. Thus he is committed to n *valid* shares. Once these two steps are carried out successfully, we are guaranteed that each player has access to his share of the secret, by means of private AVSS-Rec (the protocol $L(\cdot)$). And the shares which have been shared using AVSS-Share are guaranteed to be proper shares due to the zero knowledge proof which was carried out. The players can authenticate each share by means of public AVSS-Rec (the protocol $G(\cdot)$).

Lemma 1 *Let $n \geq 3t+1$. Then for every $\epsilon > 0$, the pair* (USS-Share[ϵ],USS-Rec[ϵ]) *is a $(1-\epsilon)$-correct, t-resilient USS scheme for n players.*

Proof: See [Rab94] □

As our final goal is multiparty computations, we have to formulate the notion of secrets that were not shared by a specific dealer, but rather computed by the network. Following is a definition of an *ultimately shared secret.*

Definition 2 *We shall say that a secret s, is* $(1-\epsilon)$-ultimately shared *if there exist shares $\beta_1, ..., \beta_n$, and protocols $L(\cdot), G(\cdot)$ and* Rec, *such that with probability $1-\epsilon$ the following hold.*

- **Secrecy.** *If no honest player has begun executing protocol* Rec, *then the faulty players have no information about the shared secret.*
- **Correctness.**
  1. *If player P_i invokes protocol $L(i)$ then at the end of the execution he locally outputs the value β_i.*
  2. *If $G(i) \neq null$ then $G(i) = L(i) = \beta_i$.*
  3. *If all the honest players invoke protocol* Rec *with inputs $G(1), ..., G(n)$, then each honest player outputs s.*

Note 1: If s is the value set by a USS-Share protocol, then s is an ultimately shared secret.

Protocol USS - Share[ϵ] (Sketch)

Code for the Dealer, sharing a secret s:

1. Choose a random polynomial $f(x)$ for s.
2. For all $1 \leq i \leq n$, compute $\beta_i \stackrel{\Delta}{=} f(i)$.
3. For all $1 \leq i \leq n$, share β_i using AVSS-Share[ϵ'].
   Comment: Note that β_i should be shared using a multi-rec protocol.
4. Prove that the values which were set by the AVSS-Share - $\beta'_1, ..., \beta'_n$ - define a polynomial of degree $\leq t$.

Code for player P_i:

5. Participate in the proof that $\beta'_1, ..., \beta'_n$ define a polynomial of degree $\leq t$.
6. If the shares define a polynomial, then A-Cast("Proper sharing").
7. Upon completing $t+1$ A-Casts of the form "Proper Sharing" terminate.

Figure 1: USS - Sharing Phase

Protocol USS - Rec[ϵ]

Code for player P_i:

1. Execute the AVSS-Rec Protocol for all shares.
2. Take any $t+1$ values and interpolate the polynomial $f(x)$.
   Output s, the constant term of f.

Figure 2: USS - Reconstruction Phase

Note 2: If a secret s is ultimately shared by the implementation described above (i.e. $s = f(0)$ and $\beta_i = f(i)$ for $1 \leq i \leq n$), we shall say that s is *ultimately shared by a polynomial f.*

4 Agreement on a Common Subset

During the computation stage, there are times that the players need to decide on a subset of players, of size at least $n - t \geq 2t + 1$, that satisfy some property. It is known that all the honest players will eventually satisfy this property, but some faulty players may satisfy it as well. For example, the property may be that a player has properly shared his input. The problem is that all the (honest) players need to agree on the same subset.

Suppose there is a dynamic predicate Q, that assigns a binary value to each player. By dynamic we mean that not all the players need to be assigned at the same time. When a player P_j is assigned his value, denoted as $Q(j)$, it is guaranteed that all the players will (eventually) learn to know this value. Furthermore, it is known that all the *honest* players will (eventually) be assigned the value 1. In this section we present a protocol that enables the players to agree on a subset of size at least $n - t$ of players for whom the predicate is 1.

The idea is to execute a BA protocol for each player, to determine whether it will be in the agreed set. That is, n BA protocols are executed simultaneously, and each player enters his inputs to the BAs asynchronously. (To save a $\log n$ factor of running the n BAs, the techniques of [BOEY91] are employed).

Protocol Agreement[Q]

Code for player P_i:

1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.
2. Upon completing $2t+1$ BA protocols with output 1, enter input 0 to all BA protocols for which you haven't entered a value yet.
3. Upon completing all n BA protocols, let your $SubSet_i$ be the set of all indices j for which BA_j had output 1. Output $SubSet_i$.

Figure 3: Agreement on a Common Subset

Lemma 2 *Using the protocol* Agreement *(Figure 3), the players agree on a common subset of size at least* $2t+1$ *of players* P_j *for whom* $Q(j)=1$.

Proof: First we shall prove that at least $2t+1$ BAs terminate with output 1: Suppose that some honest player P_i has entered a 0 value into some BA_j in Step 2. This means that at least $2t+1$ BAs terminated with output 1, which is what we want to prove. So assume that no honest player has entered a 0 value to any BA protocol. As for each honest player P_h we have $Q(h)=1$, the inputs of all the honest players to BA_h will be 1. By the correctness property of the BA protocol, the output of BA_h will be 1. As this is true for every honest player, at least $2t+1$ BAs will terminate with output 1.

Next we will show that all n BAs will terminate: As $2t+1$ BAs will terminate with output 1, all the honest players will enter an input to the remaining BAs, and thus they will terminate.

It is clear from the protocol that once an honest player terminates, he has a *SubSet* of size at least $2t+1$, and from the properties of the Byzantine Agreement protocol it follows that all the honest players agree on the same set. It remains to be shown that for each j in *SubSet*, $Q(j)=1$. This is true because if BA_j terminated with output 1, it must be that at least one honest player entered 1 as his input to BA_j. □

5 The Computation Protocols

Our protocol for computing a function $\mathcal{F}(x_1,\ldots,x_n)$ follows the general method of [BGW88]. The players simulate an arithmetic circuit for $\mathcal{F}$, gate after gate, in such a way that the intermediate results are always kept as secrets distributed among the players. To overcome the obstacles raised by the asynchronous setting, each intermediate result will be *ultimately shared.*

In the input phase each player P_i USS-Shares his input x_i. By the definitions of USS (Section 3) it follows that once this step is completed, each player P_i is committed to a value x_i', which is ultimately shared among the players. The players then use protocol Agreement (Figure 3) to agree on a common subset of inputs that were properly shared, and substitute each input not in this set with 0. They proceed to compute $\mathcal{F}$ on this set of values. In the reconstruction phase the players simply USS-Rec the final result.

It remains to be shown how the players can simulate the arithmetic gates. In the course of our protocols, there are many times where a player P will need to prove that some secret s that he shared is equal to a secret s' that was previously shared among the players, with out revealing any additional information on either one of the secrets. For clarity and simplicity's sake, we have omitted the intricate details of the elements needed for the Equality ZK Proofs that appear in the computations protocols. The full details appear in [Rab94]. The Equality ZK Protocol appears in Appendix A.

Computing any Linear Combination of Secrets

Given two secrets u and v which are *ultimately shared*, and two non-zero constants c_1 and c_2, we would like to compute the linear combination $c_1\cdot u+c_2\cdot v$ such that it will be *ultimately shared.* The protocol in Figure 4 achieves this goal.

Our protocol relies on the fact that u and v are ultimately shared by polynomials f^u and f^v, respectively. This fact immediately guarantees that each honest player can locally compute his share of the secret $c_1\cdot u+c_2\cdot v$. The self correcting property is achieved by having each player prove, using a zero knowledge technique, that he is proceeding according to the protocol, i.e. that he in fact shared (in Step 3) the linear combination of his shares of u and v.

Protocol Linear Combination (Sketch)

Code for player P_i:

1. Set $\beta_i^u=L^u(i)$ and $\beta_i^v=L^v(i)$.
2. Set $\alpha_i^u=c_1\cdot\beta_i^u$ and $\alpha_i^v=c_2\cdot\beta_i^v$.
3. Share using (generic) Two&Sum-AWSS the values α_i^u, α_i^v, and their sum $\alpha_i\stackrel{\Delta}{=}\alpha_i^u+\alpha_i^v$.
4. Prove that $\alpha_i^u=c_1\cdot\beta_i^u$ and that $\alpha_i^v=c_1\cdot\beta_i^v$, using the Equality ZK Proof (Fig. 7).

Figure 4: Computing Linear Combination

Lemma 3 *Given that* u *and* v *are ultimately shared by polynomials* f^u *and* f^v *respectively, and two constants* c_1 *and* c_2, *the protocol* Linear Combination *(Figure 4) achieves that the secret* $c_1\cdot u+c_2\cdot v$ *will be ultimately shared.*

Proof: As u and v are ultimately shared, there exist shares $\beta_1^l,\ldots,\beta_n^l$ and protocols $L^l(\cdot)$ and $G^l(\cdot)$ satisfying Definition 2 for $l\in\{u,v\}$. As we further assume that u and v are shared by polynomials f^u and f^v, we know that $\beta_i^l=f^l(i)$ $(1\le i\le n)$, and that $f^l(0)=l$ for $l\in\{u,v\}$.

We have to show that there exist shares $\beta_1, \ldots \beta_n$ and protocols $L(\cdot), G(\cdot)$ and *Rec* satisfying Definition 2 for $c_1 \cdot u + c_2 \cdot v$.
We define the following:
1. $\beta_i \stackrel{\Delta}{=} c_1 \cdot \beta_i^u + c_2 \cdot \beta_i^v$.
2. $L(i)$ is to locally computes $c_1 \cdot L^u(i) + c_2 \cdot L^v(i)$.
3. $G(i)$ is AWSS-Rec of the value α_i shared in Step 3.
4. The protocol *Rec* is to interpolate a polynomial $f(x)$ through any $t+1$ shares from $\alpha_1, \ldots, \alpha_n$ which are not *null* and output $f(0)$.

We will prove that these definitions satisfy the desired properties:
Secrecy. The only steps of the protocol that involve communication are those of AWSS-Share and the Equality ZK Proof. As both these sub-protocols are secure, no additional information about about u nor v can leak to the faulty players.
Correctness(1). By assumption $L^u(i) = \beta_i^u$ and $L^v(i) = \beta_i^v$. Hence by definition $L(i) = c_1 \cdot L^u(i) + c_2 \cdot L^v(i) = c_1 \cdot \beta_i^u + c_2 \cdot \beta_i^v = \beta_i$.
Correctness(2). By definition $G(i)$ is AWSS-Rec of the value shared by P_i in Step 3. If the output of this AWSS-Rec is not null, then due to the properties of Two&Sum-AWSS it must be $\alpha_i = \alpha_i^u + \alpha_i^v$. From the Equality Proof it follows that $\alpha_i^u = c_1 \cdot \beta_i^u$ and that $\alpha_i^v = c_2 \cdot \beta_i^v$, Hence, by the definition of β_i, $G(i) = \alpha_i = \beta_i$.
Correctness(3). As $\beta_1^u, \ldots, \beta_n^u$ define the polynomial f^u and $\beta_1^v, \ldots, \beta_n^v$ define the polynomial f^v, then clearly by the definition of β_i it is true that $\beta_1, \ldots, \beta_n$ define the polynomial $f \stackrel{\Delta}{=} c_1 \cdot f^u + c_2 \cdot f^v$. When the honest players execute *Rec* with the outputs of $G(1), \ldots, G(n)$, each one of them interpolates through $t+1$ non-*null* α_i's. Since $\alpha_i = \beta_i$, each player computes f, and outputs $f(0) = c_1 \cdot u + c_2 \cdot v$. □

Note: This protocol can be easily extended to compute any number of linear combinations.

Computing the Multiplication of Two Secrets

Given two secrets u and v which are *ultimately shared*, we would like to compute the product $u \cdot v$, such that this secret will be *ultimately shared.*

As in the case of Linear Combination, each player has a share of the product secret, but this share is on a polynomial of degree $2t$, which in addition, is not a random polynomial. As was shown in [BGW88], by computing linear transformations on the shares of the players, we can both randomize and reduce the degree of the polynomial. Each player can locally compute his share of the product $u \cdot v$, but this share is not ultimately shared, which is what we need in order to compute the linear combinations. Thus, in the first step each player will USS-Share his product share, and prove that he indeed shared the proper value. Once these operations are done, and the linear combinations have been computed, there is a random polynomial f of degree $\leq t$ with constant term $u \cdot v$, and all the shares of f are ultimately shared. The Multiplication Protocol is given in Figure 5.

Protocol Multiplication (Sketch)

Protocol for Player P_i:

1. Set $\beta_i^u = L^u(i)$ and $\beta_i^v = L^v(i)$.
2. Set $\alpha_i^u = \beta_i^u$ and $\alpha_i^v = \beta_i^v$.
3. Compute $\alpha_i \stackrel{\Delta}{=} \alpha_i^u \cdot \alpha_i^v$.
4. Share α_i using the USS-Share Protocol (Fig. 1). Share α_i^u and α_i^v using AVSS-Share.
5. Prove that $\alpha_i^u \cdot \alpha_i^v = \alpha_i$, using Product ZKP protocol.
6. Prove that $\alpha_i^u = \beta_i^u$ and $\alpha_i^v = \beta_i^v$, using the Equality ZK Proof (Fig. 7).
7. Execute Protocol Agree (Fig. 3) with the following boolean predicate: $Q(j) = 1$ if player P_j completed Steps 4-6 successfully. Set $CompSet_i$ to the output of the protocol.
8. To perform the randomization and degree reduction, use the Linear Combination Protocol (Fig. 4), and compute the transformations described in [BGW88], on the set of values agreed upon in $CompSet_i$. Let the randomized reduced polynomial be $f(x)$.

Figure 5: Computing Multiplication

Lemma 4 *Given that u and v are ultimately shared by polynomials f^u and f^v respectively, the protocol* Multiplication *(Figure 5) achieves that the secret $u \cdot v$ will be ultimately shared.*

Proof: As u and v are ultimately shared, there exist shares $\beta_1^l, \ldots, \beta_n^l$ and protocols $L^l(\cdot)$ and $G^l(\cdot)$ satisfying Definition 2 for $l \in \{u, v\}$. As we further assume that u and v are shared by polynomials f^u and f^v, we know that $\beta_i^l = f^l(i)$ $(1 \leq i \leq n)$, and that $f^l(0) = l$ for $l \in \{u, v\}$.

We have to show that there exist shares $\beta_1, \ldots \beta_n$ and protocols $L(\cdot), G(\cdot)$ and *Rec* satisfying Definition 2 for $u \cdot v$.

We define the following:

1. $\beta_i \triangleq f(i)$, where f is the polynomial computed in Step 8.
2. $L(i)$ to be the private USS-Rec of the value $f(i)$.
3. $G(i)$ to be the public USS-Rec of $f(i)$.
4. The protocol *Rec* is to interpolate a polynomial $f'(x)$ through any $t+1$ shares from $\beta_1, ..., \beta_n$ which are not *null* and output $f'(0)$.

We will prove that these definitions satisfy the desired properties:

Secrecy. The steps of the protocol that involve communication of relevant data are those of USS-Share, AVSS-Share, the two forms of Equality Proof, and the Linear Combination computation. As all these subprotocols are secure, no additional information about u nor v can leak to the faulty players.

Correctness(1). Each α_i is ultimately shared (Step 4). Hence, by lemma 3, all the linear combinations computed at Step 8 are ultimately shared, i.e. for $1 \leq i \leq n$ $f(i)$ is ultimately shared. Thus, by the definition of β_i and the properties of USS-Rec, $L(i) = \beta_i$.

Correctness(2). $G(i) = L(i)$ due to the properties of multiple reconstruction of USS-Rec.

Correctness(3). By assumption $\beta_i^u = L^u(i)$ and $\beta_i^v = L^v(i)$ From the proofs of player P_i in Steps 5 and 6, we are guaranteed that the value α_i that he shared in Step 4 is the correct product share, i.e. $\beta_i^u \cdot \beta_i^v$. As all the honest players will complete this sharing, at least $2t+1$ product shares will be ultimately shared, and the players will complete Step 7 with a common subset. As the product shares lie on a polynomial of degree $\leq 2t$, any subset of at least $2t+1$ shares defines this polynomial, and hence will suffice for the randomization and the degree reduction operations. From [BGW88] we are guaranteed that the linear transformation which is computed creates a random polynomial $f(x)$ of degree $\leq t$ which satisfies that $f(0) = u \cdot v$. From our Linear Combination protocol we are insured that each share $f(i)$ is ultimately shared. Thus any $t+1$ shares from $G(1), \ldots, G(n)$ define f, and the polynomial f' reconstructed by *Rec* equals f, so the output of *Rec* will be $f'(0) = f(0) = u \cdot v$. □

Putting it all together, we have proved Theorem 1.

6 Impossibility Result

In this section we prove that if $n \leq 4t$, then any t-resilient asynchronous AVSS scheme must have some positive probability of not terminating.

Proof (Sketch):

Let $n = 4, t = 1$, and let D, A, B and C be the four players. Given an AVSS protocol the adversary can take control of the dealer D and delay all the messages sent to and from C. Note that only D is faulty, and C's communications are simply slow.

Let M_{XY}^v be the messages exchanged between X and Y during the execution of the protocol when D shares the value $v \in \{0,1\}$. From the 1-privacy condition the messages A sees, M_{AD}^0 and M_{AB}^0 when $v = 0$ must be compatible also with $v = 1$. Therefore there are messages M_{DB}^1 that D can exchange with B that are compatible with M_{AB}^0. D will not send any messages to C.

Now during the reconstruction stage the adversary stops D and lets A, B and C exchange messages between them. From A's and C's point of view, the real situation looks completely identical to the case where B is faulty, and D is slow. A must therefore wait for additional shares from D or C to reconstruct the secret but these will never arrive (C will never receive messages from D, messages from A to C cannot help in the reconstruction, and messages from B can be wrong because B may be faulty.) Thus with exponentially small probability, by guessing correctly the messages exchanged between A and B, the adversary can force the system to wait indefinitely. □

References

[BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. "Asynchronous Secure Computations". In *Proceeding 25th Annual Symposium on the Theory of Computing*, pages 52–61. ACM, 1993.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations". In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 1–10. ACM, 1988.

[BOEY91] M. Ben-Or and R. El-Yaniv. "Interactive Consistency in Constant Time". Submitted for publication, 1991.

[Bra84] G. Bracha. "An Asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus Protocol". In *3rd Proceeding of Distributed Computing*, pages 154–162. ACM, 1984.

[Can93] R. Canetti. "Asynchronous Secure Computations". Technical Report 755, Department of Computer Science, Technion Haifa, 1993.

[CCD88] D. Chaum, C. Crepau, and I. Damgard. "Multiparty Unconditionally Secure Proto-

cols". In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 11–19. ACM, 1988.

[CR93] R. Canetti and T. Rabin. "Fast Asynchronous Byzantine Agreement with Optimal Resilience". In *Proceeding 25th Annual Symposium on the Theory of Computing*, pages 42–51. ACM, 1993.

[Fel89] P. Feldman. "Asynchronous Byzantine Agreement in Constant Expected Time". Manuscript, 1989.

[FM88] P. Feldman and S. Micali. "An Optimal Algorithm for Synchronous Byzantine Agreement". In *Proceeding 20th Annual Symposium on the Theory of Computing*, pages 148–161. ACM, 1988.

[MR] S. Micali and P. Rogoway. "Secure Computations". In preparation.

[PSL80] M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults". *Journal of the ACM*, 27(2):228–234, 1980.

[Rab89] T. Rabin. "Robust Sharing of Secrets When the Dealer is Honest or Faulty". Master's thesis, Hebrew University, Jerusalem, 1989.

[Rab94] T. Rabin. *"Asynchronous Secret Sharings and Implementations"*. PhD thesis, Hebrew University, Jerusalem, 1994.

[RB89] T. Rabin and M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority". In *Proceeding 21st Annual Symposium on the Theory of Computing*, pages 73–85. ACM, 1989.

A Proving that Two Shared Secrets are Equal

In the course of our protocols, there are many times where a player P will need to prove that some secret s that he shared is equal to a secret s' that was previously shared among the players. There is a confidence parameter ϵ, and each one of the (honest) players should be convinced that the two secrets are equal with probability at least $1-\epsilon$. In the process of the proof P does not want to reveal any additional information, aside from the fact that the two secrets are indeed equal.

Each one of the secrets s and s' may be shared by any secret sharing scheme. Furthermore, s' may have been shared by any player (not necessarily P), or computed collectively by all the players.

In this section we present a protocol that solves a slight variant of the problem, which is, given a constant $c \in E - \{0\}$, P can prove that the newly shared secret s is equal to $c \cdot s'$. When $c = 1$, this is a simple equality test.

To achieve this goal, we assume that whenever a secret s is shared among the players, it is accompanied by a set R_s of auxiliary random secrets, shared by the same secret sharing scheme as s. The number of the auxiliary secrets is polynomial in $(\lceil \log \frac{1}{\epsilon} \rceil, n)$. This assumption is easily achieved if whenever a secret is shared, the generic sharing protocol of Figure 6 is used. For the moment disregard the input R' (assume $R' = \phi$).

Protocol Generic-Share$[R', \epsilon]$

Code for the Dealer D, sharing a secret s:

1. Set $K \triangleq \text{poly}(\lceil \log \frac{1}{\epsilon} \rceil, n)$.
2. Choose, with uniform distribution, $r_1, \ldots, r_K \in_R E$.
   Set $R_s \triangleq \{r_1, \ldots, r_K\}$.
3. Two&Sum-Share* s and $r_1, s + r_1, \ldots, r_K, s + r_K$ and $r', s + r'$ for each $r' \in R'$.

* Using any desired sharing scheme.

Figure 6: Generic Sharing

It will also be shown later, that if s is computed collectively by the players (from previously shared secrets), then the appropriate auxiliary secrets may be computed as well.

Assume now that P wants to share a secret s, and prove that it equals $c \cdot s'$, where s' was previously shared. Then P executes the generic share once more, and this time he takes as input to the protocol the set $R' = c \cdot R_{s'} \triangleq \{c \cdot r' \mid r' \in R_{s'}\}$. Note that if s' was not shared by P, then he must initiate private reconstruction for the secrets in the set $R_{s'}$. This time P creates two sets of auxiliary secrets: one which will serve as the auxiliary set for s, i.e. R_s, and the second will be used just for the equality proof. After all this has been accomplished, P can turn to the players to execute the Equality ZK Proof (Figure 7). The input to the protocol will be the secrets s and $c \cdot R_{s'}$ with its appropriate sums, and s' and $R_{s'}$ with its appropriate sums.

Lemma 5 *The protocol* Equality ZK Proof *(Figure 7) achieves the following properties:*

Protocol Equality ZK Proof$[c, s, R, s', R']$

Code for player P_i:

1. For each $j, 1 \leq j \leq K$ such that $j \equiv i \pmod{n}$, randomly choose whether to expose the pair $(s + r_j, s' + r'_j)$ or the pair (r_j, r'_j), and A-Cast your choice.
2. Reconstruct the components of all the A-Casted pairs.
3. Conclude that $s = c \cdot s'$ only if in each one of the reconstructed pairs the first component equals c times the second component.

Figure 7: Equality ZK Proof

1. If $s = c \cdot s'$ then all the honest players will be convinced of this fact.
2. If $s \neq c \cdot s'$, then the probability that an honest player will falsely believe that $s = c \cdot s'$ is at most ϵ.
3. No additional information about s leaks to any player.

Proof:
1. This is clear, because all the equality tests will succeed.
2. Suppose that $s \neq c \cdot s'$. Then, in order to succeed in showing that $s + r_j = c \cdot (s' + r'_j)$, it must be that $r_j \neq c \cdot r'_j$, but in order to succeed showing that $r_j = c \cdot r'_j$, they must be equal. Since P cannot know in advance which pair P_i will want to expose, the probability that one test will succeed is $\leq 1/2$. Since there are K independent tests, the claim holds.
3. Because the r_j's are uniformly distributed, so are all the pairs $(s + r_j, s' + r'_j)$. In addition, the Two&Sum protocol reveals no information, and hence the claim holds. □

B Generating a Random Secret Polynomial

We will need the ability to create random secret polynomials in our protocol, in order to randomize the polynomial which is the result of a multiplication. The polynomials should have a constant term equal to zero. The protocol in Figure 8 achieves this.

Protocol Random Polynomial Generator (Sketch)

Protocol for player P_i:

1. Set $s_i = 0$.
2. Share s_i among the players using USS-Share.
3. Prove that $s_i = 0$.
4. Execute Protocol Agreement (for agreeing on a subset) with the boolean predicate: $Q(j) = 1$ iff P_j has completed the previous two steps.
   Set $CompSet_i$ to the output of Agreement$[Q]$.
5. Compute the sum of the secrets of the players in $CompSet_i$.

Figure 8: Random Polynomial Generation

C Computing Auxiliary Values for Computed Secrets

As we have stated earlier, for the Equality Proof each secret should be accompanied by a set of auxiliary random secrets. This is easily achieved when the secret is shared by a dealer, but we need these auxiliary random secrets even if the secret has been computed by the network.

As the secrets which have been computed by the network are ultimately shared, we can over come the problem. Say that a secret s was computed by the network, and that some player P wants to share s', and prove that $s = s'$. P chooses random secrets $r_1, ..., r_K$. He shares $s', r_1, s' + r_1, \ldots, r_K, s' + r_K$ using the desired sharing scheme, and he USS-Shares $r_1, ..., r_K$. The network computes $s + r_1, \ldots, s + r_K$. Then P proceeds to carry out the Equality ZK Proof protocol (Figure 7).