Efficient Asynchronous Multiparty Computation with Optimal Resilience

Arpita Patra **Ashish Choudhary** **C. Pandu Rangan**
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai India 600036
Email:{ arpita,ashishc }@cse.iitm.ernet.in, rangan@iitm.ernet.in

Abstract

In this paper, we propose a new *asynchronous multiparty computation* (AMPC) protocol that achieves information theoretic security with optimal fault tolerance; i.e., with $n = 3t + 1$, where n is the total number of players and t is the number of players which can be under the control of an adversary $\mathcal{A}_t$ with *unbounded computing power* in Byzantine (active) fashion. Our AMPC protocol provides information theoretic security with negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. Our AMPC protocol communicates $\mathcal{O}(n^5\kappa)$ bits per multiplication. As far as our knowledge is concerned, the only known AMPC protocol with $n = 3t + 1$ providing information theoretic security with negligible error probability is due to [8], which communicates $\Omega(n^{11}\kappa^4)$ bits per multiplication. [1]. Thus our AMPC protocol significantly improves the communication complexity of the AMPC protocol of [8]. For designing our AMPC protocol, we introduce a new asynchronous primitive called Asynchronous Ultimate Verifiable Secret Sharing (AUVSS) [2] which is first of its kind and is of independent interest.

[1] The exact communication complexity analysis of the AMPC protocol of [8] was not done earlier and we have carried out the same. The details will be presented in the full version of the paper.

[2] The outcome of AUVSS is different from the outcome of Ultimate VSS (UVSS) introduced in [8]; the difference is clearly pointed out later in section 4 of this paper.

1 Introduction

Secure Multiparty Computation: Secure multiparty computation (MPC) [26] allows a set of n players to securely compute an agreed function, even if up to t players are under the control of a centralized adversary, having *unbounded computing power*. More specifically, assume that the desired functionality can be specified by a function $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ and player P_i has input $x_i \in \{0,1\}^*$. At the end of the computation of f, P_i gets $y_i \in \{0,1\}^*$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. The function f has to be computed securely using a MPC protocol where at the end of the protocol all (honest) players receive correct outputs and the messages seen by the adversary during the protocol contain no *additional* information about the inputs and outputs of the honest players, other than what can be computed from the inputs and outputs of the corrupted players. The MPC problem has been studied extensively in the past two decades in synchronous networks (see [7, 12, 25, 2, 19, 22, 5, 20, 15, 14, 3, 5] and their references), which assumes that there is a global clock and the delay of any message in the network is bounded by a constant. However, though theoretically impressive, such networks do not model adequately real life networks like the internet.

Asynchronous Networks: In asynchronous networks, messages are delayed arbitrarily. As a worst case assumption, the adversary is given the power to schedule the delivery of messages. Such networks models real life networks like the Internet much better than their synchronous counterpart. However, protocols for asynchronous networks are much more involved than their synchronous counterparts. This is so because if a player does not receive an expected message then he cannot decide whether the sender is corrupted (and did not send the message at all) or the message is just delayed in the network. Thus in a fully asynchronous settings, it is impossible to consider the inputs of all uncorrupted players. So input of up to t (potentially honest) players have to be ignored because waiting for them could turn out to be endless. Also, the existing protocols and techniques used in synchronous settings cannot be trivially extended in asynchronous settings. For a comprehensive introduction to asynchronous protocols, the readers may refer to [10].

Asynchronous Multiparty Computation: Though MPC protocols in synchronous settings has been studied extensively, *asynchronous multiparty computation* (AMPC) has got comparatively very less attention due to its complexity. It is known that AMPC protocol with information theoretic security and *zero error* (i.e., *perfectly secure* AMPC) is possible iff $n \geq 4t + 1$ [6]. On the other hand, AMPC protocol with information theoretic security and *negligible error probability* is possible iff $n \geq 3t+1$ [8]. Currently the best known communication efficient *perfectly secure* AMPC protocol with $n = 4t+1$ is due to [4], which communicates $\mathcal{O}(n^3)$ field elements per multiplication. However, the only known AMPC protocol that achieves information theoretic security with *negligible error probability* and optimal fault tolerance (i.e., with $n = 3t + 1$) is due to [8], which communicates $\Omega(n^{11}\kappa^4)$ bits and A-Casts $\Omega(n^{11}\kappa^2 \log(n))$ bits per multiplication and has an error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. Here A-Cast is an asynchronous broadcast primitive, which allows a player to send the same information to all the other players. In [24], the authors have presented an AMPC protocol that achieves information theoretic security with negligible error probability with $n = 4t + 1$ (i.e with non-optimal resilience), whose communication complexity is significantly less than that of [8]. Thus as far our knowledge is concerned, the only known AMPC protocol with $n = 3t + 1$, is due to [8]. Though the communication complexity of the AMPC protocol of [8] is polynomial, it is too high. This motivates us to design communication efficient AMPC protocol with $n = 3t + 1$.

Approach Used in the AMPC Protocol of [8]: We now briefly explain the approach used by [8] to design their AMPC protocol. The current description is taken from [8]. The protocol consists of three stages. In the first stage, each player P_i *commits* his input value(s) x_i to all other players. In the second stage, the players agree on a common subset $CompSet$ of at least $n - t$ players who have properly committed their inputs. Finally, in the last stage, the players will compute the function $f(x_1, x_2, \ldots, x_n)$, where x_i is the input value committed by each $P_i \in CompSet$ and $x_i = 0$ otherwise.

To implement the first stage, each P_i commits his input x_i by using an asynchronous primitive called *Ultimate (Asynchronous) Verifiable Secret Sharing* (UVSS) [8]. Let us first recall the definition of Asynchronous Verifiable Secret Sharing (AVSS) [11]. Roughly speaking, AVSS allows a dealer (pos-

sibly corrupted) to share a secret among the players that can be reconstructed by the players at a later stage. AVSS forces a faulty dealer to *commit* a specific value that is guaranteed to be reconstructed later. In [11], the authors have designed an AVSS scheme with $n = 3t+1$, which has a negligible error probability and a negligible probability of non-termination. But as explained in [8], the AVSS scheme of [11] cannot be directly used for committing the input values in AMPC protocol. This is because the AVSS scheme of [11] with $n = 3t+1$ has the property that in the worst case, at most $n-t = 2t+1$ players can only obtain the shares of the committed secret and it may happen that these $2t+1$ players contain only $t+1$ honest players. But to implement AMPC with $n = 3t+1$, we require that all the honest players should obtain the shares of the $\mathbf{D}$'s committed secret. For this, the authors in [8] have extended the AVSS protocol of [11] to introduce a new asynchronous primitive called *Ultimate (Asynchronous) Verifiable Secret Sharing* (UVSS), which makes sure that all the honest players will eventually obtain their share of the committed secret. So now in the AMPC protocol each player commits his input using the UVSS protocol. In the second stage, the players agree on a common subset *CompSet* of at least $n-t$ players who have properly committed their inputs. Once this is done, the players compute the function $f(x_1, x_2, \ldots, x_n)$ using the general method of [7]. Thus the players simulate an arithmetic circuit for f, gate by gate, in such a way that the intermediate results are always kept as secrets, distributed among the players.

Our Contribution: In this paper, we design a new AMPC protocol with $n = 3t+1$, which achieves information theoretic security with a negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. Our AMPC protocol communicates $\mathcal{O}(n^5\kappa)$ bits per multiplication. This is a significant improvement over the AMPC protocol of [8] which communicates $\Omega(n^{11}\kappa^4)$ bits and A-Casts $\Omega(n^{11}\kappa^2 \log(n))$ bits per multiplication. For designing our AMPC protocol, we introduce a new asynchronous primitive called Asynchronous Ultimate VSS (AUVSS) (the outcome of AUVSS is different from the outcome of UVSS of [8]; the difference is clearly pointed out later in section 4) which is first of its kind and is of independent interest. Also we adapt few existing techniques from synchronous MPC protocols of [14, 15]into asynchronous settings for designing our AMPC protocol.

2 Preliminaries

2.1 Model

We consider a set of n players denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$, who are pairwise connected by secure asynchronous channels. An adversary $\mathcal{A}_t$ with unbounded computing power can control at most $t < \frac{n}{3}$ players in Byzantine fashion. By Byzantine corruption we mean that the adversary can make the corrupted players to deviate from the protocol in any arbitrary manner. Moreover the adversary is active, adaptive and rushing [14]. As a worst case assumption, we assume that $\mathcal{A}_t$ can schedule the transmission of messages along the channels. Thus he can arbitrarily delay any message. However every sent message will be eventually delivered. Notice that $\mathcal{A}_t$ can only delay the transmission of the messages sent by honest players, without having any access to them.

The function to be computed is specified by an arithmetic circuit over $\mathbb{F}$, consisting of input, linear (e.g. addition), multiplication, random and output gates. We denote the number of gates of each type by c_I, c_L, c_M, c_R and c_O respectively. Our AMPC protocol evaluates the circuit gate by gate and provides information theoretic security with negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. To bound the error probability by $2^{-\mathcal{O}(\kappa)}$, all our computations are performed over a finite field $\mathbb{F}$, where $\mathbb{F} = GF(2^\kappa)$. Thus each field element can be represented by κ bits. Moreover, we assume that n is also polynomial in κ.

We also assume that our protocols are executed in steps. Each step begins by the scheduler choosing one message (out of the queue) to be delivered to its designated recipient. The recipient is activated by receiving the message, after which he performs some local computation and possibly send messages on his outgoing channel. If the received message refers to a sub-protocol which is not yet "in execution", then the player keeps the message until the relevant sub-protocol is invoked.

2.2 A-cast, Agreement on a Core Set (ACS) and AVSS

A-Cast[11]: It is an asynchronous broadcast primitive, which was introduced and elegantly implemented by Bracha [9] with $n = 3t+1$. Let Π be an asynchronous protocol initiated by a special player (called the sender), having input m (the message to be broadcast). We say that Π is a t-resilient A-cast protocol if the following holds, for every possible $\mathcal{A}_t$ and input:

• **Termination**:

1. If the sender is honest and all the honest players participate in the protocol, then each honest player will eventually complete the protocol.
2. Irrespective of the behavior of the sender, if any honest player completes the protocol then each honest player will eventually complete the protocol.

• **Correctness**: If the honest players complete the protocol then they have a common output m^*. Furthermore, if the sender is honest then $m^* = m$.

Agreement on Core Set (ACS)[4]: It is a primitive presented in [6, 8]. We use it to determine a set of $n - t$ players that correctly shared their values. More concretely, every player starts the ACS protocol with an accumulative set of players who from his view point correctly shared one or more values (the share sub-protocols in which they acted as dealers terminated properly). The output of the protocol is a set of at least $n - t$ players, who correctly shared their values.

Asynchronous Verifiable Secret Sharing (AVSS) [11]: Let (Sh, Rec) be a pair of protocols in which a dealer $\mathbf{D} \in \mathcal{P}$ shares a secret S containing $\ell \geq 1$ field elements. We say that (Sh, Rec) is a t-resilient, $(1 - 2^{-O(\kappa)})$ terminating AVSS scheme for n players if the following hold for every possible $\mathcal{A}_t$.

• **Termination**: With probability at least $1 - 2^{-O(\kappa)}$, the following requirements hold:

1. If $\mathbf{D}$ is honest then each honest player will eventually terminate protocol Sh.
2. If some honest player has completed protocol Sh, then irrespective of the behavior of $\mathbf{D}$, each honest player will eventually terminate Sh.
3. If an honest player has completed Sh and all the honest players invoke protocol Rec, then each honest player will terminate Rec.

• **Correctness**:

1. If $\mathbf{D}$ is honest then with probability at least $1-2^{-O(\kappa)}$, each honest player upon completing protocol Rec, outputs the shared secret.
2. Once the first honest player completes protocol Sh, then there exists an $r \in \mathbb{F}^\ell$, such that with probability at least $1 - 2^{-O(\kappa)}$, each honest player upon completing Rec, will output r.

• **Secrecy**: If $\mathbf{D}$ is honest and no honest player has begun executing protocol Rec, then the corrupted players have no information about the shared secret.

2.3 Random Value Generation

We now design a simple asynchronous protocol called RandomGenerator, which allows the players to jointly generate a random number $r \in \mathbb{F}$. The idea behind RandomGenerator is very simple: Each player $P_i \in \mathcal{P}$ with input $x_i \in_R \mathbb{F}$ participates in an instance of asynchronous multiparty computation (AMPC) protocol described in [8], where the function to be computed is $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and each $y_i = (x_1 + x_2 + \ldots + x_n)$. Now r is nothing but $(x_1 + x_2 + \ldots + x_n)$. Thus at the termination of the AMPC instance, every player in $\mathcal{P}$ will have a value r which is completely random.

3 AMPC Protocol Overview

Our AMPC protocol proceeds in three phases: preparation phase, input phase and computation phase. Every honest player will eventually complete every phase with very high probability. We call a triple (a, b, c) as a random multiplication triple if a, b are random and $c = ab$. In the preparation phase,

sharings of $c_M + c_R$ random multiplication triples will be generated in parallel. Each multiplication and random gates of the circuit (representing the function f to be jointly computed by the n players) will be associated with a multiplication triple. Actually a random gate uses only the first component of the associated triple (a, b, c), namely a. In the input phase the players share (commit) their inputs and agree on a core set of $n - t$ players who correctly shared their inputs (every honest player will eventually get a share of the inputs of the players in the core set). In the computation phase, the actual circuit will be computed gate by gate, based on the inputs of the players in core set. Due to the linearity of the used secret-sharing, the linear gates can be computed locally without communication. Each multiplication gate will be evaluated using the circuit randomization technique of Beaver [1] with the help of a multiplication triple (generated in preparation phase) associated with it.

4 Generating t-1D-Sharing

In this section, we present a novel protocol that allows a dealer **D** (possibly corrupted) to generate correct t-1D-sharing of **D**'s secrets in an asynchronous settings. The t-1D-sharing of a single secret s is defined as follows:

Definition 1 *t-1D-Sharing: We say that a value s is correctly t-1D-shared among the players in $\mathcal{P}$ if* **every honest** *player $P_i \in \mathcal{P}$ is holding a share s_i of s, such that there exists a degree t polynomial $f(x)$ over $\mathbb{F}$ with $f(0) = s$ and $f(j) = s_j$ for every $P_j \in \mathcal{P}$. The vector $(s_1, s_2, \ldots, s_n)$ of shares is called a t-1D-sharing of s and is denoted by $[s]_t$. We may skip the subscript t when it is clear from the context. A set of shares (possibly incomplete) is called t-consistent if these shares lie on a t degree polynomial.*

In synchronous world, verifiable secret sharing (VSS) is employed to achieve correct t-1D-sharing. Hence lot of research work has been carried out to design VSS [7, 25, 18, 17, 21] in synchronous settings. A very important property that can be easily achieved by synchronous VSS but not by asynchronous VSS (AVSS) with $n = 3t + 1$ players is the following: **Every honest player** in $\mathcal{P}$ will have the share of the **D**'s committed secret at the end of sharing phase. We call this property as Ultimate-Property. In literature the first ever AVSS with $n = 3t + 1$ players is proposed by [11] which lacks Ultimate-Property. Thus the AVSS protocol of [11] is unable to generate t-1D-sharing. This very reason motivated the authors of [8] to enhance the definition of AVSS to accommodate Ultimate-Property along with the remaining properties of AVSS and thus the definition of Ultimate VSS (UVSS) had emerged. Also the inherent difficulty in using an AVSS protocol without Ultimate-Property (such as AVSS protocol of [11]) in AMPC is nicely pointed out in the introduction section of [8]. Then [8] has designed an UVSS protocol using AVSS protocol of [11] and some zero knowledge technique from [12].

With the almost same incentive in mind, in this paper we now introduce a new concept (asynchronous primitive) called AUVSS which is defined as follows:

Definition 2 *Asynchronous Ultimate Verifiable Secret Sharing (AUVSS): Let (Sh, Rec) be a pair of protocols in which a dealer $\mathbf{D} \in \mathcal{P}$ shares a secret S containing ℓ field elements. We say that (Sh, Rec) is a t-resilient $(1 - 2^{-\mathcal{O}(\kappa)})$ terminating AUVSS scheme for n players if for every possible $\mathcal{A}_t$ and input, the protocols satisfy the termination, correctness and secrecy property of AVSS, along with Ultimate-Property.*

According to our definition, the outcome of a correct AUVSS Sh protocol is that **every honest player** will eventually hold the correct sharing of **D**'s committed secret. Here is the discernable difference between the outcome of UVSS share protocol of [8] and our AUVSS share protocol: The UVSS share protocol of [8] allows **D** to commit the individual shares of the committed secret using AVSS protocol of [11] and then employ zero knowledge from [12] to ensure the committed shares indeed define a t degree polynomial. After this they leave the shares in the committed fashion (and does not reconstruct them towards respective players) and perform the MPC. But our AUVSS share protocol end with correct t-1D-sharing of the committed secret.

To design our AUVSS protocol, we almost follow the same track of [8] except that we go one step ahead to reconstruct the shares towards respective players and end with t-1D-sharing of the committed secret. In addition to this difference, our AUVSS is strikingly better than the UVSS of [8] in terms of communication complexity. While our AUVSS share communicates $O(\ell n^4\kappa) + \mathsf{poly}(n,\kappa)$ bits and A-casts $n^5\log(n)$ bits, the UVSS share protocol of [8] communicates $O(n^{10}\kappa^4)$ bits and A-casts $\mathcal{O}(n^{10}\kappa^2\log(n))$ bits. To design our AUVSS, we borrow the AVSS-Weak and AVSS-Weak-Share-MultiSet protocol of [23]. In the sequel we briefly recall the properties of these two protocols.

Protocol AVSS-Weak: This protocol consists of three sub-protocols: AVSS-Weak-Share, AVSS-Weak-Rec-Private and AVSS-Weak-Rec-Public. Protocol AVSS-Weak-Share is an AVSS protocol with the weaker property that (corrupted) $\mathbf{D}$'s committed secret S can be $NULL$. Thus AVSS-Weak-Share satisfies all the properties of AVSS (share) except that it does not force $\mathbf{D}$ to commit a secret which is not $NULL$. Hence $S \in \mathbb{F}^\ell \cup NULL$. AVSS-Weak-Share communicates $O((\ell n^3 + n^4\kappa)\kappa)$ bits for sharing ℓ secrets. While AVSS-Weak-Rec-Private performs private reconstruction of the secret S to a specific player P_α, AVSS-Weak-Rec-Public allows reconstruction of the secret S towards every player in $\mathcal{P}$. We recall the following lemma from [23].

Lemma 1 *Protocol AVSS-Weak-Share communicates $O((\ell n^3 + n^4\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^4\log(n))$ bits. Protocol AVSS-Weak-Rec-Private and AVSS-Weak-Rec-Public communicates $O((\ell n^3 + n^4\kappa)\kappa)$ and $O((\ell n^5 + n^5\kappa)\kappa)$ bits, respectively.*

In protocol **AVSS-Weak**, the above sub-protocols are invoked in the following way: AVSS-Weak$(\mathbf{D}, \mathcal{P}, S)$, AVSS-Weak-Rec-Private$(\mathbf{D}, \mathcal{P}, S, P_\alpha)$ and AVSS-Weak-Rec-Public$(\mathbf{D}, \mathcal{P}, S, \mathcal{P})$.

Protocol AVSS-Weak-Share-MultiSet: This protocol is the extension of AVSS-Weak-Share. Precisely, the goal of AVSS-Weak-Share-MultiSet is to share $\mathcal{M}$ sets of ℓ secrets at once such that later any linear combination of the $\mathcal{M}$ sets can be reconstructed. More clearly, AVSS-Weak-Share-MultiSet can share $S^m = (s^{(m,1)}, \ldots, s^{(m,\ell)})$ for $m = 1, \ldots, \mathcal{M}$ at once such that later $S^* = \sum_{m=1}^{\mathcal{M}} r_m S^m = (\sum_{m=1}^{\mathcal{M}} r_m s^{(m,1)}, \ldots, \sum_{m=1}^{\mathcal{M}} r_m s^{(m,\ell)})$ can be reconstructed using Protocol AVSS-Weak-Rec-Private or AWSS-Weak-Rec-Public for some $(r_1, \ldots, r_{\mathcal{M}}) \in \mathbb{F}^{\mathcal{M}}$. Now here is a very important observation: Let r be a random number generated with the help of all (honest) players after the execution of AVSS-Weak-Share-MultiSet. Now if we reconstruct the linear combination $S^* = \sum_{m=1}^{\mathcal{M}} r^{m-1} S^m$ then with very high probability it will be $NULL$ if one of the set S^m shared by $\mathbf{D}$ was $NULL$. The reason for this follows the same line of the argument given in section 4.4 of [15] and is explained in full details in [23]. Thus with very high probability, we can detect a corrupted $\mathbf{D}$ who did not share all the sets meaningfully. Also by the reconstruction the information theoretic security on S^1 will be lost. But the remaining set $S^2, \ldots, S^{\mathcal{M}}$ remains secure. To avoid this, we can make AVSS-Weak-Share-MultiSet to share $\mathcal{M}+1$ sets where the first set S^0 can be used for padding the remaining sets of secrets, namely $S^1, \ldots, S^{\mathcal{M}}$. This provides a good clue of how AVSS-Weak-Share-MultiSet can be used to ensure that $\mathbf{D}$'s committed secrets are all meaningful. We implement the above idea in our protocol AUVSS-Share which will be given in the sequel. AVSS-Weak-Share-MultiSet retains all the properties of AVSS-Weak-Share. We now recall the following lemma on the communication complexity of AVSS-Weak-Share-MultiSet

Lemma 2 ([23]) *AVSS-Weak-Share-MultiSet communicates $O((\ell n^3 + n^4\kappa)\mathcal{M}\kappa)$ bits and A-casts $n^4\log(n)$ bits.*

AVSS-Weak-Share-MultiSet is invoked as AVSS-Weak-Share-MultiSet$(\mathbf{D}, \mathcal{P}, S^1, \ldots, S^{\mathcal{M}})$

Here we stress that AVSS-Weak-Share-MultiSet, satisfying weaker property of allowing the committed secret to be $NULL$ is enough for our implementation of AUVSS protocol. The reason is that in our AUVSS protocol we use verification mechanism which ensures both the following: (a) the committed secrets in AVSS-Weak-Share-MultiSet is non-NULL or *meaningful*, (b) the sharing done by $\mathbf{D}$ is indeed t-1D-sharing. We would like to mention that our verification is simpler and much efficient than the one used in [8] to upgrade AVSS scheme of [11] into UVSS scheme (they called the verification as zero knowledge proof in [8]).

We are now ready to present our AUVSS protocol called AUVSS which share ℓ secrets. The high level idea of the share protocol of AUVSS is as follows: $\mathbf{D}$ divides the ℓ secrets into n^2 sets $S^1, \ldots, S^{n^2}$,

where set S^m contains the secrets $s^{(m,1)}, \ldots, s^{(m,L)}$, where $L = \lceil \frac{\ell}{n^2} \rceil$. Thus each set contains L secrets. In addition, $\mathbf{D}$ puts another L random numbers in a set S^0. Now to share the secrets in set S^m, $\mathbf{D}$ selects degree-t univariate polynomials $f^{(m,1)}(x), \ldots, f^{(m,L)}(x)$. Let the i^{th} shares of these polynomials be denoted by $P^{(i,m)} = (f^{(m,1)}(i), \ldots, f^{(m,L)}(i))$. Now $\mathbf{D}$ shares $P^{(i,0)}, P^{(i,1)}, \ldots, P^{(i,n^2)}$ together using the AVSS-Weak-Share-Multiset protocol. Now as explained earlier, if $\mathbf{D}$ is corrupted, then protocol AVSS-Weak-Share-Multiset protocol does not ensure whether $\mathbf{D}$ is sharing non-$NULL$ (or meaningful values). To check whether $\mathbf{D}$ has AVSS shared values from $\mathbb{F}$ and not $NULL$, the players in $\mathcal{P}$ do the following: once $\mathbf{D}$ has completed AVSS-Weak-Share-Multiset, the players in $\mathcal{P}$ jointly generate a random number r. Now the players locally compute the sharing of $P^{(i,*)} = P^{(i,0)} + rP^{(i,1)} + \ldots + r^{n^2} P^{(i,n^2)}$ and then $P^{(i,*)}$ is publicly reconstructed toward each player. Notice that $P^{(i,*)}$ is an L tuple, where the l^{th} component of $P^{(i,*)}$ is $f^{(0,l)}(i) + rf^{(1,l)}(i) + \ldots + r^{n^2} f^{(n^2,l)}(i)$. Now from Theorem 2 of [23], if each component of reconstructed $P^{(i,*)}$ is non-$NULL$, then with very high probability each individual element of shared (using AVSS-Weak-Share-Multiset) $P^{(i,0)}, P^{(i,1)}, \ldots, P^{(i,n^2)}$ was non-$NULL$. Notice that by publicly reconstructing $P^{(i,*)}$, information theoretic security on $P^{(i,1)}, \ldots, P^{(i,n^2)}$ is still maintained.

However, even though each component of each reconstructed $P^{(i,*)}$ is non-$NULL$, it does not imply that the values shared by (corrupted) $\mathbf{D}$ using different instances of AVSS-Weak-Share-Multiset are points on a t degree polynomial. For this the players perform an additional check. Notice that the l^{th} component of each $P^{(i,*)}$ together defines the polynomial $f^{(*,l)}(x) = f^{(0,l)}(x) + rf^{(1,l)}(x) + \ldots + r^{n^2} f^{(n^2,l)}(x)$. Ideally, if $\mathbf{D}$ is honest, then each of the polynomials $f^{(0,l)}(x), f^{(1,l)}(x), \ldots, f^{(n^2,l)}(x)$ would be of degree t and hence the resultant polynomial $f^{(*,l)}(x)$ would also be of degree t. However, even if one of the $(n+1)$ polynomials is of degree more than t, then with very high probability, the resultant polynomial $f^{(*,l)}(x)$ will be also of degree more that t. The argument for this is same as given in section 4.4 of [15]. Thus, once each $P^{(i,*)}$ is publicly reconstructed, the players check whether the l^{th} component of each $P^{(i,*)}$, together defines a t degree polynomial. If not, then (corrupted) $\mathbf{D}$ has selected some more than degree t polynomial to share at least one value in the sets $S^0, S^1, \ldots, S^{n^2}$.

Thus the above two verifications ensure that with very high probability, $\mathbf{D}$ has shared non-$NULL$ values and each value are points on t degree polynomials. If both the verifications pass then $P^{(i,1)}, P^{(i,2)}, \ldots, P^{(i,n^2)}$ are privately reconstructed towards player P_i, who will thus obtain the i^{th} share of each of the ℓ secrets. The reconstruction of AUVSS protocol can be achieved using *On-line Error Correcting Technique* explained in [6, 10]. It is easy to see that our protocol AUVSS satisfies termination, correctness for honest $\mathbf{D}$, secrecy, and Ultimate-Property given that AVSS-Weak-Share-MultiSet satisfies termination, correctness and secrecy properties and RandomGenerator satisfies termination and correctness properties. So we prove the correctness of AUVSS when $\mathbf{D}$ is corrupted.

Lemma 3 *Protocol AUVSS satisfies correctness property even if* $\mathbf{D}$ *is corrupted.*

Proof: Correctness 2: Here we prove that if every value in $P^{(i,*)}$ for $i = 1, \ldots, n$ is some value from $\mathbb{F}$ (i.e. non-NULL) and l^{th} value from all sets $P^{(i,*)}$ together define a degree-t univariate polynomial, then with very high probability, all the secrets are non-$NULL$ and shared using degree-t univariate polynomials. First, by Theorem 2 of [23], our verification ensures that none of the values in sets $P^{(i,m)}$ for all $i \in \{1, \ldots, n\}$ and $m \in \{0, \ldots, n^2\}$ are $NULL$ with very high probability if every value in $P^{(i,*)}$ for $i = 1, \ldots, n$ is some value from $\mathbb{F}$ (i.e. non-NULL). So now assuming the values in sets $P^{(i,m)}$ for all $i \in \{1, \ldots, n\}$ and $m \in \{0, \ldots, n^2\}$ are values from $\mathbb{F}$, we prove that the sharing for the secrets are done using degree-t polynomials. Recall that $f^{(m,1)}(x), \ldots, f^{(m,L)}(x)$ are the polynomials corresponding to the L secrets in S^m. Essentially in the protocol, we check whether every polynomial $f^{(*,l)}(x) = \sum_{m=0}^{n^2} r^m f^{(m,l)}(x)$ for $l = 1, \ldots, L$ is a degree-t polynomial or not. Following the argument given in [15], it can be easily proved that $f^{(*,l)}(x)$ will be of degree more than t with very high probability, if one of the polynomials $f^{(0,l)}(x), \ldots, f^{(n^2,l)}(x)$ is of degree more than t and r is a random number generated after the completion of AVSS-Weak-Share-MultiSet's. Hence if a corrupted $\mathbf{D}$ share some secret using polynomial of degree more than t, he will be caught with very high probability. So if $\mathbf{D}$ is not caught and if the AUVSS-Share protocol terminates, then from the properties of *on line error correction* [10], same degree-t polynomials (as committed by $\mathbf{D}$ in AUVSS-Share) will be reconstructed in AUVSS-Rec-Private and AUVSS-Rec-Public. □

AUVSS($\mathbf{D}, \mathcal{P}, S$)

AUVSS-Share($\mathbf{D}, \mathcal{P}, S$):

SHARING BY $\mathbf{D}$:

1. $\mathbf{D}$ first divides ℓ secrets equally into n^2 sets, namely $S^1, \ldots, S^{n^2}$. So every set S^m contains $\lceil \frac{\ell}{n^2} \rceil$ secrets. If n^2 does not divide ℓ, then add random numbers in the last set so that it contains $\lceil \frac{\ell}{n^2} \rceil$ secrets. $\mathbf{D}$ also selects $\lceil \frac{\ell}{n^2} \rceil$ random numbers from $\mathbb{F}$ to put in a set S^0.
2. For every set S^m, $\mathbf{D}$ selects $L = \lceil \frac{\ell}{n^2} \rceil$ degree-t univariate polynomials $f^{(m,1)}(x), \ldots, f^{(m,L)}(x)$ such that $f^{(m,l)}(0) = s^{(m,l)}$ for $l = 1, \ldots, L$. Let $P^{(i,m)}$ is the set containing the values of the polynomials $f^{(m,1)}(x), \ldots, f^{(m,L)}(x)$ evaluated at i. Thus $P^{(i,m)} = (f^{(m,1)}(i), \ldots, f^{(m,L)}(i))$ for every $i \in \{1, \ldots, n\}$ and $m \in \{0, 1, \ldots, n^2\}$.
3. $\mathbf{D}$ executes AVSS-Weak-Share-MultiSet($\mathbf{D}, \mathcal{P}, P^{(i,0)}, P^{(i,1)}, \ldots, P^{(i,n^2)}$) for $i = 1, \ldots, n$. We denote the i^{th} instance as AVSS-Weak-Share-MultiSet^i.

VERIFICATION STEP: CODE FOR P_j

1. Upon completion of AVSS-Weak-Share-MultiSet($\mathbf{D}, \mathcal{P}, P^{(i,0)}, P^{(i,1)}, \ldots, P^{(i,n^2)}$) for all $i = 1 \ldots, n$, player P_j participates in protocol RandomGenerator to generate a random number r.
2. Once r is generated player P_j locally computes the sharings of $P^{(i,*)} = \sum_{m=0}^{n^2} r^m P^{(i,m)}$.
3. P_j participates in AVSS-Weak-Rec-Public($\mathbf{D}, \mathcal{P}, P^{(i,*)}, \mathcal{P}$) to reconstruct $P^{(i,*)}$ towards every player in $\mathcal{P}$. Carefully notice that for an honest $\mathbf{D}$, the l^{th} value from all sets $P^{(i,*)}$ together will define $f^{(*,l)}(x) = \sum_{m=0}^{n^2} r^m f^{(m,l)}(x)$ which is a degree-t univariate polynomial.
4. P_j waits for the completion of all instances of AVSS-Weak-Rec-Public and checks if all the L values in $P^{(i,*)}$ for $i = 1, \ldots, n$ belongs to $\mathbb{F}$ and l^{th} value from all sets $P^{(i,*)}$ together defines a degree-t univariate polynomial. If yes then P_j sets $Happy_j = 1$. Otherwise P_j sets $Happy_j = 0$.

GENERATION OF t-1D-SHARING: CODE FOR P_j:

1. If $Happy_j = 1$, P_j participates in AVSS-Weak-Rec-Private($\mathbf{D}, \mathcal{P}, P^{(i,m)}, P_i$) for $m = 1, \ldots, n^2$ to give the i^{th} share of all the secrets to P_i.
2. P_j waits for the termination AVSS-Weak-Rec-Private($\mathbf{D}, \mathcal{P}, P^{(j,m)}, P_j$) for $m = 1, \ldots, n^2$ and then terminates. Now eventually all the (honest) players will have shares of all the secrets in S.

AUVSS-Rec-Private($\mathcal{P}, S, P_\alpha$): Private Reconstruction towards P_α:

1. A single secret $s \in S$ is recovered by P_α in the following way: CODE FOR P_j: Player P_j sends the j^{th} share of s to P_α. Upon receiving at least $2t + 1$ t-consistent shares P_α interpolates a degree-t polynomial and takes the constant term of it as the secret s. This method of reconstruction is called *On-line error correction* [6, 10].

AUVSS-Rec-Public($\mathcal{P}, S, \mathcal{P}$): Public Reconstruction towards $\mathcal{P}$:

1. Run AVSS-Rec-Private($\mathbf{D}, \mathcal{P}, S, P_\alpha$) for every $P_\alpha \in \mathcal{P}$ parallely.

Theorem 1 *AUVSS-Share communicates* $\mathcal{O}((\ell n^4 \kappa) + \mathsf{poly}(n, \kappa))$ *bits and A-casts* $\mathcal{O}(n^5 \log(n))$ *bits. AUVSS-Rec-Private and AUVSS-Rec-Public communicate* $\mathcal{O}(\ell n \kappa)$ *and* $O(\ell n^2 \kappa)$ *bits respectively.*

PROOF: Protocol AUVSS-Share calls n instances of AVSS-Weak-Share-MultiSet with $n^2 + 1$ sets, each containing $\lceil \frac{\ell}{n^2} \rceil$ values. Hence by Lemma 2, it incurs a communication cost $O(\ell n^4 \kappa) + \mathsf{poly}(n, \kappa)$ bits and A-casts of $n^5 \log(n)$ bits. Generation of random number requires communication complexity independent of secret size ℓ. Again AUVSS-Share calls n instances of AVSS-Weak-Rec-Public with $P^{(i,*)}$ containing $\lceil \frac{\ell}{n^2} \rceil$ values. This requires $O(\ell n^4 \kappa) + \mathsf{poly}(n, \kappa)$ bits of communication complexity. Finally AUVSS-Share calls n^2 instances of AVSS-Weak-Rec-Private each with $\lceil \frac{\ell}{n^2} \rceil$ values. This requires $O(\ell n^4 \kappa) + \mathsf{poly}(n, \kappa)$ bits of communication. □

5 Generating t-2D-Sharing

In this section, we present a novel protocol that allows a dealer $\mathbf{D}$ (possibly corrupted) to generate correct t-2D-sharing of $\mathbf{D}$'s secrets in an asynchronous settings. The t-2D-sharing of a single secret s is defined as follows:

Definition 3 (t-2D-sharing [3]) *: We say that a value s is correctly t-2D-shared among the players in $\mathcal{P}$ if there exists t degree polynomials $f, f^1, f^2 \ldots, f^n$ with $f(0) = s$ and for $i = 1, \ldots, n$, $f^i(0) = f(i)$.*

Moreover, **every (honest) player** $P_i \in \mathcal{P}$ *holds a share* $s_i = f(i)$ *of* s*, the polynomial* $f^i(x)$ *for sharing* s_i *and a share-share* $s_{ji} = f^j(i)$ *of the share* s_j *of every player* $P_j \in \mathcal{P}$*. We denote* t*-2D-sharing of* s *as* $[[s]]_t$*.*

If a secret s is t-2D-shared by a dealer $\mathbf{D} \in \mathcal{P}$ (any player from $\mathcal{P}$ may perform the role of a dealer), then we denote the sharing by $[[s]]_t^{\mathbf{D}}$. Notice that when a secret s is t-2D-shared, then s is t-1D-Shared and it's shares are also individually t-1D-shared. Now we present a protocol t-2D-Share which allows $\mathbf{D}$ to generate t-2D-sharing of ℓ secrets. Protocol t-2D-Share works in the following way: Let the ℓ secrets of $\mathbf{D}$ is denoted by $s^1, \ldots, s^\ell$. In addition to this, $\mathbf{D}$ selects a random value $s^0 \in \mathbb{F}$. Now $\mathbf{D}$ selects $(\ell+1)$ degree-t univariate polynomials $f^0(x), \ldots, f^\ell(x)$ such that $f^l(0) = s^l$ for $l = 0, \ldots, \ell$. $\mathbf{D}$ then t-1D-shares $S^0 = (f^0(0), \ldots, f^\ell(0)) = (s^0, \ldots, s^\ell)$ using AUVSS-Share protocol. In addition to this, $\mathbf{D}$ also t-1D-shares $S^1, \ldots, S^n$, where $S^i = (f^0(i), \ldots, f^\ell(i))$ using AUVSS-Share. If $\mathbf{D}$ is honest then the problem is solved because he will correctly t-1D-share each S^i. However, a corrupted $\mathbf{D}$ may share $\overline{S^i} \neq S^i$. Now to check whether $\mathbf{D}$ has t-1D-shared S^i and not $\overline{S^i} (\neq S^i)$, the players do the following. Once $\mathbf{D}$ has finished all the $(n+1)$ instance of AUVSS-Share, the players in $\mathcal{P}$ jointly generate a random r. Then each player P_j locally compute the j^{th} share s_j^* of secret s^* (where $s^* = f^*(0) = f^0(0) + rf^1(0) + \ldots + r^\ell f^\ell(0) = s^0 + rs^1 + \ldots + r^\ell s^\ell$) from the j^{th} shares of $s^0, \ldots, s^\ell$ (which he has already got from the instance of AUVSS-Share, used to t-1D-share $S^0 = (s^0, \ldots, s^\ell)$). He also locally computes the j^{th} share of s_i^*, namely s_{ij}^* from the j^{th} shares of $s_i^0, \ldots, s_i^\ell$ (which he has already got from the instance of AUVSS-Share, used to t-1D-share $S^i = (s_i^0, \ldots, s_i^\ell)$) for all $i = 1, \ldots, n$. Let s_{ji}^* be the i^{th} share of polynomial $f_j^*(x)$. Now ideally, if $\mathbf{D}$ is honest, then $f^*(j) = f_j^*(0)$ should hold. So once the players locally compute all the required shares, they publicly reconstruct $f^*(x)$ and each polynomial $f_j^*(x)$. Once this is done, the players check whether $f^*(j) = f_j^*(0)$ holds for each j. If so, then with very high probability, $\mathbf{D}$ has indeed t-1D-shared $S^0, S^1, \ldots, S^n$. Now to complete the t-2D-sharing of the secrets, every player P_i must know the polynomials used by $\mathbf{D}$ to t-1D-share the shares of the secrets. For that the polynomials used by $\mathbf{D}$ for sharing i^{th} shares of all the ℓ secrets are reconstructed towards only player P_i.

t-2D-Share$(\mathbf{D}, \mathcal{P}, S)$

Sharing by $\mathbf{D}$:

1. $\mathbf{D}$ first chooses $\ell+1$ degree-t random polynomials $f^0(x), f^1(x), \ldots, f^\ell(x)$ such that $f^l(0) = s^l$ for $l = 0, \ldots, \ell$ and $s^0 \in \mathbb{F}$ is a random value chosen by $\mathbf{D}$. Let $S^i = (f^0(i), \ldots, f^\ell(i)) = (s_i^0, \ldots, s_i^\ell)$ for $i = 0, 1, \ldots, n$.
2. $\mathbf{D}$ now invokes AUVSS-Share$(\mathbf{D}, \mathcal{P}, S^i)$ for $i = 0, \ldots, n$ to generate t-1D-sharing of the secrets S^0 and their shares S^i for $i = 1, \ldots, n$. We call i^{th} instance of AUVSS-Share as AUVSS-Share^i

Verification: Code for P_j

1. Upon completion of AUVSS-Share^i for all $i = 0, \ldots, n$, player P_j participates in protocol RandomGenerator to generate a random value $r \in \mathbb{F}$. Notice that once player P_j completes all AUVSS-Share^i's, he has already got j^{th} shares of the secrets in S^0 which is $\overline{S^j}$ (for an honest $\mathbf{D}$, $S^j = \overline{S^j}$) and also j^{th} share-shares of the shares in S^i for $i = 1, \ldots, n$.
2. Once r is generated P_j locally computes j^{th} shares of $s_i^* = \sum_{l=0}^{\ell} r^l s_i^l$ for $i = 1, \ldots, n$ and $s^* = \sum_{l=0}^{\ell} r^l s^l$. Thus P_j computes $s_{ij}^* = \sum_{l=0}^{\ell} r^l s_{ij}^l$ for $i = 1, \ldots, n$ and $s_j^* = \sum_{l=0}^{\ell} r^l s_j^l$ where s_{ij}^l denotes the j^{th} share-share of s_i^l .
3. P_j participates in AUVSS-Rec-Public$(\mathbf{D}, \mathcal{P}, s^*, \mathcal{P})$ and AUVSS-Rec-Public$(\mathbf{D}, \mathcal{P}, s_i^*, \mathcal{P})$ for $i = 1, \ldots, n$ to reconstruct $s^*, s_1^*, \ldots, s_n^*$ towards every player in $\mathcal{P}$. Upon completion of all the instances of AUVSS-Rec-Public, player P_j obtains reconstruction polynomials $f^*(x)$ and $f_1^*(x), \ldots, f_n^*(x)$ with $f^*(0) = s^*$ and $f_i^*(0) = s_i^*$. Now he checks whether $f^*(i) \stackrel{?}{=} f_i^*(0) = s_i^*$ for $i = 1, \ldots, n$. If this is not true, then $\mathbf{D}$ has not done proper t-1D-sharing and hence P_j sets $\mathsf{Ver}_j = 0$. Otherwise P_j sets $\mathsf{Ver}_j = 1$.

Reconstruction of polynomials used for sharing i^{th} shares of secrets towards P_i: Code for P_j:

1. If $\mathsf{Ver}_j = 1$, then P_j participates in AUVSS-Rec-Private$(\mathbf{D}, \mathcal{P}, s_i^l, P_i)$ for $l = 1, \ldots, \ell$. This ensures that upon completion of AUVSS-Rec-Private$(\mathbf{D}, \mathcal{P}, s_i^l, P_i)$, player P_i will have $f^{(i,l)}(x)$ which was used by $\mathbf{D}$ to t-1D-share s_i^l, the i^{th} share of secret s^l.
2. P_j waits for the completion of AUVSS-Rec-Private$(\mathbf{D}, \mathcal{P}, s_j^l, P_j)$ for $l = 1, \ldots, \ell$. As a part of AUVSS-Rec-Private, P_j gets $f^{(j,l)}(x)$ which was used by $\mathbf{D}$ to t-1D-share s_j^l, the j^{th} share of secret s^l and terminates.

Lemma 4 *Protocol t-2D-Share generates correct t-2D-sharings of* ℓ *secrets.*

PROOF: The termination of protocols AUVSS-Share$(\mathbf{D}, \mathcal{P}, S^i)$ for $i = 0, \ldots, n$ ensures proper generation of t-1D-sharing of the secrets S^0 and supposedly their shares S^i for $i = 1, \ldots, n$. Let due to the execution of AUVSS-Share$(\mathbf{D}, \mathcal{P}, S^0)$, player P_j has got S^j, the j^{th} shares on S^0. We have to ensure that $\mathbf{D}$ has indeed done t-1D-sharing of shares S^j in the instance AUVSS-Share^j for all honest P_j. Let $\mathbf{D}$ has done t-1D-sharing of shares $\overline{S^j}$ in protocol AUVSS-Share^j with $\overline{S^j} \neq S^j$ for an honest P_j. Then with very high probability, $f^*(j) \neq f^*_j(0)$. The reason is that $f^*(j) = s^*_j = \sum_{l=0}^{\ell} r^l s^l_j$ will be different from $\overline{s^*_j} = \sum_{l=0}^{\ell} r^l \overline{s^l_j}$ with probability of at least $1 - \frac{\ell}{2^\kappa} \approx 1 - 2^{-O(\kappa)}$. In that case Ver_i will be 0 for every honest $P_i \in \mathcal{P}$. Hence if $\mathsf{Ver}_i = 1$ for every honest P_i, then with very high probability $\mathbf{D}$ has indeed done t-1D-sharing of shares S^j in the instance AUVSS-Share^j for all honest P_j. □

Theorem 2 *t-2D-Share communicates $\mathcal{O}(\ell n^5 \kappa) + \mathsf{poly}(n, \kappa)$ bits and A-casts $\mathcal{O}(n^6 \log(n))$ bits.*

PROOF: Protocol t-2D-Share calls $n + 1$ instances of AUVSS-Share with $\ell + 1$ secrets each. Hence by Theorem 1, it incurs a communication cost of $O(\ell n^5 \kappa) + \mathsf{poly}(n, \kappa)$ bits and A-cast of $\mathcal{O}(n^6 \log(n))$ bits. The execution of RandomGenerator requires communication complexity independent of ℓ. The $n + 1$ instances of AUVSS-Rec-Public requires $O(n^3 \kappa)$ bits of communication. Finally, ℓn instances of AUVSS-Rec-Private requires $O(\ell n^2 \kappa)$ bits of communication. □

6 Preparation Phase

The goal of the preparation phase is to generate correct t-1D-sharings of $c_M + c_R$ secret random triples (a^k, b^k, c^k), such that $c^k = a^k b^k$ for $k = 1, \ldots, c_M + c_R$. For this we first generate t-2D-sharing of secret random doubles (a^k, b^k) for $k = 1, \ldots, c_M + c_R$. Then we generate c^k (i.e $[c^k]_t$) given $([[a^k]]_t, [[b^k]]_t)$.

6.1 Generating Secret Random t-2D-Sharing

In section 5, we have presented a protocol called t-2D-Share which allows a $\mathbf{D}$ to generate t-2D-sharing of ℓ secrets. Here we present a protocol called Random-t-2D-Share which generates random t-2D-sharing of ℓ secrets. The idea of the protocol is taken from [4]. Random-t-2D-Share asks individual player to act as dealer and to t-2D-Share $\frac{\ell}{n-2t}$ random secrets. Then we run ACS protocol to agree on a core set of $n-t$ players who have correctly t-2D-shared $\frac{\ell}{n-2t}$ random secrets. Among the $n-t$ players in core-set at most t can be corrupted and hence the corresponding shared secrets are known to the adversary and (possibly) non-random. But the remaining $\frac{\ell}{n-2t}$ random secrets for each of the $n - 2t$ honest players are unknown to the adversary and completely random. Hence we use a *Vandermonde Matrix* to extract $\frac{\ell}{n-2t} * (n - 2t) = \ell$ t-2D-sharings out of $\frac{\ell}{n-2t} * (n - t)$ t-2D-sharings (done by the players in core set). Next, we briefly recall *Vandermonde Matrix* and it's ability to extract randomness.

Vandermonde Matrix and Randomness Extraction [15]: We express a matrix $M \in F^{(r,c)}$ with r rows and c columns as $M = \{m_{i,j}\}_{i=1,\ldots,r}^{j=1,\ldots,c}$. We use M^T to denote the transpose of M. For distinct elements $\beta_1, \ldots, \beta_r$ from $\mathbb{F}$, we use $V^{(r,c)}$ to denote the Vandermonde Matrix $V^{(r,c)} = \{\beta_i^j\}_{i=1\ldots,r}^{j=0,\ldots,c-1}$. Now we can use $V^{(r,c)}$ for randomness extraction. Let $U = V^{(r,c)^T}$ be the transpose of $V^{(r,c)}$ with $r > c$. Now assume that $(x_1, \ldots, x_r)$ is generated by picking up c elements from $\mathbb{F}$ uniformly at random and then picking the remaining $r - c$ elements from $\mathbb{F}$ with an arbitrary distribution independent of the first c elements. Now we compute $(y_1, \ldots, y_c) = U(x_1, \ldots, x_r)$. $(y_1, \ldots, y_c)$ is an uniformly random vector of length c extracted from $(x_1, \ldots, x_r)$. We now present our protocol Random-t-2D-Share.

Lemma 5 *Protocol Random-t-2D-Share (eventually) terminates with very high probability for every honest player. It outputs ℓ t-2D-sharings of random secret values. It communicates $\mathcal{O}(\ell n^5 \kappa) + \mathsf{poly}(n, \kappa)$ bits, A-Cast $\mathcal{O}(n^7 \log(n))$ bits and requires one invocation to ACS.*

6.2 Proving $c = ab$

Consider the following problem: let $\mathbf{D} \in \mathcal{P}$ holds ℓ pairs of values $(a^1, b^1), \ldots, (a^\ell, b^\ell)$ such that all of them are correctly t-1D-shared among the players in $\mathcal{P}$. Now $\mathbf{D}$ wants to correctly t-1D-share

Random-t-2D-Share($\mathcal{P}, \ell$)

CODE FOR P_i

1. P_i as a dealer invokes t-2D-Share($P_i, \mathcal{P}, S^i$) to generate t-2D-sharing of $L = \frac{\ell}{n-2t}$ secrets in $S^i = (s^{(i,1)}, \ldots, s^{(i,L)})$.
2. For $j = 1, \ldots, n$, P_i takes part in t-2D-Share($P_j, \mathcal{P}, S^j$).

AGREEMENT ON A CORE-SET: CODE FOR P_i

1. P_i creates an accumulative set $C^i = \emptyset$.
2. Upon completing t-2D-Share($P_j, \mathcal{P}, S^j$) with dealer P_j, P_i includes P_j in C^i.
3. P_i takes part in ACS with the accumulative set C^i as input.

GENERATION OF RANDOM t-2D-SHARING: CODE FOR P_i:

1. P_i waits until ACS completes with output C. For simple notation, assume that $\{P_1, \ldots, P_{n-t}\} \in C$.
2. For every $k \in \{1, \ldots, L\}$, the $t+1$ random values, unknown to the adversary, are defined as $r^{(1,k)}, \ldots, r^{(n-2t,k)} = U(s^{(1,k)}, \ldots, s^{(n-t,k)})$, where U denotes a $(n-2t) \times (n-t)$ transpose of a *Vandermonde Matrix*. So P_i locally computes shares of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$ for every $k \in \{1, \ldots, L\}$ by computing $(r_i^{(1,k)}, \ldots, r_i^{(n-2t,k)}) = U(s_i^{(1,k)}, \ldots, s_i^{(n-t,k)})$ where $(s_i^{(1,k)}, \ldots, s_i^{(n-t,k)})$ denotes i^{th} shares of $(s^{(1,k)}, \ldots, s^{(n-t,k)})$. Now to generate t-2D-sharing of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$, players must compute t-1D-sharing of $(r_j^{(1,k)}, \ldots, r_j^{(n-2t,k)})$ from the t-1D-sharing of $(s_j^{(1,k)}, \ldots, s_j^{(n-t,k)})$. P_i computes i^{th} shares of $(r_j^{(1,k)}, \ldots, r_j^{(n-2t,k)})$ by computing $(r_{ji}^{(1,k)}, \ldots, r_{ji}^{(n-2t,k)}) = U(s_{ji}^{(1,k)}, \ldots, s_{ji}^{(n-t,k)})$ where $(s_{ji}^{(1,k)}, \ldots, s_{ji}^{(n-t,k)})$ denotes i^{th} shares of $(s_j^{(1,k)}, \ldots, s_j^{(n-t,k)})$.

$c^1, \ldots, c^\ell$ without leaking any *additional* information about a^l, b^l and c^l, such that every (honest) player in $\mathcal{P}$ knows that $c^l = a^l b^l$ for $l = 1, \ldots, \ell$. We propose a protocol ProveCeqAB to achieve this task. The idea of the protocol is inspired from [14] which is as follows. We try to explain the idea of the protocol with a single pair (a, b). Thus $\mathbf{D}$ has two values a and b such that they are already t-1D-shared. Now he wants to generate t-1D-sharing of c, where $c = ab$, without leaking any *additional* information about a, b and c. For this, he first selects a random non-zero $\beta \in \mathbb{F}$ and generates t-1D-sharing of c, β and βb. All the players in $\mathcal{P}$ then jointly generate a random value r. Then everybody computes the sharing of $p = ra + \beta$ locally and then p is reconstructed towards every player by public reconstruction (AUVSS-Rec-Public). Then again every player locally computes the sharing of $q = pb - b\beta - rc = (ra + \beta)b - b\beta - rc$ ($[q]_t = p[b]_t - [b\beta]_t - r[c]_t$) and then q is reconstructed towards every player by public reconstruction (AUVSS-Rec-Public). Player P_i believes that c is correctly t-1D-shared by $\mathbf{D}$ with $c = ab$ if $q = 0$.

The error probability of the protocol is negligible because of the random r which is jointly generated by all the players after c, β and $b\beta$ is t-1D-shared. Specifically, a corrupted $\mathbf{D}$ might have shared $\overline{\beta b} \neq \beta b$ or $\overline{c} \neq c$ but still q can be zero and this will happen iff $\overline{\beta b} + r\overline{c} = \beta b + rc$. However this equation is satisfied by only one value of r. Since r is randomly generated, independent of $\mathbf{D}$, the probability that the equality will hold is $\frac{1}{|\mathbb{F}|}$ which is negligibly small. The secrecy follows from the fact that p and q are independent of a, b and c. Now we can extend the above idea parallely for each of the ℓ pairs $(a^{(l)}, b^{(l)})$.

Lemma 6 *Protocol ProveCeqAB communicates $O((\ell n^4 \kappa) + \mathsf{poly}(n, \kappa))$ bits. Protocol ProveCeqAB terminates for an honest $\mathbf{D}$ with very high probability.*

6.3 Generating Secret Random Triples; The Preparation Phase Main Protocol

We now present protocol PreparationPhase which generates $c_M + c_R$ secret random triples (a^k, b^k, c^k). The protocol works as follows: First protocol Random-t-2D-Share is invoked to generate (a^k, b^k) for $k = 1, \ldots, c_M + c_R$. Then a player locally multiplies the shares of (a^k and b^k) and then t-1D-shares the multiplied value with the proof that he indeed shared the correct value. To accomplish this, the player invokes ProveCeqAB. Finally, an instance of ACS protocol will be executed to agree on a core set of $n - t$ players whose ProveCeqAB has terminated. Then we apply *Lagranges interpolation formula* on the sharing done by the players in core set to obtain t-1D-sharings of c^k. We explain the idea with a single pair, say (a, b). Given that a and b are correctly t-2D-shared among the players in $\mathcal{P}$,

ProveCeqAB($\mathbf{D}, \mathcal{P}, [a^1]_t, \ldots, [a^\ell]_t, [b^1]_t, \ldots, [b^\ell]_t$)

SHARING BY $\mathbf{D}$

1. CODE FOR $\mathbf{D}$: $\mathbf{D}$ randomly selects ℓ non-zero random values $\mathcal{B} = (\beta^1, \ldots, \beta^\ell) \in \mathbb{F}^\ell$. Let $\mathcal{C} = (a^1b^1, \ldots, a^\ell b^\ell) = (c^1, \ldots, c^\ell)$ and $\Lambda = (b^1\beta^1, \ldots, b^\ell\beta^\ell) = (d^1, \ldots, d^\ell)$. $\mathbf{D}$ invokes AUVSS-Share($\mathbf{D}, \mathcal{P}, \mathcal{B}$), AUVSS-Share($\mathbf{D}, \mathcal{P}, \mathcal{C}$) and AUVSS-Share($\mathbf{D}, \mathcal{P}, \Lambda$) to generate t-1D-sharing of the values in $\mathcal{B}$, $\mathcal{C}$ and Λ.
2. CODE FOR P_i: P_i participates in the AUVSS-Share protocols initiated by $\mathbf{D}$.

VERIFYING WHETHER $c = ab$: CODE FOR P_i

1. P_i waits for the termination of the three instances of AUVSS-Share (initiated by $\mathbf{D}$).
2. P_i participates in protocol RandomGenerator to generate a random value $r \in \mathbb{F}$.
3. P_i locally computes the shares of $p^l = ra^l + \beta^l$ for $l = 1, \ldots, \ell$. That is P_i computes $p_i^l = ra_i^l + \beta_i^l$ for $l = 1, \ldots, \ell$, where a_i^l and β_i^l are the i^{th} shares of a^l and β^l, respectively.
4. P_i participates in AUVSS-Rec-Public($\mathbf{D}, \mathcal{P}, (p^1, \ldots, p^\ell), \mathcal{P}$) to reconstruct p^l for $l = 1, \ldots, \ell$. Notice that p^l does not leak any information on a^l.
5. Upon reconstruction of p^l's, P_i locally computes the shares of $q^l = p^l b^l - d^l - rc^l$ for $l = 1, \ldots, \ell$. That is P_i computes $q_i^l = p^l b_i^l - d_i^l - rc_i^l$ for $l = 1, \ldots, \ell$.
6. P_i participates in AUVSS-Rec-Public($\mathbf{D}, \mathcal{P}, (q^1, \ldots, q^\ell), \mathcal{P}$) to reconstruct q^l for $l = 1, \ldots, \ell$.
7. Upon reconstruction of q^l's, P_i locally checks whether $q^l \stackrel{?}{=} 0$ for $l = 1, \ldots, \ell$. If yes then P_i terminates.

PreparationPhase($\mathcal{P}$)

CODE FOR P_i

1. P_i participates in two instances of Random-t-2D-Share($\mathcal{P}, c_M + c_R$) to generate t-2D-sharings of $a^1, \ldots, a^{c_M+c_R}$ and $b^1, \ldots, b^{c_M+c_R}$.
2. Upon termination of both the instances of Random-t-2D-Share, P_i as a dealer invokes ProveCeqAB($P_i, \mathcal{P}, [a_i^1]_t, \ldots, [a_i^{c_M+c_R}]_t, [b_i^1]_t, \ldots, [b_i^{c_M+c_R}]_t$) to generate t-1D-sharing of $c_i^1, \ldots, c_i^{c_M+c_R}$ where $a_i^1, \ldots, a_i^{c_M+c_R}$ and $b_i^1, \ldots, b_i^{c_M+c_R}$ denotes i^{th} of $a^1, \ldots, a^{c_M+c_R}$ and $b^1, \ldots, b_i^{c_M+c_R}$ respectively and $c_i^k = a_i^k b_i^k$ is the i^{th} share of c^k.
3. P_i participates in ProveCeqAB($P_j, \mathcal{P}, [a_j^1]_t, \ldots, [a_j^{c_M+c_R}]_t, [b_j^1]_t, \ldots, [b_j^{c_M+c_R}]_t$) for $j = 1, \ldots, n$.

AGREEMENT ON A CORE-SET: CODE FOR P_i

1. P_i creates an accumulative set $C^i = \emptyset$.
2. Upon completing ProveCeqAB($P_j, \mathcal{P}, [a_j^1]_t, \ldots, [a_j^{c_M+c_R}]_t, [b_j^1]_t, \ldots, [b_j^{c_M+c_R}]_t$) with dealer P_j, P_i includes P_j in C^i.
3. P_i takes part in ACS with the accumulative set C^i as input.

GENERATION OF t-1D-SHARING OF $c^1, \ldots, c^{c_M+c_R}$: CODE FOR P_i

1. P_i waits until ACS completes with output C. For simple notation, assume that $\{P_1, \ldots, P_{n-t}\} \in C$.
2. P_i locally computes share of $[c^k]_t = \sum_{j=1}^{n-t} r_j [c_j^k]_t$, where $(r_1, \ldots, r_{n-t})$ represents the recombination vector.

it implies that implicitly P_i holds a_i and b_i where a_i and b_i are the i^{th} share of a and b respectively. Now multiplying a_i and b_i, P_i obtains i^{th} share $e_i = a_i b_i$ of c where $(e_1, \ldots, e_n)$ is $2t$-1D-sharing of c. This is not what we desire. We want P_i to hold c_i such that c_i is the i^{th} share of t-1D-sharing of c. For this, each player P_i t-1D-shares the value $e_i = a_i b_i$ with the proof that e_i is indeed the multiplication of a_i and b_i (he invokes ProveCeqAB to accomplish this task). Now an instance of ACS will be executed to agree on a core set of $n - t$ players whose ProveCeqAB has been terminated. For simplicity let $P_1, \ldots, P_{n-t}$ are in core set. Then, all the players jointly hold $[e_1]_t, \ldots, [e_{n-t}]_t$. Since $e_1, \ldots, e_{n-t}$ are $n-t$ points on a $2t$ degree polynomial, say $C(x)$ whose constant term is c, by Lagrange interpolation formula [13], c can be computed as $c = \sum_{i=1}^{n-t} r_i e_i$ where $r_i = \prod_{j=1, j\neq i}^{n-t} \frac{x-j}{i-j}$. The vector $(r_1, \ldots, r_{n-t})$ is called recombination vector [13] which is public and known to every player. So for shorthand notation, we write $c = Lagrange(e_1, \ldots, e_{n-t}) = \sum_{i=1}^{n-t} r_i e_i$. Now all players compute $[c]_t = Lagrange([e_1]_t, \ldots, [e_{n-t}]_t)$, to obtain the desired output. Notice that c is the constant term of a $2t$ degree polynomial $C(x)$. So $n - t \geq 2t + 1$ t-1D-shared values $[e_1]_t, \ldots, [e_{n-t}]_t)$ are enough to do the computation.

Lemma 7 *Protocol PreparationPhase (eventually) terminates with very high probability for every hon-*

InputPhase($\mathcal{P}$)

Secret Sharing: Code for P_i

1. P_i invokes AUVSS-Share($P_i, \mathcal{P}, x_i$) to generate t-1D-sharings of his input x_i.
2. For every $j = 1, \ldots, n$, P_i takes part in AUVSS-Share($P_j, \mathcal{P}, x_j$) where P_j is dealer.

Agreement on a Core-Set: Code for P_i

1. P_i creates an accumulative set $C^i = \emptyset$.
2. Upon completing AUVSS-Share($P_j, \mathcal{P}, x_j$) with dealer P_j, P_i includes P_j in C^i.
3. P_i takes part in ACS with the accumulative set C^i as input.
4. P_i outputs the agreed core-set C and local shares of all inputs corresponding to players in C.

est player. It outputs t-1D-sharings of $c_M + c_R$ random triples. It communicates $O((c_M + c_R)n^5\kappa) +$ poly(n, κ) bits, A-Cast $\mathcal{O}(n^7 \log(n))$ bits and requires three invocations to ACS.

7 Input Phase

In the InputPhase protocol every player P_i acts as a dealer in one AUVSS-Share protocol in order to share his input x_i. However the asynchrony of the network does not allow the players to wait for more than $n - t$ AUVSS-Share protocols to be completed. In order to agree on the players whose inputs will be taken into to computation (of the circuit), one ACS protocol is run.

Lemma 8 *Protocol InputPhase (eventually) terminates for every honest player. It outputs t-1D-sharings of inputs of the players in agreed core set C. It communicates $O(c_I n^4 \kappa) +$ poly(n, κ) bits, A-Cast $\mathcal{O}(n^6 \log(n))$ bits and requires one invocation to ACS.*

8 Computation Phase

In the computation phase, the circuit is evaluated gate by gate, where all inputs and intermediate values are shared among the players. As soon as a player holds his shares of the input values of a gate, he joins the computation of the gate.

Due to the linearity of the secret-sharing scheme, linear gates can be computed locally simply by applying the linear function to the shares, i.e. for any linear function $f(,)$, a sharing $[c]_t = [f(a, b)]_t$ is computed by letting every player P_i compute $c_i = f(a_i, b_i)$. With every random gate, one random triple (from the preparation phase) is associated, whose first component is directly used as outcome of the random gate. With every multiplication gate, one random triple (from the preparation phase) is associated, which is used to compute a sharing of the product following the circuit randomization technique of Beaver [1]. So the *Circuit Randomization* [1] allows to evaluate a multiplication gate at the cost of two public reconstructions, given a preprocessed random triple. The trick is as follows: Let $z = xy$. Now z can be expressed as $z = ((x - a) + a)((y - b) + b) = (\alpha + a)(\beta + b)$, where (a, b, c) is a multiplication triple. So given $([a]_t, [b]_t, [c]_t)$, $[z]_t$ can be computed as $[z]_t = \alpha\beta + \alpha[b]_t + \beta[a]_t + [c]_t$ after reconstructing α and β publicly. The security follows from the fact that α and β are random and independent of x and y for a random triple (a, b, c).

Lemma 9 *Protocol ComputationPhase (eventually) terminates with very high probability for every honest player. Given t-1D-sharing of $c_M + c_R$ random triples, unknown to the adversary, it computes the outputs of the circuit correctly and privately, while communicating $O(n^2(c_M + c_O)\kappa)$ bits (where c_M, and c_O denote the number of multiplication and output gates in the circuit, respectively).*

9 The New AMPC Protocol

Now our new AMPC protocol AMPC for evaluating function f which is represented by a circuit containing c_I, c_L, c_M, c_R and c_O input, linear, multiplication, random and output gates, is: (1). Invoke PreparationPhase (2). Invoke InputPhase (3). Invoke ComputationPhase.

ComputationPhase($\mathcal{P}$)

For every gate in the circuit, Code for P_i

1. P_i waits until he has shares of each of the inputs of the gate
2. Depending on the type of the gate, P_i proceeds as follows:
   (a) **Input Gate:** $[s]_t = \mathsf{IGate}([s]_t)$: There is nothing to be done here.
   (b) **Linear Gate:** $[z]_t = \mathsf{LGate}([x]_t, [y]_t, \ldots)$: P_i computes his share of z as $z_i = LGate(x_i, y_i, \ldots)$.
   (c) **Multiplication Gate:** $[z]_t = \mathsf{MGate}([x]_t, [y]_t, ([a^k]_t, [b^k]_t, [c^k]_t))$: Let $([a^k]_t, [b^k]_t, [c^k]_t)$ be the triple associated with the multiplication gate. Then P_i computes his share of α and β as $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$ respectively. P_i then participates in $\mathsf{AUVSS\text{-}Rec\text{-}Public}(\mathcal{P}, \alpha, \mathcal{P})$ and $\mathsf{AUVSS\text{-}Rec\text{-}Public}(\mathcal{P}, \beta, \mathcal{P})$ to reconstruct α and β towards every player. Upon completion of both the instances of $\mathsf{AUVSS\text{-}Rec\text{-}Public}$, P_i computes his share of z as $z_i = \alpha\beta + \alpha b_i + \beta a_i + c_i$.
   (d) **Random Gate:** $[r]_t = \mathsf{RGate}([a^k]_t, [b^k]_t, [c^k]_t)$: Let $([a^k]_t, [b^k]_t, [c^k]_t)$ be the triple associated with the random gate. P_i computes his share of r as $r_i = a_i^k$.
   (e) **Output Gate:** $x = \mathsf{OGate}([x]_t)$: P_i participates in $\mathsf{AUVSS\text{-}Rec\text{-}Public}(\mathcal{P}, x, \mathcal{P})$ and reconstructs x at the termination of the protocol.

Theorem 3 *For every coalition of up to $t < n/3$ bad players, the protocol AMPC securely computes the circuit representing function f and terminates with very high probability for all the honest players. AMPC communicates $O(c_I n^4 + c_M n^5 + c_R n^5 + c_O n^2)\kappa + \mathsf{poly}(n, \kappa)$ bits, A-Cast $\mathcal{O}(n^7 \log(n))$ bits and requires 4 invocations to ACS.*

10 Conclusion

In this paper, we have designed an AMPC protocol which provides information theoretic security with negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. Our AMPC protocol communicates $\mathcal{O}(n^5\kappa)$ bits per multiplication. The only known AMPC protocol with $n = 3t + 1$ providing information theoretic security with negligible error probability is due to [8], which communicates $\Omega(n^{11}\kappa^4)$ bits and A-Casts $\Omega(n^{11}\kappa^2 \log(n))$ bits per multiplication. Thus our AMPC protocol significantly improves the communication complexity of the AMPC protocol of [8]. For designing our AMPC protocol, we introduce a new asynchronous primitive called AUVSS which is first of its kind and is of independent interest. Recently in [16], the authors have designed an MPC protocol. However, they have not considered fully asynchronous network. More specifically, they assume that the network is synchronous up to some point in the protocol. After that, the protocol is continued in a fully asynchronous setting. Under such an assumption, the authors in [16] have designed a MPC protocol which communicates $\mathcal{O}(n^2)$ field elements per multiplication. However, our MPC protocol works in a network which is fully asynchronous from the beginning. It would be interesting to see whether it is possible to reduce the communication complexity of the MPC protocol of [16] using the techniques proposed in this paper.

References

[1] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. of CRYPTO 1991*, volume 576 of *LNCS*, pages 420–432. Springer Verlag, 1991.

[2] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(4):75–122, 1991.

[3] Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In *Proc. of TCC*, pages 305–328, 2006.

[4] Z. Beerliová-Trubíniová and M. Hirt. Simple and efficient perfectly-secure asynchronous mpc. In *ASIACRYPT*, pages 376–392, 2007.

[5] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Proc. of TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer Verlag, 2008.

[6] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61, 1993.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[8] M. BenOr, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *PODC*, pages 183–192, 1994.

[9] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In 3^{rd} *ACM PODC*, pages 154 – 162, 1984.

[10] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.

[11] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proc. of STOC 1993*, pages 42–51. ACM, 1993.

[12] D. Chaum, C. Crpeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of FOCS 1988*, pages 11–19, 1988.

[13] R. Cramer and I. Damgård. *Multiparty Computation, an Introduction*. Contemporary Cryptography. Birkhuser Basel, 2005.

[14] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Proc. of EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 311–326. Springer Verlag, 1999.

[15] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Proc. of CRYPTO*, volume 4622 of *LNCS*, pages 572–590. Springer Verlag, 2007.

[16] I. Damgrd, M. Geisler, M. Krigaard, and J. Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. Cryptology ePrint Archive, Report 2008/415, 2008.

[17] M. Fitzi, J. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In *Proc. of TCC 2006*, volume 3876 of *LNCS*, pages 329–342. Springer Verlag, 2006.

[18] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *STOC*, pages 580–589, 2001.

[19] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multiparty computation. In *Proc. of ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 143–161. Springer Verlag, 2000.

[20] M. Hirt and U. M. Maurer. Robustness for free in unconditional multi-party computation. In *Proc. of CRYPTO 2001*, LNCS, 2001.

[21] J. Katz, C. Koo, and R. Kumaresan. Improving the round complexity of vss in point-to-point networks. In *ICALP(2)*, pages 499–510, 2008.

[22] J. Katz and C. Y. Koo. Round-efficient secure computation in point-to-point networks. In *Proc. of EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 311–328. Springer Verlag, 2007.

[23] A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and byzantine agreement with optimal resilience. Cryptology ePrint Archive 2008. Also available at www.cs.iitm.ernet.in/∼ashishc/draft_1.pdf.

[24] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Trading players for efficiency in unconditional multiparty computation. In *SCN*, pages 342–353, 2002.

[25] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[26] A. C. Yao. Protocols for secure computations. In *Proc. of 23rd IEEE FOCS*, pages 160–164, 1982.