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

We propose an efficient information theoretic secure *asynchronous multiparty computation* (AMPC) protocol with optimal fault tolerance; i.e., with $n = 3t + 1$, where n is the total number of parties and t is the number of parties that can be under the influence of a *Byzantine (active)* adversary $\mathcal{A}_t$ having *unbounded computing power*. Our protocol communicates $\mathcal{O}(n^5\kappa)$ bits per multiplication and involves a negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. 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 [9], which communicates $\Omega(n^{11}\kappa^4)$ bits and A-Casts $\Omega(n^{11}\kappa^2 \log(n))$ bits per multiplication. Here A-Cast is an asynchronous broadcast primitive, which allows a party to send the same information to all other parties identically. Thus our AMPC protocol shows significant improvement in communication complexity over the AMPC protocol of [9]. As a tool for our AMPC protocol, we introduce a new asynchronous primitive called *Asynchronous Complete Verifiable Secret Sharing* (ACVSS), which is first of its kind and is of independent interest. For designing our ACVSS, we employ a new *asynchronous verifiable secret sharing* (AVSS) protocol which is better than all known communication-efficient AVSS protocols with $n = 3t + 1$.

1 Introduction

Secure Multiparty Computation (MPC): Secure multiparty computation (MPC) [32] allows a set of n parties to securely compute an agreed function f, even if up to t parties are under the control of a centralized adversary, having *unbounded computing power*. More specifically, assume that the agreed function f can be expressed as $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ and party 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) parties 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 parties, other than what can be computed from the inputs and outputs of the corrupted parties. The MPC problem has been studied extensively over synchronous networks (see [8, 13, 30, 3, 21, 26, 6, 22, 16, 15, 4] and their references), which assumes that there is a global clock and the delay of any message in the network channels is bounded. However, such networks do not model adequately real life networks like Internet.

Asynchronous Networks: Asynchronous networks model real life networks like the Internet much better than their synchronous counterpart. Here the communication channels between the parties have arbitrary, yet finite delay (i.e the messages are guaranteed to reach eventually). To model this, the adversary is given the power to schedule the delivery of messages. The inherent difficulty in designing a protocol in asynchronous network, comes from the fact that when a party 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. Therefore it is impossible to consider the inputs of all uncorrupted parties in case of AMPC. So input of up to t (potentially honest) parties may get ignored because waiting for them could turn out to be endless. Moreover the asynchronity of network calls for a complete new set of primitives for designing protocols. For a comprehensive introduction to asynchronous protocols, the readers may refer to [11].

Asynchronous Multiparty Computation (AMPC): Unlike MPC in synchronous networks, designing AMPC protocols has received very less attention due to their inherent difficulty. It is known that

AMPC under *cryptographic assumptions* [23, 24] is possible iff $n \geq 3t+1$. In information theoretic settings, AMPC with *zero error* (i.e., *perfectly secure* AMPC) is possible iff $n \geq 4t+1$ [7], whereas AMPC with *negligible error probability* is possible iff $n \geq 3t+1$ [9]. The communication complexities *per multiplication* of the best known AMPC protocols in different settings are given in the following table. So, in the table, C_M denotes the number of multiplication gates in arithmetic circuit computing the function f (general MPC/AMPC protocols assume that the function f is to be computed over a finite field $\mathbb{F}$ and the function can be expressed as an arithmetic circuit over $\mathbb{F}$, consisting of input, random, linear, multiplication and output gates). Moreover, IT denotes Information Theoretic security. Furthermore, for cryptographic AMPC, κ is the security parameter, while for information theoretic AMPC (with negligible error probability), κ is the error parameter.

Reference	Type of Security	Resilience	Communication Complexity in bits
[23]	Cryptographic	$t < n/3$ (optimal)	$\mathcal{O}(C_M n^3 \kappa)$
[24]	Cryptographic	$t < n/3$ (optimal)	$\mathcal{O}(C_M n^2 \kappa)$
[31]	IT (no error)	$t < n/4$ (optimal)	$\Omega(C_M n^5 \log(\|\mathbb{F}\|))$
[5]	IT (no error)	$t < n/4$ (optimal)	$\mathcal{O}(C_M n^3 \log(\|\mathbb{F}\|))$
[9]	IT (negligible error)	$t < n/3$ (optimal)	private - $\Omega(C_M n^{11} \kappa^4)$ A-Cast - $\Omega(C_M n^{11} \kappa^2 \log(n))$
[28]	IT (negligible error)	$t < n/4$ (non-optimal)	$\mathcal{O}(C_M n^4 \kappa)$

From the table, we find that the only known information theoretic secure AMPC with $n = 3t+1$, involves high communication complexity [1]. Recently in [17], the authors have designed communication efficient MPC protocols over networks that exhibit partial asynchrony (where the network is synchronous up to certain point and becomes completely asynchronous after that) and hence we do not compare it with our MPC protocol, which is designed in completely asynchronous settings.

Our Contribution: We design an efficient information theoretic secure AMPC protocol with $n = 3t+1$ having a negligible error probability of $2^{-\mathcal{O}(\kappa)}$. Our AMPC protocol (privately) communicates $\mathcal{O}(n^5\kappa)$ bits per multiplication, thus significantly improving the communication complexity of the only known AMPC protocol of [9] in the same settings. As a tool for our AMPC protocol, we introduce a new asynchronous primitive called *Asynchronous Complete Verifiable Secret Sharing* (ACVSS). For designing our ACVSS, we present a new *asynchronous verifiable secret sharing* (AVSS) protocol, which is the most communication-efficient AVSS protocol so far with $n = 3t+1$. We believe that our ACVSS and the tools that we use for designing ACVSS can be used in many other applications for improving communication complexity and hence are of independent interest.

A Brief Discussion on the Approaches used in the AMPC of [9] and Current Article:

1. The AMPC protocol of [9] proceeds in two phases: input phase and computation phase. In the input phase every party shares (or commits) his input x_i. All the parties then decide on a set of $n-t$ parties who have done proper sharing of their input. Now for sharing/committing inputs (secrets), a natural choice is to use AVSS protocol which can be treated as a form of *commitment*, where the commitment is held in a distributed fashion among the parties. Before [9], the only known AVSS scheme with $n = 3t+1$ was due to [12]. But it is shown in [9] that the use of the AVSS protocol of [12] for committing inputs (secrets), does not allow to compute the circuit robustly in a straight-forward way. This is because *for robust computation of the circuit, it is to be ensured that at the end of AVSS sharing phase, every honest party should have access to proper share of the secret.* Unfortunately the AVSS of [12] does not guarantee the above property, which we may refer as *ultimate* property. This very reason motivated Ben-Or et. al [9] to introduce a new asynchronous primitive called *Ultimate Secret Sharing* (USS) which not only ensures that every honest party has access to the proper share of secret, but also offers all the properties of AVSS. Thus [9] presents an USS scheme with $n = 3t+1$ using the AVSS protocol of Canetti and Rabin [12] as a building block. Essentially, in the USS protocol of [9], every share of the secret is committed using AVSS of [12] which ensures that each honest party P_i can have an access to the i^{th} share of secret by means of private reconstruction of AVSS. A secret s that is shared using USS is called *ultimately shared.* Now in the input phase of AMPC in [9], parties *ultimately share* their inputs. Then in the computation phase, for every gate (except output gate), *ultimate sharing* of the output is computed from the *ultimate sharing* of the inputs, following approach of [8, 30].

[1] The communication complexity analysis of the AMPC protocol of [9] was not done earlier and for the sake of completeness, we carry out the same in **APPENDIX A**.

2. Our AMPC protocol is presented in preprocessing model of [2] and proceeds in three phases: preparation phase, input phase and computation phase. We call a triple (a, b, c) as a random multiplication triple if a, b are random and $c = ab$. In the preparation phase, sharing of $c_M + c_R$ random multiplication triples are generated, where c_M and c_R are the number of multiplication and random gates in the circuit. Each multiplication and random gate of the circuit is associated with a multiplication triple. In the input phase the parties share (commit) their inputs and agree on a core set of $n - t$ parties who correctly shared their inputs. In the computation phase, the actual circuit will be computed gate by gate, based on the inputs of the parties in core set. Due to the linearity of the used secret-sharing, the linear gates can be computed locally. Each multiplication gate will be evaluated using the circuit randomization technique of Beaver [2] with the help of a multiplication triple.

For committing/sharing secrets, we use a new asynchronous primitive called ACVSS in our AMPC protocol. There is a slight *definitional difference* between the USS of [9] and our ACVSS, though both of them offer all the properties of AVSS. While USS of [9] ensures that every honest party has *access* to proper share of secret (*but may not hold the share directly*), our ACVSS ensures that *every honest party holds proper share* of secret. We may refer to the above property of ACVSS as *completeness* property. The advantages of ACVSS over USS are: (a) it makes the computation of the gates very simple, (b) reconstruction phase of ACVSS is very simple, efficient and can be achieved using *on-line error correction* of [11]. Apart from these advantages, our ACVSS is strikingly better than USS of [9] in terms of communication complexity. While our ACVSS share protocol communicates $\mathcal{O}((\ell n^4 + n^5\kappa)\kappa))$ bits and $\mathsf{A\text{-}casts}$ $\mathcal{O}(n^4 \log(n))$ bits to share ℓ *secrets concurrently*, the USS share protocol of [9] communicates $\Omega(n^{10}\kappa^4)$ bits and $\mathsf{A\text{-}casts}$ $\Omega(n^{10}\kappa^2 \log(n))$ bits to share *only one secret.* To design our ACVSS, we design a new AVSS scheme with $n = 3t + 1$.

<u>**Comparison of Our AVSS Protocol with Existing AVSS Protocols:**</u> Our AVSS is to be compared with the AVSS protocols of [12] and [27] with $n = 3t + 1$. Notice that all the three AVSS protocols lack both *ultimate* and *completeness* property and hence they cannot be *directly used* in AMPC. While AVSS of [12] was extended to USS [9], we extend our AVSS to ACVSS before applying for AMPC. While the AVSS protocol of [12] shares a *single secret* and requires private communication of $\mathcal{O}(n^9\kappa^4)$ bits and $\mathsf{A\text{-}Cast}$ $\mathcal{O}(n^9\kappa^2 \log(n))$ bits, our AVSS protocol shares ℓ *secrets concurrently* and requires private communication of $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits and $\mathsf{A\text{-}cast}$ $\mathcal{O}(n^3 \log n)$ bits. Our AVSS protocol is superior to the AVSS protocol of [27] both property-wise and communication-wise (though both of them share ℓ secrets concurrently). While the AVSS protocol of [27] does not bar a *corrupted* D to commit $NULL \notin \mathbb{F}$ during sharing phase, our AVSS protocol stops D to do so and allows D to commit ℓ secrets from $\mathbb{F}$ *only.* Though it looks like a slightly stronger property, it calls for a completely new design approach. Moreover, while it is very easy to design an ACVSS protocol using our AVSS protocol as a building block, it may not be the case for the AVSS protocol of [27]. Now the AVSS of [27] requires private communication of $\mathcal{O}((\ell n^3 + n^4)\kappa)$ bits and $\mathsf{A\text{-}casts}$ $\mathcal{O}((\ell n^3 + n^4)\kappa)$ bits. Comparing this with our AVSS, we find that our AVSS calls for $\mathsf{A\text{-}casts}$ that is independent of ℓ while keeping private communication almost same. The AVSS of [27] is used to design efficient ABA protocol, where the maximum value of ℓ can be n. But in AMPC, the size of ℓ depends upon the circuit size and can be arbitrarily large. So in the context of AMPC, our AVSS, whose $\mathsf{A\text{-}Cast}$ is independent of ℓ, fits better than AVSS of [27].

The improvements in our AVSS protocol are due to the key factors like: (a) new design approach, (b) use of efficient building blocks, (c) harnessing the advantages offered by dealing with multiple secrets *concurrently.* To emphasize on the last factor, we remark that our protocols dealing with ℓ secrets *concurrently* are far better than ℓ repeated applications of protocols dealing with single secret.

2 Preliminaries

<u>**Model:**</u> We follow the network model of [9], where there is a set of n parties 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 $t < \frac{n}{3}$ parties in Byzantine fashion and can make the corrupted parties to deviate from the protocol arbitrarily. Moreover, $\mathcal{A}_t$ is given the power to schedule messages over each channel. But he will have no access to the messages sent by honest parties. The function to be computed is specified by an arithmetic circuit over a finite field $\mathbb{F}$. The circuit consists 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 involves a 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 $\mathbb{F}$, where $\mathbb{F} = GF(2^{\kappa})$. Thus each field element can be represented by κ bits. Moreover, without loss of generality, we assume that $n = \text{poly}(\kappa)$.

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

A-Cast[12]: It is an asynchronous broadcast primitive, introduced and implemented in [10] with $n = 3t+1$. From [10], A-Cast of b bits incurs a private communication of $\mathcal{O}(n^2 b)$ bits. Let Π be an asynchronous protocol initiated by a special party (called sender), having input m (the message to be broadcast). We say that Π is a t-resilient A-cast protocol if following holds, for every $\mathcal{A}_t$ and input m:

• **Termination**: 1. If the sender is honest and all the honest parties participate in the protocol, then each honest party will eventually complete the protocol.

2. Irrespective of the behavior of the sender, if any honest party completes the protocol then each honest party will eventually complete the protocol.

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

In the sequel, we use following convention: *we say that P_j listens m from the A-Cast of P_i, indicating that P_j has completed the execution of P_i's A-Cast with output m.*

ACS[5]: It is a primitive presented in [7, 9] to determine a set of $n-t$ parties that correctly shared their values. More specifically, each party starts ACS protocol with an accumulative set of parties who from his view point correctly shared their values. The output of the protocol is a set of $n-t$ parties, who correctly shared their values. The communication cost of an ACS is $\mathcal{O}(poly(n,\kappa))$ bits.

Asynchronous Weak Secret Sharing (AWSS) [12]: Let (Sh, Rec) be a pair of protocols in which a dealer $D \in \mathcal{P}$ shares a secret $S = (s^1 \dots s^{\ell})$ containing $\ell \geq 1$ field elements. We say that (Sh, Rec) is a t-resilient AWSS scheme for n parties if the following hold for every possible $\mathcal{A}_t$.

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

1. If D is honest then each party will eventually terminate protocol Sh.
2. If some honest party has terminated protocol Sh, then irrespective of the behavior of D, each honest party will eventually terminate Sh.
3. If an honest party has terminated Sh and all the honest parties invoke protocol Rec, then each honest party will eventually terminate Rec.

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

1. If D is *honest* then each honest party upon completing protocol Rec, outputs the shared secret S.
2. If D is *faulty* and some honest party has terminated Sh, then there exists a unique $S' = (s'^1 \dots s'^{\ell}) \in (\mathbb{F}^{\ell} \cup NULL)$, such that each honest party upon completing Rec, will output either S' or $NULL$.

• **Secrecy**: If D is honest and no honest party has begun executing protocol Rec, then the corrupted parties have no information about S.

AVSS [12]: The **Termination** and **Secrecy** conditions for AVSS is same as in AWSS. The only difference is in the **Correctness** 2 property:

Correctness 2: If D is *faulty* and some honest party has terminated Sh, then there exists a unique $S' = (s'^1 \dots s'^{\ell}) \in \mathbb{F}^{\ell}$, such that with probability at least $1 - 2^{-\mathcal{O}(\kappa)}$, each honest party upon completing Rec, will output *only* S'.

ACVSS: The **termination, correctness** and **secrecy** property of ACVSS are same as in AVSS. In addition, ACVSS requires the following *completeness* property: *every honest party holds proper share of secret at the end of Sh.*

3 Asynchronous Complete Verifiable Secret Sharing (ACVSS)

We now present a novel ACVSS protocol that allows a dealer $D \in \mathcal{P}$ to generate complete sharing of ℓ secrets. To design our ACVSS, we first design a sequence of asynchronous primitives, namely Information Checking Protocol (ICP), AWSS and AVSS.

3.1 Information Checking Protocol and IC Signature

The Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of computationally unbounded corrupted parties. The notion of ICP was first introduced by Rabin et.al [30, 29]. The ICP of Rabin et. al. was also used as a tool by Canetti et. al. [12] for designing asynchronous Byzantine Agreement (ABA) with optimal resilience (i.e $n = 3t + 1$). As described in [30, 29, 12, 15], an ICP is executed among three parties: a dealer D, an intermediary INT and a verifier R. The dealer D hands a secret value s to INT. At a later stage, INT is required to hand over s to R and convince R that s is indeed the value which INT received from D.

The basic definition of ICP involves only a *single* verifier R and deals with *only one* secret s [29, 15, 12]. We extend this notion to *multiple* verifiers, where all the n parties in $\mathcal{P}$ act as verifiers. Thus our ICP is executed among three entities: the dealer $D \in \mathcal{P}$, an intermediary $INT \in \mathcal{P}$ and entire set $\mathcal{P}$ acting as verifiers. This will be later helpful in using ICP as a tool in AWSS protocol. Moreover, when appropriate, we run our ICP to *concurrently* work on *multiple* secrets, denoted by S, which contains $\ell \geq 1$ secret values. So, instead of repeating multiple instances of ICP dealing with single secret, we can run a single instance of ICP dealing with multiple secrets *concurrently*, leading to significant reduction in communication complexity. We use our ICP in our AWSS schemes, where it is required to execute instances of ICP dealing with multiple secrets concurrently. Note that, as opposed to the case of a single verifier, when multiple verifiers *simultaneously* participate in ICP, we need to distinguish between synchronity and asynchronity of the network. Our ICP is executed in asynchronous settings and is denoted by $\mathsf{A\text{-}ICP}(D, INT, P, S)$.

Recently, in [27] an A-ICP protocol is proposed, dealing with *multiple secrets* and *multiple verifiers* in asynchronous settings. The A-ICP is further used in [27] to design an efficient ABA protocol. However, the A-ICP protocol of [27] incurs a private communication of $\mathcal{O}((\ell + n)\kappa)$ bits and A-Cast of $\mathcal{O}((\ell + n)\kappa)$ bits. On the other hand, our A-ICP incurs *only* private communication of $\mathcal{O}((\ell + n\kappa)\kappa)$ bits (and *no* A-cast). As in [30, 29, 12], our A-ICP is also structured in three phases:

1. Generation Phase: is initiated D. Here D hands over the secret S, along with *authentication information* to *intermediary* INT and some *verification information* to individual *verifiers* in $\mathcal{P}$.

2. Verification Phase: is carried out by INT and the set of verifiers $\mathcal{P}$. Here INT decides whether to continue or abort the protocol depending upon the prediction whether in **Revelation Phase**, the secret S held by INT will be (eventually) accepted/will be considered as valid by the honest verifier(s) in $\mathcal{P}$. INT achieves this by setting a boolean variable $\mathsf{Ver} = 0/1$, where $\mathsf{Ver} = 0$ (resp. 1) implies abortion (resp. continuation) of the protocol. If $\mathsf{Ver} = 1$, then the *authentication information*, along with S, held by INT at the end of **Verification Phase** is called D's IC *signature* on S.

3. Revelation Phase: is carried out by INT and the verifiers in $\mathcal{P}$. **Revelation Phase** can be presented in two flavors: (i) *Public Revelation* of IC signature to all the parties in $\mathcal{P}$; (ii) P_α*-private-revelation* of IC signature: Here the signature is privately revealed to only a specific party P_α. First, INT reveals his secret S along with the *authentication information* to P_α. The verifiers reveal their respective *verification information* to P_α. Then P_α privately performs some checking with the information received from INT and verifiers and finally either accepts INT's secret S or rejects it. We denote the acceptance by verifier P_α, (resp., rejection) by $\mathsf{Reveal}_\alpha = S$ (resp., $NULL$).

Protocol A-ICP satisfies the following properties:

1. If D and INT are honest, then S will be accepted in **Revelation phase** by each honest verifier.

2. If INT is honest and Ver =1, then S held by INT will be accepted in **Revelation phase** by each honest verifier, except with probability $2^{-\mathcal{O}(\kappa)}$.

3. If D is honest, then during **Revelation phase**, with probability at least $1 - 2^{-\mathcal{O}(\kappa)}$, every $S' \neq S$ produced by a corrupted INT will be not be accepted by an honest verifier.

4. If D and INT are honest and INT has not started **Revelation phase**, then S is information theoretically secure.

Notice that unlike other asynchronous primitives (e.g. AWSS, AVSS, ACVSS), A-ICP need not have to satisfy any termination property. The reason is that A-ICP will never be executed as a stand alone application. Rather, A-ICP will act as a tool to design AWSS, which has its own termination properties. This is in line with [12], where ICP is defined without termination property and is used as a tool in

AWSS/AVSS protocol. We now present our A-ICP, which allows D to deal with secret S containing $\ell \geq 1$ field elements, where $n = 3t + 1$. In our implementation of AWSS, AVSS and ACVSS protocols, we do not require public revelation of IC signature. So we give the details of private revelation of IC signature only (though we claim to have an implementation of public revelation of IC signature). For the proof of the properties of our A-ICP, see **APPENDIX B**.

Protocol A-ICP(D,INT,$\mathcal{P}$,S)

Generation Phase: Gen(D$, INT, \mathcal{P}, S$**)**

1. The dealer D, having secret $S = (s^1, \ldots, s^\ell)$, selects a random $\ell + t\kappa$ degree polynomial $f(x)$ whose lower order ℓ coefficients are the secrets in S. D also picks $n\kappa$ random non-zero elements from $\mathbb{F}$, denoted by $\alpha_1^i, \ldots, \alpha_\kappa^i$, for $1 \leq i \leq n$.
2. D sends $f(x)$ to INT and the verification tags $z_1^i = (\alpha_1^i, a_1^i), \ldots, z_\kappa^i = (\alpha_\kappa^i, a_\kappa^i)$ to P_i, where $a_j^i = f(\alpha_j^i)$.

Verification Phase: Ver(D$, INT, \mathcal{P}, S$**)**

1. Every verifier P_i randomly partitions the index set $\{1, \ldots, \kappa\}$ into two sets I^i and $\overline{I^i}$ of (almost) equal size and sends I^i and z_j^i for all $j \in I^i$ to INT.
2. For every verifier P_i from which INT has received values, INT checks whether for *every* $j \in I^i$, $f(\alpha_j^i) \overset{?}{=} a_j^i$.
3. (a) If for at least $2t + 1$ verifiers, the above condition is satisfied, then INT sets Ver $= 1$. If Ver $= 1$, then the information held by INT is called D's IC signature on S.

   (b) If for at least $t + 1$ verifiers, the above condition is not satisfied, then INT sets Ver $= 0$.

Revelation Phase: Reveal-Private($D, INT, \mathcal{P}, S, P_\alpha$**)**: P_α*-private-revelation* of IC signature

1. To party P_α, INT sends $f(x)$.
2. To party P_α, every verifier P_i sends the index set $\overline{I^i}$ and all z_j^i such that $j \in \overline{I^i}$.
3. Upon receiving the values from verifier P_i, P_α checks whether for *any* $j \in \overline{I^i}$, $f(\alpha_j^i) \overset{?}{=} a_j^i$.

   (a) If for at least $t + 1$ verifiers the condition is satisfied, then P_α sets Reveal$_\alpha = S$, where S is lower order ℓ coefficients of $f(x)$. In this case, we say that INT is 'successful' in producing IC *signature* to P_α.

   (b) If for at least $2t+1$ verifiers the above condition is not satisfied, then P_α sets Reveal$_\alpha = NULL$. In this case, we say that INT 'fails' in producing IC *signature* to P_α.

Lemma 1 *Protocol Gen, Ver and Reveal-Private privately communicate $\mathcal{O}((\ell + n\kappa)\kappa)$ bits each.*

3.2 Asynchronous Weak Secret Sharing

We now present an AWSS protocol called AWSS with $n = 3t + 1$, which allows D to share a secret S containing $\ell \geq 1$ field elements from $\mathbb{F}$. The protocol is similar to the AWSS protocol AWSS-Multiple-Secret given in [27] (which also deals with multiple secrets), except with the following difference: *AWSS-Multiple-Secret uses the A-ICP protocol of [27] as a black-box. We use our new A-ICP protocol to design our AWSS protocol.* Thus our AWSS protocol requires *significantly less* A-Cast than AWSS-Multiple-Secret of [27]. Due to the similarity of our AWSS protocol with AWSS-Multiple-Secret of [27], we present AWSS in **APPENDIX C**. We request the reader to go through **APPENDIX C**, as we have introduced few new key words/terminologies in our AWSS protocol (not present in AWSS-Multiple-Secret), which will be later used in the description of our new AVSS protocol. For the ease of reference, we provide the communication complexity of protocol AWSS in the following lemma:

Lemma 14: *Protocol AWSS-Share privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^2 \log n)$ bits. Protocol AWSS-Rec-Private privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits.*

3.3 Asynchronous Verifiable Secret Sharing

We now present a novel AVSS protocol called AVSS with $n = 3t + 1$, which allows D to share a secret S containing $\ell \geq 1$ field elements from $\mathbb{F}$. We first design an AVSS protocol, called AVSS-Single that deals with a single secret s. Later we present AVSS which is a simple extension of AVSS-Single.

Broad-level Discussion of AVSS-Single: D selects a random degree-t polynomial $g(x)$ such that $g(0) = s$. Now party P_i has to hold $g(i)$, the i^{th} share of s, in a way that ensures robust reconstruction of s later in reconstruction phase. In order to do so, D is first asked to commit n points on $g(x)$, namely

$g(1), \ldots, g(n)$ using n distinct invocations of our AWSS protocol. Here is the problem that may arise when D is *corrupted*:

D may commit n values which do not lie on a t degree polynomial. In order to deal with this, in our protocol D is asked to do the following: D selects random t-degree polynomials $f^1(x), \ldots, f^{t+1}(x)$, such that for $i = 1, \ldots, t+1$, $f^i(0) = g(i)$. Using these polynomials, D constructs additional t-degree polynomials $f^{t+2}(x), \ldots, f^n(x)$, such that for $j = 0, \ldots, n$, the points $f^1(j), \ldots, f^n(j)$ lie on a unique t-degree polynomial. Note that this ensures for $i = 1, \ldots, n$, $f^i(0) = g(i)$ holds. Thus essentially, D constructs an $n \times n$ matrix M, where each row and each column of M are t-consistent. Moreover given *any* $(t+1)$ rows and columns of M, M is *completely* and *uniquely* fixed and so is the secret s. So given $g(x)$, the aforementioned technique of generating $f^1(x), \ldots, f^n(x)$ can be captured as a function: $(f^1(x), \ldots, f^n(x)) = \mathsf{Generate}(g(x))$. Now D commits M by committing its columns i.e $f^1(x), \ldots, f^n(x)$ using n distinct invocations of AWSS protocol. Consequently, each P_i receives i^{th} point on the polynomials, namely $f^1(i), \ldots, f^n(i)$ which is the i^{th} row of M. Now we let P_i participate further in the AWSS instances only if $(f^1(i), \ldots, f^n(i))$ is t-consistent. So if the AWSS instances of D terminate with a common $WCORE$, then all the honest parties (at least $t+1$) in the $WCORE$ ensure that they hold a potential t-consistent row of M. Now to check whether D has committed at least $t+1$ columns of M properly, the reconstruction of i^{th} invocation of D's AWSS is executed to enable P_i-weak-private-reconstruction of $f^i(x)$ (for the meaning of P_i-weak-private-reconstruction, see protocol AWSS-Single in **APPENDIX C**). In case P_i reconstructs $f^i(x)$ then P_i's task is to let the other parties know about it. Once the parties know that enough number of *honest* parties (at least $t+1$) have reconstructed their respective $f^i(x)$'s, they will be convinced that D has indeed committed at least $(t+1)$ proper rows and columns of M, which in turn fix M and the secret. So sharing phase of AVSS can be terminated.

The last concern now is how do we ensure the robust reconstruction of s in the reconstruction phase of AVSS. The reconstruction of the AWSS instances of D does not assure reconstruction of committed secret s, because for a *corrupted* D, the reconstruction of the D's AWSS instances may output $NULL$. So we do the following trick in the sharing phase of AVSS: When P_i reconstructs $f^i(x)$ from P_i-weak-private-reconstruction, P_i acts as a dealer to initiate a new instance of AWSS to *re-commit* $f^i(x)$. At this point, if P_i attempts to re-commit any polynomial, other than $f^i(x)$, then his re-commitment will not be terminated. Moreover, the sharing phase of AVSS terminates only when $2t+1$ parties have successfully re-committed their polynomials in their respective AWSS instances. This re-commitment by $2t+1$ parties serves dual purpose: **(a)** Every P_i among these $2t+1$ parties informs others that he has reconstructed $f^i(x)$, **(b)** it allows robust reconstruction of s later in reconstruction phase. In fact, during the reconstruction phase of the AVSS, an honest P_i's instance of AWSS will always reconstruct $f^i(x)$, while a corrupted P_i's instance will output either $f^i(x)$ or $NULL$. Since during the sharing phase of AVSS, the sharing phase of AWSS instances for $2t+1$ parties had terminated, the reconstruction phase of at least $t+1$ among them will be successful. This will enable the robust reconstruction of s.

Micro-level Discussion of AVSS-Single: Since there are many subtle steps in the protocol, we try to facilitate the readers to build up their intuition as they move along the protocol steps. We motivate each involved step in our protocol along with the explanation on what it achieves. We request the reader to read this discussion along with protocol steps, to get an easy understanding of the protocol. In step [D's COMMITMENT], D commits $f^1(x), \ldots, f^n(x)$ by executing n invocations of AWSS-Single-Share. This can be viewed as D is asked to commit n columns of the matrix M. Now in sub-step [CODE FOR P_i], we want P_i to synchronize all the n invocations of D's AWSS-Single-Share in order to end up with a common set $WCORE^D$ for all these n invocations (instead of n different $WCORE$s). A common $WCORE^D$ not only makes the life simple, but also ensures that all the n invocations terminate simultaneously. Also in sub-step [CODE FOR P_i], P_i verifies whether $((1, f^1(i)), \ldots, (n, f^n(i)))$ lie on a t degree polynomial, where P_i obtains $f^j(i)$ in the j^{th} invocation of D's AWSS-Single-Share. This is to assure P_i that $(f^1(i), \ldots, f^n(i))$ may be the potential i^{th} row of matrix M that D is supposed to commit. So if all the n invocations of D's AWSS-Single-Share terminate with $WCORE^D$, then everyone knows that at least $t+1$ honest P_i's in $WCORE^D$ are holding $t+1$ potential rows of M. Now to be sure that D has indeed committed M, everyone should know that at least $t+1$ potential columns of M are also t-consistent. This is accomplished by several moves in our protocol.

The first move is that the potential j^{th} column of M, $f^j(x)$ which is committed by D, is P_j-weak-private-reconstructed by executing the step [P_j-WEAK-PRIVATE-RECONSTRUCTION OF $f^j(x)$ FOR $j = 1, \ldots, n$:].

When P_j recovers $f^j(x)$, he further acts as a dealer himself and initiates an instance of AWSS-Single-Share to re-commit $f^j(x)$. This recommitment serves dual purpose as described in our **Broad-level** discussion.

Protocol AVSS-Single$(D, \mathcal{P}, s)$

AVSS-Single-Share$(D, \mathcal{P}, s)$

D's Commitment:

i. Code for D:

1. Select a degree-t random polynomial $g(x)$ with $g(0) = s$. Compute $(f^1(x), \ldots, f^n(x)) = \mathsf{Generate}(g(x))$.
2. For $i = 1, \ldots, n$, initiate AWSS-Single-Share$(D, \mathcal{P}, f^i(x))$ for sharing $f^i(x)$ (see Note 5 in **APPENDIX C** for the interpretation of sharing polynomial using AWSS-Single-Share).

ii. Code for P_i:

1. For $j = 1, \ldots, n$, synchronize each step of [Verification: Code for P_i] in AWSS-Single-Share$(D, \mathcal{P}, f^j(x))$. That is, execute step k of [Verification: Code for P_i] in all these n invocations and if the requirements (if any) of k^{th} step are met for *all* the n invocations, then proceed to step $k+1$ for all n invocations.
2. As an additional requirement, after the completion of step 1 of [Verification: Code for P_i] for all the n invocations, check whether $(1, f^1(i)), (2, f^2(i)), \ldots, (n, f^n(i))$ lies on a degree-t polynomial. If yes proceed further to participate and execute step 2 of [Verification: Code for P_i] for all the n invocations.
3. If step 4 of [Verification: Code for P_i] is executed successfully for all n invocations of D's AWSS-Single-Share, then A-cast a single $\mathtt{OK}(P_i, P_j)$ for all n invocations instead of n $\mathtt{OK}(P_i, P_j)$'s for each of them.

iii. WCore Construction: Code for D– Construct only one copy of $WCORE$, called $WCORE^D$, common for all the n invocations of AWSS initiated by D, by following the steps for core construction in AWSS-Single-Share.

iv. WCore verification & Agreement: Code for P_i– Similar to the description in AWSS-Single-Share.

P_j-Weak-Private-Reconstruction of $f^j(x)$ for $j = 1, \ldots, n$: (Code for P_i:)

1. After executing step iv. of D's Commitment, participate in AWSS-Single-Rec-Private$(D, \mathcal{P}, f^j(x), P_j)$ (see Note 5 in **APPENDIX C**)), for $j = 1, \ldots, n$, to enable P_j privately reconstruct $f^j(x)$. At the completion of AWSS-Single-Rec-Private$(D, \mathcal{P}, f^i(x), P_i)$, obtain either t-degree polynomial $f^i(x)$ or $NULL$.

Re-Commitment by Individual Party

i. Code for P_i:

1. If $f^i(x)$ is reconstructed in AWSS-Single-Rec-Private$(D, \mathcal{P}, f^i(x), P_i)$, then acting as a dealer, initiate AWSS-Single-Share$(P_i, \mathcal{P}, f^i(x))$ to recommit $f^i(x)$.
2. Construct $WCORE^D_i = WCORE^D$, where $WCORE^D_i$ denotes the local copy of $WCORE^D$ for party P_i. Keep updating $WCORE^D_i$ locally with new parties, where a new P_j will be added to $WCORE^D_i$ if $2t+1$ $\mathtt{OK}(., P_j)$s are A-casted in the AWSS instances initiated by D, i.e $|OKSetP_j| \geq 2t+1$.
3. If P_j is a new entrant in $WCORE^D_i$, then participate (as INT or/and as verifier) in IC-Reconstructing $f^k(j)$ towards P_k for $k = 1, \ldots, n$ (for the meaning of IC-Reconstruction, see protocol AWSS-Single).
4. Participate in AWSS-Single-Share$(P_j, \mathcal{P}, f^j(x))$ if (A) $P_i \in WCORE^D_i$ and (B) the $f^j(i)$ received from D in AWSS-Single-Share$(D, \mathcal{P}, f^j(x))$ is same as $f^j(i)$ now received from P_j in AWSS-Single-Share$(P_j, \mathcal{P}, f^j(x))$.
5. $WCORE^{P_i}$ Construction for AWSS-Single-Share$(P_i, \mathcal{P}, f^i(x))$: Here P_i as a dealer, constructs $WCORE$ for AWSS-Single-Share$(P_i, \mathcal{P}, f^i(x))$ in a different way (this is not same as in protocol AWSS-Single-Share) to prove that *he has indeed re-committed* $f^i(x)$.
   (a) Construct a set $ProbCORE^{P_i}$ ($= \emptyset$ initially). Include P_j in $ProbCORE^{P_i}$ and A-cast $\mathtt{Message}(P_j, ProbCORE^{P_i})$ if (A) At least $2t+1$ A-casts of the form $\mathtt{OK}(., P_j)$ are heard in the instance AWSS-Single-Share$(P_i, \mathcal{P}, f^i(x))$, (B) $P_j \in WCORE^D_i$, and (C) P_j's point on $f^i(x)$ (namely $f^i(j)$), which is IC-Reconstructed towards P_i is consistent with re-committed polynomial $f^i(x)$.
   (b) On listening $\mathtt{Message}(P_j, ProbCORE^{P_k})$, consider it as *valid* if there are at least $2t+1$ A-casts of the form $\mathtt{OK}(., P_j)$ in the instance AWSS-Single-Share$(P_k, \mathcal{P}, f^k(x))$.
   (c) Construct $WCORE^{P_i}$. Add P_j in $WCORE^{P_i}$ if (A) $P_j \in ProbCORE^{P_i}$ and (B) for at least $2t+1$ P_k's, *valid* $\mathtt{Message}(P_j, ProbCORE^{P_k})$ are listened. A-cast $WCORE^{P_i}$ when $|WCORE^{P_i}| = 2t+1$.

ii. Final CORE Construction: Code for D

1. Create a list $VCORE$. Include P_i in $VCORE$ if *a valid* $WCORE^{P_i}$ is listened from P_i. Here $WCORE^{P_i}$ is *valid*, if P_i has indeed constructed it following the steps in 5 (a(A), c(B)). /* After listening $WCORE^{P_i}$, any party can verify whether P_i has indeed constructed $WCORE^{P_i}$ using steps in 5 (a(A), c(B)). */
2. A-cast $VCORE$ and $WCORE^{P_i}$'s for each P_i in $VCORE$, when $|VCORE| = 2t+1$.

iii. Final CORE Verification & Agreement on CORE: Code for P_i

1. Terminate after listening $VCORE$ and $WCORE^{P_j}$'s from D's A-Cast, where $|VCORE| = 2t+1$, such that for each P_j in $VCORE$, $WCORE^{P_j}$ is *valid.*

Thus when the (honest) parties see that enough number of parties (at least $2t+1$) have recommitted their reconstructed polynomial, then they are convinced that D has indeed committed M. The recommitment part is achieved in step [RE-COMMITMENT BY INDIVIDUAL PARTY]. But now it is to be ensured each P_i indeed re-commits $f^i(x)$, as a corrupted P_i may try to recommit $\overline{f^i(x)} \neq f^i(x)$. When D is honest, then it is very easy to make a corrupted P_i recommit $f^i(x)$. This is ensured in step 4. of sub-step [CODE FOR P_i]. Steps 2 and 3 in sub-step [CODE FOR P_i] are required to enable the parties who were outside of $WCORE^D$ but hold correct values on $f^k(x)$'s in D's AWSS instances, to participate in the re-commitment of $f^k(x)$'s. This is crucial, as even for an honest D, an honest P_i's re-commitment on $f^i(x)$ may not be completed in the absence of steps 2 and 3. This is because for an honest D, $WCORE^D$ may contain only $t+1$ honest parties. So t potentially corrupted parties from $WCORE^D$ may not purposefully participate in the re-commitment of $f^i(x)$'s by honest P_i's.

Now in the case when D is corrupted but has committed M, it is to be enforced that a corrupted P_i recommits $f^i(x)$. This is ensured in step 5 of sub-step [CODE FOR P_i]. Essentially, a corrupted P_i is forced to send $f^i(j)$ to an *honest* $P_j \in WCORE^{P_i}$ during the execution of his AWSS instance. This is ensured by the following two steps: (i) P_i is allowed to include a party P_j in $WCORE^{P_i}$ only when P_j is in $ProbCORE$ of at least $2t+1$ parties. This implies an *honest* $P_j \in WCORE^{P_i}$ holds correct row of committed M. This is because, out of these $2t+1$ parties at least $t+1$ P_k's are honest and thus their $f^k(x)$'s are valid columns of M. It implies that P_j has correct points on these $t+1$ correct $f^k(x)$'s. This together with the fact that P_j have also checked t-consistency of $(f^1(j), \ldots, f^n(j))$ (since P_j is present in $WCORE_i^D$ of $t+1$ honest parties), implies that P_j has received a valid row of M from D during the AWSS instances of D. (ii) Finally since P_j has participated in P_i's recommitment and is considered to be part of $ProbCORE^{P_i}$, it is clear that P_i has passed on $f^i(j)$ to P_j during execution of his recommitment. The proof of the properties of AVSS-Single is provided in **APPENDIX D**.

REMARK: In protocol AVSS-Single-Share, we assume that when a party P_j A-Cast OK(P_j, P_k) during an AWSS instance, he also A-Cast the identity of the dealer of that AWSS instance.

Protocol **AVSS-Single**$(D, \mathcal{P}, s)$

AVSS-Single-Rec-Private$(D, \mathcal{P}, s, P_\alpha)$: Private reconstruction of s by party P_α:

P_α-WEAK-PRIVATE-RECONSTRUCTION OF $f^j(x)$ FOR EVERY $P_j \in VCORE$: (CODE FOR P_i)

1. Participate in AWSS-Single-Rec-Private$(P_j, \mathcal{P}, f^j(x), P_\alpha)$ for every $P_j \in VCORE$.

LOCAL COMPUTATION: CODE FOR P_α

1. For every $P_j \in VCORE$, obtain either $\overline{f^j(x)}$ or $NULL$ from P_α-weak-private-reconstruction of $f^j(x)$. Add party $P_j \in VCORE$ to REC if non-NULL output is obtained.
2. Wait until $|REC| = t+1$. Construct polynomial $\overline{g(x)}$ passing through the points $(k, \overline{f^k(0)})$ where $P_k \in REC$. Compute $\overline{s} = \overline{g(0)}$ and terminate.

Note 1 *In AVSS-Single, we have explained only private reconstruction of the secret by a specific party P_α, as our implementation of ACVSS (where AVSS is used as a black-box) requires only private reconstruction of secrets. Though we have protocol for public reconstruction, we skip it here.*

Note 2 (Important Remark:) *We may invoke AVSS-Single as AVSS-Single$(D, \mathcal{P}, g(x))$ (instead of AVSS-Single$(D, \mathcal{P}, s)$) and by doing so, we mean that D executes AVSS-Single with a degree-t polynomial $g(x)$ with $g(0) = s$. As a result of this, party P_i in $VCORE$ gets $g(i)$. The polynomial $g(x)$ is not completely random but preserves the secrecy of $s = g(0)$. Similarly, AVSS-Single-Rec-Private$(D, \mathcal{P}, g(x), P_\alpha)$ can be used for reconstructing $g(x)$ for P_α, which we call as P_α-private-reconstruction of $g(x)$.*

As mentioned earlier, protocol AVSS, dealing with multiple secrets concurrently, is a simple extension of AVSS-Single-Secret. We defer the presentation of AVSS along with the proof of its properties in **APPENDIX D**. For ease of reference, the communication complexity of AVSS is given below:

Lemma 18: *Protocol AVSS-Share privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^3 \log n)$ bits. Protocol AVSS-Rec-Private privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits.*

3.4 Asynchronous Complete Verifiable Secret Sharing

We now present a novel ACVSS protocol called ACVSS with $n = 3t+1$, which allows D to share a secret S containing $\ell \geq 1$ field elements from $\mathbb{F}$. For the ease of understanding, we first design ACVSS-Single that deals with a single secret s. Briefly the high-level idea of ACVSS-Single is as follows: D selects a random degree-t polynomial $h(x)$ such that $h(0) = s$. Then D computes $(g^1(x), \ldots, g^n(x)) = \mathsf{Generate}(h(x))$ satisfying $g^i(0) = h(i)$ (for $i = 1, \ldots, n$). Now D commits $g^i(x)$ using an instance of AVSS-Single-Share. When a party P_i receives i^{th} points on all the g polynomials (during the execution of n instances AVSS-Single-Share), namely $g^1(i), \ldots, g^n(i)$, he checks whether they are t-consistent or not. If yes then only P_i further performs communication and computation following the protocol steps for all the n instances of D's AVSS-Single-Share. Now once all the n instances terminate with a common $VCORE$, every party knows that D has committed n polynomials $g^1(x), \ldots, g^n(x)$, each of degree t, such that every honest party $P_i \in VCORE$ has checked that $(g^1(i), \ldots, g^n(i))$ is t-consistent. Since there are at least $t+1$ honest parties in $VCORE$, it is clear that the points $(g^1(0), \ldots, g^n(0))$ also defines a t degree polynomial $h(x)$ whose constant term is the committed secret. Now to achieve *completeness* property of ACVSS, every party $P_i \in \mathcal{P}$ should hold $h(i) = g^i(0)$. This is attained by enabling P_i-private-reconstruction of $g^i(x)$ (i.e by executing AVSS-Rec-Private$(D, \mathcal{P}, g^i(x), P_i)$. Now reconstruction phase of ACVSS becomes very simple and follows from the *On-line error correction* method introduced in [7, 11]. It is very easy to see that protocol ACVSS-Single satisfies all the properties of ACVSS.

Protocol ACVSS-Single$(D, \mathcal{P}, s)$

ACVSS-Single-Share$(D, \mathcal{P}, s)$

D's Commitment:

i. Code for D:

1. Select a degree-t random polynomial $h(x)$ with $h(0) = s$. Compute $(g^1(x), \ldots, g^n(x)) = \mathsf{Generate}(h(x))$.
2. For $i = 1, \ldots, n$, initiate AVSS-Single-Share$(D, \mathcal{P}, g^i(x))$ to commit $g^i(x)$.

ii. Code for P_i:

1. Participate in AVSS-Single-Share$(D, \mathcal{P}, g^j(x))$ for $j = 1, \ldots, n$.
2. Wait to construct $WCORE^{P_i}$ in each of the n instances of AVSS-Single-Share. For $j = 1, \ldots, n$, let $WCORE^{(P_i,j)}$ be the $WCORE^{P_i}$ constructed by P_i in AVSS-Single-Share$(D, \mathcal{P}, g^j(x))$. Wait until $|WCORE^{(P_i,*)}| \geq 2t+1$, where $WCORE^{(P_i,*)} = \cap_{i=1}^{n} WCORE^{(P_i,j)}$.
3. Check whether the points $((1, g^1(i)), \ldots, (n, g^n(i)))$ lie on a t degree polynomial, where $g^j(i)$ is obtained during execution of AVSS-Single-Share$(D, \mathcal{P}, g^j(x))$. If yes then A-cast $WCORE^{(P_i,*)}$ which will be considered as common $WCORE^{P_i}$ for all the instances of AVSS-Single-Share initiated by D.

iii. VCORE Construction: Code for D: Create a single copy of $VCORE$ for all the n instances of AVSS-Single-Share by following the steps for core construction in AVSS-Single-Share and A-cast the same along with $WCORE^{(P_i,*)}$ for each $P_i \in VCORE$.

iv. Verification & Agreement on VCORE: Code for P_i: Similar as in Protocol AVSS-Single-Share.

P_j-private-reconstruction of $g^j(x)$ for $j = 1, \ldots, n$: Code for P_i:

1. For $j = 1, \ldots, n$, participate in AVSS-Single-Rec-Private$(D, \mathcal{P}, g^j(x), P_j)$ for P_j-private-reconstruction of $g^j(x)$, after terminating all the n instances of AVSS-Single-Share.
2. Obtain $g^i(x)$ from AVSS-Single-Rec-Private$(D, \mathcal{P}, g^i(x), P_i)$, compute $h(i) = g^i(0)$, the i^{th} share of s and terminate **ACVSS-Single-Share**.

ACVSS-Single-Rec-Private$(D, \mathcal{P}, s, P_\alpha)$: P_α-private-reconstruction of s:

Code for P_i 1. Send $h(i)$, the i^{th} share of the secret to P_α.

Local Computation: Code for P_α

1. Upon receiving at least $2t+1$ t-consistent shares, $\overline{h(i)}$'s, interpolate a degree-t polynomial $\overline{h(x)}$, compute the secret $\overline{s} = \overline{h(0)}$ and terminate **ACVSS-Single-Rec-Private**. This method of reconstruction is called *On-line error correction* [7, 11].

ACVSS-Single-Rec-Public$(D, \mathcal{P}, S, \mathcal{P})$: Public reconstruction of s: Run ACVSS-Rec-Private$(D, \mathcal{P}, s, P_\alpha)$ for every $P_\alpha \in \mathcal{P}$.

We now extend ACVSS-Single to ACVSS which deals with ℓ secrets from $\mathbb{F}$. The description of ACVSS is given in **APPENDIX E**. The communication complexity of ACVSS is stated in the following:

Theorem 1 *Protocol ACVSS-Share privately communicates $\mathcal{O}((\ell n^4 + n^5\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^4 \log n)$ bits. Protocol ACVSS-Rec-Public privately communicates $\mathcal{O}(\ell n^2 \kappa)$ bits.*

Note 3 *We can invoke ACVSS as ACVSS$(D, \mathcal{P}, (h^1(x), \ldots, h^\ell(x)))$ and by doing so, we mean that D executes ACVSS with ℓ degree-t polynomials $h^1(x), \ldots, h^\ell(x)$ such that for $l = 1, \ldots, \ell$, $h^l(0) = s^l$. The polynomials $h^1(x), \ldots, h^\ell(x)$ are not completely random but preserves the secrecy of their constant terms. As a result of this execution, each party P_i gets the shares $h^1(i), \ldots, h^\ell(i)$.*

Note 4 (Random Number Generation) *Using our ACVSS scheme, we now design a protocol* RNG *that allows the parties to jointly generate a random number: each P_i shares a random non-zero $r_i \in \mathbb{F}$ using* ACVSS-Share*. The parties then run* ACS *to agree on a core set of $2t+1$ parties who did proper sharing of their r_i's. Once core set is determined, each party locally adds their shares of r_i's corresponding to each P_i in the core set and then publicly reconstruct the sum of these r_i's. It is easy to see that the sum value is random.* RNG *privately communicates $\mathcal{O}(n^6\kappa^2)$ bits and* A-casts *$\mathcal{O}(n^5 \log n)$ bits.*

Definition 1 (t-1D-Sharing) *If a secret s is shared using* ACVSS-Share*, then we say that s is t-1D-shared, denoted as $[s]_t$, which means that there exists a t degree polynomial $f(x)$, with $f(0) = s$, such that each (honest) P_i holds the i^{th} share $f(i) = s_i$ of s.*

4 Generating t-2D-Sharing

We now present a protocol that allows a dealer D to generate correct t-2D-sharing of secret(s). The protocol will be required for generating multiplication triples. We first define t-2D-sharing.

Definition 2 (t-2D-sharing [4]) *: A value s is t-2D-shared among the parties in $\mathcal{P}$ if there exists t degree polynomials $f(x), f^1(x), \ldots, f^n(x)$ with $f(0) = s$ and for $i = 1, \ldots, n$, $f^i(0) = f(i)$. Moreover,* **every (honest) party** *$P_i \in \mathcal{P}$ 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 party $P_j \in \mathcal{P}$. We denote t-2D-sharing of s as $[[s]]_t$.*

The t-2D-sharing of s implies that s as well as it's shares are individually t-1D-shared. Now we present a protocol t-2D-Share which allows D to generate t-2D-sharing of $\ell \geq 1$ secrets, namely $s^1, \ldots, s^\ell$. If D is correct, then every honest party will eventually complete t-2D-Share, and if some honest party has completed t-2D-Share, then all the honest parties will eventually complete t-2D-Share. The high level idea of the protocol is as follows: D selects a random value $s^0 \in \mathbb{F}$ and hides each s^i in the constant term of a random t-degree polynomial $q^i(x)$. D then t-1D-shares the secrets $S^0 = (s^0, \ldots, s^\ell)$ as well as their i^{th} shares $S^i = (q^0(i), \ldots, q^\ell(i))$. The parties then jointly employ a verification technique to ensure that D indeed t-1D-shared S^i for $i = 1, \ldots, n$ which defines S^0. A similar verification technique was used in [4]. The secret s^0 is used to ensure the secrecy during the verification process. After verification, the polynomials used for the t-1D-sharing S_i are privately reconstructed for P_i to complete the specification of t-2D-sharing. The protocol is given below:

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

SHARING BY D: CODE FOR D

1. Select $s^0 \in_R \mathbb{F}$ and $\ell + 1$ degree-t random polynomials $q^0(x), \ldots, q^\ell(x)$ such that for $l = 0, \ldots, \ell$, $q^l(0) = s^l$. Let $s^l_i = q^l(i)$ and $S^i = (q^0(i), \ldots, q^\ell(i))$ for $i = 0, \ldots, n$. So $S^0 = (s^0, \ldots, s^\ell)$ and $S^i = (s^0_i, \ldots, s^\ell_i)$.
2. For $l = 0, \ldots, \ell$ and $i = 1, \ldots, n$, select random t-degree polynomials $q^{(l,i)}(x)$, such that $q^{(l,i)}(0) = q^l(i) = s^l_i$. Let $S^{ij} = (q^{(0,i)}(j), q^{(1,i)}(j), \ldots, q^{(\ell,i)}(j)) = (s^0_{ij}, s^1_{ij}, \ldots, s^\ell_{ij})$.
3. Invoke ACVSS-Share$(D, \mathcal{P}, (q^0(x), q^1(x), \ldots, q^\ell(x)))$ for generating t-1D-sharing of S^0 where P_j receives the shares S^j. Denote this instance of ACVSS-Share by ACVSS-Share^0.
4. For $i = 1, \ldots, n$, invoke ACVSS-Share$(D, \mathcal{P}, (q^{(0,i)}(x), q^{(1,i)}(x), \ldots, q^{(\ell,i)}(x)))$ for generating t-1D-sharing of S^i where P_j receives the share-shares S^{ij}. Denote this instance of ACVSS-Share by ACVSS-Share^i.

VERIFICATION: CODE FOR P_i

1. Upon completion of ACVSS-Share^j for all $j \in \{0, \ldots, n\}$, participate in protocol RNG to jointly generate a random value $r \in \mathbb{F}$ (see Note 4 in section 3.4).
2. Once r is generated, locally compute $s^*_i = \sum_{l=0}^{\ell} r^l s^l_i$ which is the i^{th} share of $s^* = \sum_{l=0}^{\ell} r^l s^l$. In addition, for $j = 1, \ldots, n$, locally compute $s^*_{ji} = \sum_{l=0}^{\ell} r^l s^l_{ji}$ which is the i^{th} share-share of s^*_j.
3. Participate in ACVSS-Rec-Public$(D, \mathcal{P}, (s^*, s^*_1, \ldots, s^*_n), \mathcal{P})$ to publicly reconstruct $s^*, s^*_1, \ldots, s^*_n$. This results in every party reconstructing $q^*(x)$ and $q^*_1(x), \ldots, q^*_n(x)$ with $q^*(0) = s^*$ and $q^*_i(0) = s^*_i$.
4. Check whether for $i = 1, \ldots, n$, $q^*(i) \overset{?}{=} q^*_i(0) = s^*_i$. If yes then set $\mathsf{Ver}_j = 1$. Else set $\mathsf{Ver}_j = 0$. Here $\mathsf{Ver}_j = 1$ (0) means means that D has (not) done proper t-1D-sharing of S^j for $j = 0, \ldots, n$.

PRIVATE RECONSTRUCTION OF POLYNOMIALS USED FOR SHARING S^j TOWARDS P_j: CODE FOR P_i:

1. If $\mathsf{Ver}_i = 1$, then for $j = 1, \ldots, n$, participate in ACVSS-Rec-Private$(D, \mathcal{P}, S^j, P_j)$ for enabling P_j to privately reconstruct the polynomials $q^{(0,j)}(x), \ldots, q^{(\ell,j)}(x)$ which were used by D to share S^j.
2. Wait to privately reconstruct $q^{(0,i)}(x), \ldots, q^{(\ell,i)}(x)$ from ACVSS-Rec-Private$(D, \mathcal{P}, S^i, P_i)$ and terminate.

For the proof of the properties of t-2D-Share, see **APPENDIX F**.

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

5 Preparation Phase

Here we generate correct t-1D-sharing of $c_M + c_R$ *secret* random *multiplication triples* (a^k, b^k, c^k), such that for $k = 1, \dots, c_M + c_R$, $c^k = a^k b^k$. For this we first generate t-2D-sharing of *secret* random doubles $([[a^k]]_t, [[b^k]]_t)$ for $k = 1, \dots, c_M + c_R$. Given these random doubles, we generate t-1D-sharing of c^k, for $k = 1, \dots, c_M + c_R$, by adapting a technique from [15] which was given for synchronous settings.

5.1 Generating Secret and Random t-2D-Sharing

In section 4, we have presented a protocol called t-2D-Share which allows a $D \in \mathcal{P}$ to generate t-2D-sharing of ℓ secrets. We now present a protocol called Random-t-2D-Share which allows the parties to jointly generate random t-2D-sharing of ℓ secrets, unknown to $\mathcal{A}_t$. Random-t-2D-Share asks individual party to act as dealer and t-2D-Share $\frac{\ell}{n-2t}$ random secrets. Then we run ACS protocol to agree on a core set of $n-t$ parties who have correctly t-2D-shared $\frac{\ell}{n-2t}$ random secrets. Now out of these $n-t$ parties, at least $n-2t$ are honest. Hence the secrets that are t-2D-shared by these $n-2t$ honest parties are truly random and unknown to $\mathcal{A}_t$. So if we consider the $\frac{\ell}{n-2t}$ t-2D-sharing done by only the honest parties in core set, then we will get $\frac{\ell}{n-2t} * (n-2t) = \ell$ random t-2D-sharing. For this, we use *Vandermonde Matrix* [16] and its ability to extract randomness which has been exploited by [16, 5]. Protocol Random-t-2D-Share is given in **APPENDIX G**.

Lemma 2 *Protocol Random-t-2D-Share (eventually) terminates with very high probability for every honest party. It outputs t-2D-sharings of ℓ random secret values, unknown to $\mathcal{A}_t$. The protocol privately communicates $\mathcal{O}((\ell n^5 + n^7\kappa)\kappa)$ bits, A-Cast $\mathcal{O}(n^6\log(n))$ bits and requires one invocation of ACS.*

5.2 Proving $c = ab$

Consider the following problem: let $D \in \mathcal{P}$ has t-1D-shared ℓ pairs of values $(a^1, b^1), \dots, (a^\ell, b^\ell)$. Now D wants to t-1D-share $c^1, \dots, c^\ell$ where $c^l = a^l b^l$ without leaking any *additional* information about a^l, b^l and c^l. We propose a protocol ProveCeqAB to achieve this task in asynchronous settings, following a technique proposed in [15] for synchronous settings. The idea of the protocol for a single pair (a, b) is as follows. D selects a random non-zero $\beta \in \mathbb{F}$ and generates t-1D-sharing of c, β and βb. Then all the parties in $\mathcal{P}$ jointly generate a random value r. Each party locally computes the sharing of $p = ra + \beta$ and then p is publicly reconstructed. Then each party locally computes the sharing of $q = pb - b\beta - rc = (ra+\beta)b - b\beta - rc$ and then q is publicly reconstructed. If $q = 0$, then each party believes that with very high probability, D has indeed t-1D-shared $c = ab$. Moreover, if D is honest then a, b and c will remain information theoretic secure. For the proof of correctness and secrecy, see [15]. If D is correct, then every honest party will eventually complete ProveCeqAB, and if some honest party has completed ProveCeqAB, then all the honest parties will eventually complete ProveCeqAB. The protocol is given in **APPENDIX G**.

Lemma 3 *ProveCeqAB privately communicates $\mathcal{O}((\ell n^4 + n^6\kappa)\kappa)$ bits and A-Casts $\mathcal{O}(n^5\log(n))$ bits. If an honest party terminates, then with high probability, D has t-1D-shared $c^l = a^l b^l$, for $1 \leq l \leq \ell$.*

5.3 Generating Multiplication Triples; The Preparation Phase Main Protocol

We now present protocol PreparationPhase which generates t-1D-sharing of $c_M + c_R$ multiplication triples (a^k, b^k, c^k). We explain the idea considering a single triplet (a, b, c). First, Random-t-2D-Share is invoked to generate t-2D-sharing of (a, b) which results in P_i holding the i^{th} share of a and b, namely a_i and b_i respectively. Now if each P_i locally computes $e_i = a_i b_i$, then this results in $2t$-1D-sharing of c. But we want each (honest) P_i to hold c_i, where $(c_1, \dots, c_n)$ is the t-1D-sharing of c. For this we adapt a technique given in [19] for synchronous settings: Each P_i invokes ProveCeqAB to t-1D-share e_i. Now an instance of ACS will be executed to agree on a core set of $n - t = 2t + 1$ parties whose instances of ProveCeqAB has been terminated. For simplicity let core set contains $P_1, \dots, P_{2t+1}$. Since $e_1, \dots, e_{2t+1}$ are $2t+1$ distinct points on a $2t$ degree polynomial, say $C(x)$ where $C(0) = c$, by Lagrange interpolation formula [14], c can be computed as $c = \sum_{i=1}^{n-t} r_i e_i$ where $r_i = \prod_{j=1, j\neq i}^{2t+1} \frac{x-j}{i-j}$. The vector $(r_1, \dots, r_{2t+1})$ is called recombination vector [14] and is known publicly. Now to get t-1D-sharing of c, P_j locally computes $c_j = \sum_{i=1}^{2t+1} r_i e_{ij}$ where e_{ij} is the j^{th} share of e_i. The protocol is given in **APPENDIX G**.

Lemma 4 *PreparationPhase terminates with very high probability. It privately communicates $\mathcal{O}(((c_M + c_R)n^5 + n^7\kappa)\kappa)$ bits, A-Cast $\mathcal{O}(n^6 \log(n))$ bits and requires three invocations of ACS.*

6 Input Phase

In protocol InputPhase, each P_i acts as a dealer to t-1D-share his input X_i containing c_i values. So $c_I = \sum_{i=1}^{n} c_i$. The asynchrony of the network does not allow the parties to wait for more than $n-t$ ACVSS-Share protocols to be completed. To agree on the parties whose inputs will be taken into consideration for computation (of the circuit), an ACS is run. InputPhase is given in **APPENDIX H**.

Lemma 5 *InputPhase eventually terminates and outputs t-1D-sharing of inputs of the parties in agreed core set C with very high probability. It privately communicates $\mathcal{O}((c_I n^4 + n^6\kappa)\kappa)$ bits, A-Casts $\mathcal{O}(n^5 \log(n))$ bits and requires one invocation of ACS.*

7 Computation Phase

Once the input phase is over, in the computation phase, the circuit is evaluated gate by gate, where all inputs and intermediate values are t-1D-shared among the parties. As soon as a party 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 by applying the linear function to the shares, i.e. for any linear function $c = f(a, b)$, the sharing $[c]_t$ is computed by letting every party P_i to compute $c_i = f(a_i, b_i)$, where a_i, b_i and c_i are the i^{th} shares of a, b and c respectively. 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 then used to compute t-1D-sharing of the product, following the circuit randomization technique of Beaver [2]. Given a preprocessed random multiplication triple, which is already correctly t-1D-shared, *Circuit Randomization* [2] allows to evaluate a multiplication gate at the cost of two public reconstructions. Let $z = xy$, where x, y are the inputs of the multiplication gate. Now z can be expressed as $z = ((x-a)+a)((y-b)+b) = (\alpha+a)(\beta+b)$, where (a, b, c) is a random 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 (a, b, c). The protocol ComputationPhase for computation phase is given in **APPENDIX I**.

Lemma 6 *Protocol ComputationPhase (eventually) terminates with very high probability. Given t-1D-sharing of $c_M + c_R$ secret random triples, it computes the outputs of the circuit correctly and privately, by privately communicating $O(n^2(c_M + c_O)\kappa)$ bits.*

8 The New AMPC Protocol with Optimal Resilience

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.

Theorem 3 *For every coalition of up to $t < n/3$ bad parties, the protocol AMPC securely computes the circuit representing function f and eventually terminates with very high probability for all the honest parties. AMPC privately communicates $\mathcal{O}((c_I n^4 + c_M n^5 + c_R n^5 + c_O n^2 + n^7\kappa)\kappa)$ bits, A-Cast $\mathcal{O}(n^6 \log(n))$ bits and requires 4 invocations to ACS.*

9 Conclusion and Open Problems

In this paper, we have designed an information theoretically secure AMPC protocol with $n = 3t+1$, having negligible error probability of $2^{-\mathcal{O}(\kappa)}$, where κ is the error parameter. Our AMPC protocol significantly improves the communication complexity of the only known AMPC protocol of [9] in the same settings. Here we summarize the key factors that has contributed for the gain in the communication complexity: (a) we introduce a new asynchronous primitive called ACVSS which is first of its kind and is of independent interest, (b) our ACVSS protocol is very efficient in terms of communication complexity and uses a new

AVSS protocol as a building block which is the most communication-efficient AVSS so far with $n = 3t+1$. It would be interesting to see whether it is possible to further reduce the communication complexity of the AMPC protocol with $n = 3t + 1$ by using techniques such as player elimination [21].

References

[1] I. Abraham, D. Dolev, and J. Y. Halpern. An almostsurely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *PODC*, pages 405–414, 2008.

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

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

[4] Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In *Proc. of TCC*, volume 3876 of *LNCS*, pages 305–328. Springer Verlag, 2006.

[5] Z. Beerliová-Trubíniová and M. Hirt. Simple and efficient perfectly-secure asynchronous mpc. In *ASIACRYPT*, volume 4833 of *LNCS*, pages 376–392. Springer Verlag, 2007.

[6] 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.

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

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of 20th ACM STOC*, pages 1–10, 1988.

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

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

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

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

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

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

[15] 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.

[16] 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.

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

[18] 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.

[19] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[20] 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.

[21] 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.

[22] M. Hirt and U. M. Maurer. Robustness for free in unconditional multi-party computation. In *Proc. of CRYPTO 2001*, volume 2139 of *LNCS*, pages 101–118. Springer Verlag, 2001.

[23] M. Hirt, J. B Nielsen, and B. Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In *EUROCRYPT*, volume 3494 of *LNCS*, pages 322–340. Springer Verlag, 2005.

[24] M. Hirt, J. B Nielsen, and B. Przydatek. Asynchronous multi-party computation with quadratic communication. In *ICALP (2)*, volume 5126 of *LNCS*, pages 473–485. Springer Verlag, 2008.

[25] J. Katz, C. Koo, and R. Kumaresan. Improving the round complexity of VSS in point-to-point networks. In *ICALP(2)*, volume 5126 of *LNCS*, pages 499–510. Springer Verlag, 2008.

[26] 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.

[27] A. Patra, A. Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous byzantine agreement with optimal resilience. Cryptology ePrint Archive, Report 2008/424, 2008.

[28] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Trading players for efficiency in unconditional multiparty computation. In *SCN*, volume 2576 of *LNCS*, pages 342–353. Springer Verlag, 2002.

[29] T. Rabin. Robust sharing of secrets when the dealer is honest or cheating. *J. ACM*, 41(6):1089–1109, 1994.

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

[31] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *INDOCRYPT*, volume 1977 of *LNCS*, pages 117–129. Springer Verlag, 2000.

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

APPENDIX A: Communication Complexity Analysis of the AMPC Protocol of [9]

The communication complexity analysis of the AMPC protocol of [9] was not done anywhere before. So we have carried out the same at this juncture. The main tool used in the AMPC protocol of [9] is USS. So we first analyze the communication complexity of the USS protocol of [9]. The USS protocol of [9] is designed using the AVSS scheme of [12]. Recently, in [27], the authors have carried out the communication complexity analysis of the AVSS scheme of [12]. In [27], it is shown that for a single secret, the sharing phase of the AVSS scheme of [12] involves a private communication of $\mathcal{O}(n^9\kappa^4)$ bits and A-Cast of $\mathcal{O}(n^9\kappa^2\log(n))$ bits, while the reconstruction phase involves private communication of $\mathcal{O}(n^6\kappa^3)$ bits and A-cast of $\mathcal{O}(n^6\kappa\log(n))$ bits.

Now the sharing phase of the USS scheme of [9] is as follows: Dealer $D \in \mathcal{P}$, having an input secret s, selects a random t-degree polynomial $f(x)$, such that $f(0) = s$. D then computes $f(i)$, for $1 \leq i \leq n$ and shares each $f(i)$ by using the AVSS share protocol of [12]. Thus this step requires a private communication of $\mathcal{O}(n^{10}\kappa^4)$ bits and A-Cast of $\mathcal{O}(n^{10}\kappa^2\log(n))$ bits. Ignoring other steps in the USS protocol of [9], we conclude that the USS sharing phase incurs a private communication of $\Omega(n^{10}\kappa^4)$ bits and A-Cast of $\Omega(n^{10}\kappa^2\log(n))$ bits.

Now using their USS scheme, the authors of [9] compute the circuit using the approach of [8]. To compute multiplication gates, the parties proceed as follows: suppose u and v are the inputs of a multiplication gate. Moreover, both u and v are already *ultimately shared* by the USS sharing protocol. To compute the ultimate sharing of uv, each party locally computes the product of their shares of u and v and then share the product share using USS protocol. This incurs a communication complexity of $\Omega(n^{11}\kappa^4)$ bits and A-Cast of $\Omega(n^{11}\kappa^2\log(n))$ bits. Ignoring other steps of the multiplication protocol and the computation of other gates (namely, addition and output gates), we conclude that the AMPC protocol of [9] incurs a private communication of $\Omega(n^{11}\kappa^4)$ bits and A-Cast of $\Omega(n^{11}\kappa^2\log(n))$ bits per multiplication gate.

APPENDIX B: Proof of the Properties of Protocol A-ICP

Lemma 7 *If D and INT are honest, then S will be accepted by honest verifier P_α.*

Proof(sketch): For an honest D and honest INT, $\mathsf{Ver} = 1$ at the end of **Verification phase** and honest P_α will eventually output $\mathsf{Reveal}_\alpha = S$ at the end of **Revelation phase**. □

Lemma 8 *If INT is honest and $\mathsf{Ver} = 1$ at the end of* **Verification phase***, then S held by INT will be accepted in* **Revelation phase** *by honest P_α, except with probability $2^{-\mathcal{O}(\kappa)}$.*

Proof: We have to consider the case when D is corrupted. Since INT is honest and $\mathsf{Ver} = 1$ at the end of **Verification phase**, INT has ensured that for at least $2t+1$ verifiers the condition specified in step 2 of **verification phase** has been satisfied. Among these $2t+1$ verifiers, at least $t+1$ are honest, say denoted as H. So $|H| \geq t+1$. To prove the lemma, we prove that corresponding to each verifier in H, the condition stated in step 3 of Reveal-Private will be satisfied with very high probability. We first note that corresponding to a verifier P_i in H, the condition stated in step 3 of Reveal-Private will fail if for all $j \in \overline{I^i}$, $f(\alpha^i_j) \neq a^i_j$. This implies that D (who is corrupted in this case) must have distributed $f(x)$ (to INT) and z^i_j (to P_i) inconsistently for all $j \in \overline{I^i}$ and it so happens that P_i has partitioned $\{1, \ldots, \kappa\}$ into two sets I^i and $\overline{I^i}$ such that $\overline{I_i}$ contains only inconsistent tuples (z^i_j's). Thus corresponding to a verifier P_i in H, the probability that the condition stated in step 3 of Reveal-Private fails is same as the probability of P_i selecting all consistent tuples in I^i (or selecting all inconsistent tuples in $\overline{I^i}$), which is at most $2^{-\mathcal{O}(\kappa)}$. □

Lemma 9 *If D is honest, then during* **Revelation phase***, with probability at least $1 - 2^{-\mathcal{O}(\kappa)}$, every $S' \neq S$ produced by a corrupted INT will be rejected by honest P_α.*

Proof: For a corrupted INT producing $S' \neq S$ in Reveal-Private, t corrupted verifiers may produce verification information such that the condition stated in step 3 of Reveal-Private gets satisfied for all of them. So D's signature on S' will be validated if the condition stated in step 3 of Reveal-Private gets satisfied corresponding to at least one honest verifier (in addition to t corrupted verifiers). So we have to analyze: what is the probability that corresponding to an honest verifier P_i, the condition stated in step 3 of Reveal-Private gets satisfied. The probability of the above event is same as the probability that a corrupted INT can guess a verification tag z^j_i for at least one $j \in \overline{I^i}$ for any $\overline{I^i}$ corresponding to honest parties, which is at most $2^{-\mathcal{O}(\kappa)}$. □

Lemma 10 *If D and INT are honest and INT has not started* **Revelation phase***, then S is information theoretically secure from $\mathcal{A}_t$ controlling t verifiers in $\mathcal{P}$.*

Proof: The adversary will know only $t\kappa$ points on $f(x)$. Since $f(x)$ is a polynomial of degree $\ell + t\kappa$, the secret S will remain information theoretically secure. □

APPENDIX C: AWSS Protocols and Their Properties

Before presenting protocol AWSS, for the ease of understanding, we first design protocol AWSS-Single that deals with a single secret. The protocol is similar to protocol AWSS-Single-Secret of [27], except that we use our new A-ICP protocol instead of the A-ICP protocol of [27]. This results in significant reduction in the bits A-Casted in our protocol. Moreover, AWSS-Single-Secret of [27] is presented with public reconstruction, where at the end of reconstruction phase, the secret is publicly available to all the parties. However, we present our protocol AWSS-Single (as well as AWSS) with private reconstruction, where the secret is available only to a specific party P_α at the end of reconstruction phase. We call the private reconstruction as *P_α-weak-private-reconstruction*. Thus though we have an implementation for the public reconstruction of AWSS-Single, we skip it here as our implementation of AVSS protocol (where AWSS is used as a black-box) requires only private reconstruction of AWSS-Single and AWSS.

We first describe the high level idea used in AWSS-Single which is mostly same as that provided in [27] for AWSS-Single-Secret. The protocol follows the general idea of [8, 15, 20, 18, 25] in synchronous settings for sharing the secret s with a degree-t symmetric bivariate polynomial $F(x, y)$, where each party P_i gets the univariate polynomial $f_i(x) = F(x, i)$. In particular, protocol AWSS-Single is somewhat inspired by the WSS protocol given in [15], for synchronous settings.

In the sharing phase of AWSS-Single, D first commits to n points on $f_i(x)$ (which means committing $f_i(x)$) to P_i by giving his IC signature on these values. Then D, in conjunction with all other parties, perform a sequence of communication and computation. As a result of this, at the end of the sharing phase, every party agrees on a set of $2t+1$ parties, called $WCORE$, such that every party $P_j \in WCORE$ is *confirmed* by $2t+1$ parties which are listed in a set $OKSetP_j$. The protocol ensures that every party $P_k \in OKSetP_j$ provides the confirmation to P_j, only when it possesses proper IC signature of D on $f_k(j)$ (j^{th} point on polynomial $f_k(x)$, that P_k is entitled to receive from D) as well as IC signature of P_j on the point $f_j(k)$ (k^{th} point on polynomial $f_j(x)$, which P_j is entitled to receive from D), such that $f_j(k) = f_k(j)$ holds (which should ideally hold due to the selection and distribution of symmetric bivariate polynomial). In some sense, we may view these checkings as every $P_j \in WCORE$ is attempting to commit his polynomial $f_j(x)$ among the parties in $OKSetP_j$ (by giving *IC Signature* on one point of $f_j(x)$ to each party) and the parties in $OKSetP_j$ are allowing him to do so after verifying that they have got D's IC signature on the same value of $f_j(x)$. We will refer this commitment as P_j's *IC-Commitment* on $f_j(x)$. Notice that for an honest D, the degree-t univariate polynomial $F(x, 0) = f_0(x)$ is used to share s, where party P_i gets the share $f_0(i) = f_i(0) = F(i, 0) = F(0, i)$, the polynomial $f_i(x) = F(x, i)$ and the share-share $f_j(i) = F(i, j) = f_i(j)$ corresponding to every other party P_j.

Achieving the agreement (among the parties) on $WCORE$ and corresponding $OKSet$s is a bit tricky in asynchronous network. Even though the *confirmations* are A-casted by parties, parties may end up with different versions of $WCORE$ and $OKSet$'s while attempting to generate them locally, due to the asynchronous nature of the network. We solve this problem by asking D to construct $WCORE$ and $OKSet$s after listening *confirmations* and ask D to A-cast the same. After listening $WCORE$ and $OKSet$s from the A-cast of D, individual parties ensure the validity of (verifies) these sets by listening the same *confirmations* from the parties in the received $OKSet$s. Once the verification is done, every honest party agree on these sets. A similar approach was used in the protocols of [1].

In the reconstruction phase, the parties in $WCORE$ and corresponding $OKSet$'s are used for reconstructing the secret. Precisely, in the reconstruction phase, P_j's *IC-Commitment* on $f_j(x)$ is revealed by reconstructing it with the help of the parties in $OKSetP_j$ for every $P_j \in WCORE$. Then the values $f_j(0)$'s are used to construct the polynomial (if possible) $f_0(x)$ that is committed by D during sharing phase. Since $f_j(x)$ is a degree-t polynomial, any $t+1$ points on it are enough to interpolate $f_j(x)$. The points on $f_j(x)$ are obtained by requesting each party P_k in $OKSetP_j$ to reveal IC signature of D on $f_k(j)$ and IC signature of P_j on $f_j(k)$ such that $f_j(k) = f_k(j)$ holds. Asking $P_k \in OKSetP_j$ to reveal D's signature ensures that when D is *honest*, then even for a *corrupted* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one handed over by D to P_j in sharing phase. This helps the AWSS protocol to satisfy CORRECTNESS 1 property. Now asking P_k in $OKSetP_j$ to reveal P_j's signature ensures that even if D is *corrupted*, for an *honest* $P_j \in WCORE$, the reconstructed polynomial $f_j(x)$ will be same as the one received by P_j from D in the sharing phase. This ensures CORRECTNESS 2 property of AWSS. Summing up, when at least one of D and P_j is honest, P_j's *IC-Commitment* on $f_j(x)$ is revealed properly. But when both D and P_j are corrupted, P_j's *IC-Commitment* on $f_j(x)$ can be revealed as any t-degree polynomial $\overline{f_j(x)}$. It is this property that makes the protocol to qualify as an AWSS protocol rather than an AVSS protocol. Protocol AWSS-Single is given in Table 1.

Lemma 11 *Protocol AWSS-Single satisfies termination property.*

PROOF: TERMINATION 1: When D is honest, then every honest P_j will eventually complete its *IC-Commitment* on $f_j(x)$ with at least $2t+1$ honest parties in $OKSetP_j$. Hence, D will include all the $2t+1$ honest parties in $WCORE$ and A-cast the same. Now by the property of A-cast, each honest party will eventually listen $WCORE$ from the A-cast of D. Finally, since honest D had included P_j in $WCORE$ after listening the OK signals from the parties in $OKSetP_j$'s, each honest party will also listen them and will terminate AWSS-Single-Share.

Protocol AWSS-Single$(D, \mathcal{P}, s)$

AWSS-Single-Share$(D, \mathcal{P}, s)$

DISTRIBUTION: CODE FOR D

1. Select a random degree-t symmetric bivariate polynomial $F(x, y)$ such that $F(0, 0) = s$.
2. For $1 \leq i \leq n$, deliver $f_i(x) = F(x, i)$ to P_i, along with *IC signature* on each $f_i(j)$ by considering P_i as INT and executing $\mathsf{Gen}(D, P_i, \mathcal{P}, f_i(j))$ for every $j \in \{1, \ldots, n\}$.

VERIFICATION: CODE FOR P_i

1. Wait until $\mathsf{Gen}(D, P_i, \mathcal{P}, f_i(j))$ is completed for every $j \in \{1, \ldots, n\}$.
2. If $(f_i(1), \ldots, f_i(n)$ is t-consistent, then acting as INT, execute $\mathsf{Ver}(D, P_i, \mathcal{P}, f_i(j))$ for every $j \in \{1, \ldots, n\}$.
3. If $\mathsf{Ver}(D, P_i, \mathcal{P}, f_i(j))$ is completed with $\mathsf{Ver} = 1$ for all $j \in \{1, \ldots, n\}$, then hand over $f_i(j)$ to P_j, along with *IC signature* by acting as dealer, treating P_j as INT and executing $\mathsf{Gen}(P_i, P_j, \mathcal{P}, f_i(j))$ for all $j \in \{1, \ldots, n\}$. In addition, participate in $\mathsf{Gen}(P_j, P_i, \mathcal{P}, f_j(i))$ for all $j \in \{1, \ldots, n\}$ by acting as INT.
4. Wait until $\mathsf{Gen}(P_j, P_i, \mathcal{P}, f_j(i))$ is completed. If $f_i(j) = f_j(i)$ then execute $\mathsf{Ver}(P_j, P_i, \mathcal{P}, f_j(i))$ as an INT. If $\mathsf{Ver}(P_j, P_i, \mathcal{P}, f_j(i))$ is completed with $\mathsf{Ver} = 1$, then A-cast $\mathtt{OK}(P_i, P_j)$. Here $j \in \{1, \ldots, n\}$.

CORE CONSTRUCTION : CODE FOR D

1. For each P_j, build a set $OKSetP_j = \{P_i | D \textit{ listens } \mathtt{OK}(P_i, P_j)\}$. When $|OKSetP_j| = 2t + 1$, then P_j's *IC-Commitment* on $f_j(x)$ is over and add P_j in $WCORE$ (which is initially empty).
2. Wait until $|WCORE| = 2t + 1$. Then A-cast $WCORE$ and $OKSetP_j$ for all $P_j \in WCORE$.

CORE VERIFICATION & AGREEMENT ON CORE : CODE FOR P_i

1. Wait to obtain $WCORE$ and $OKSetP_j$ for all $P_j \in WCORE$ from D's A-cast, such that $|WCORE| = 2t + 1$ and $|OKSetP_j| = 2t + 1$ for each $P_j \in WCORE$.
2. Wait to receive $\mathtt{OK}(P_k, P_j)$ for all $P_k \in OKSetP_j$ and $P_j \in WCORE$. After receiving, accept the $WCORE$ and $OKSetP_j$'s and terminate **AWSS-Single-Share**.

AWSS-Single-Rec-Private$(\mathrm{D}, \mathcal{P}, s, P_\alpha)$: P_α-weak-private-reconstruction of s:

SIGNATURE REVELATION: CODE FOR P_i

1. If P_i belongs to $OKSetP_j$ for some $P_j \in WCORE$, then participate in Reveal-Private$(D, P_i, \mathcal{P}, f_i(j), P_\alpha)$ and Reveal-Private$(P_j, P_i, \mathcal{P}, f_j(i), P_\alpha)$ as an INT.
2. Participate in Reveal-Private$(D, P_k, \mathcal{P}, f_k(j), P_\alpha)$ and Reveal-Private$(P_j, P_k, \mathcal{P}, f_j(k), P_\alpha)$ for all $P_k \in OKSetP_j$ and $P_j \in WCORE$ as a verifier.

LOCAL COMPUTATION**:** CODE FOR P_α

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-Commitment* as follows:
   (a) Construct a set $ValidSetP_j = \emptyset$.
   (b) Add party $P_k \in OKSetP_j$ to $ValidSetP_j$ if both the following conditions are true:
      i. Reveal-Private$(D, P_k, \mathcal{P}, f_k(j), P_\alpha)$ and Reveal-Private$(P_j, P_k, \mathcal{P}, f_j(k), P_\alpha)$ are successfully completed, with outputs, $\mathsf{Reveal}_\alpha = \overline{f_k(j)}$ and $\mathsf{Reveal}_\alpha = \overline{f_j(k)}$ respectively; and
      ii. $\overline{f_k(j)} = \overline{f_j(k)}$.
   (c) Wait until $|ValidSetP_j| = t+1$. Construct a polynomial $\overline{f_j(x)}$ passing through the points $(k, \overline{f_j(k)})$ where $P_k \in ValidSetP_j$. Associate $\overline{f_j(0)}$ with $P_j \in WCORE$. *We say that $\overline{f_j(0)} = \overline{f_0(j)}$ is IC-Reconstructed towards P_α with the help of the parties in $OKSetP_j$.*
2. Wait for $\overline{f_j(0)} = \overline{f_0(j)}$ to be IC-Reconstructed for every P_j in $WCORE$.
3. Check whether the points $(j, \overline{f_j(0)})$ for $P_j \in WCORE$ lie on a unique t degree polynomial $\overline{f_0(x)}$. If yes, then compute $\overline{s} = \overline{f_0(0)}$ and terminate. Else set $\overline{s} = NULL$ and terminate.

Table 1: **AWSS for Sharing a Single Secret s with $n = 3t + 1$**

TERMINATION 2: If an honest P_i has completed AWSS-Single-Share, then he must have listened $WCORE$ and $OKSetP_j$'s from the A-cast of D and verified their validity. By properties of A-cast, each honest party will also listen the same and will eventually terminate AWSS-Single-Share.

TERMINATION 3: By Lemma 8, if P_i (acting as INT) is honest and $\mathsf{Ver} = 1$ at the end of **Verification Phase**, then IC signature produced by P_i during Reveal-Private will be accepted by an honest P_α, except with probability $2^{-\mathcal{O}(\kappa)}$. Since for every $P_j \in WCORE$, $|OKSetP_j| = 2t+1$, there are at least $t+1$ honest parties in $OKSetP_j$ who will be present in $ValidSetP_j$ with very high probability. Hence for every $P_j \in WCORE$, P_j's *IC-Commitment* will be reconstructed. Thus with very high probability, an honest P_α will terminate AWSS-Single-Rec-Private after executing the remaining steps of [LOCAL COMPUTATION]. □

Lemma 12 *Protocol AWSS-Single satisfies secrecy property.*

PROOF: Follows from the secrecy of A-ICP and properties of symmetric bivariate polynomial. □

Lemma 13 *Protocol AWSS-Single satisfies correctness property.*

PROOF: CORRECTNESS 1: Here we have to consider the case when D is *honest*. We first prove that if D is honest, then with very high probability, for each $P_j \in WCORE$, the value $\overline{f_j(0)}$ which is IC-reconstructed towards P_α, is same as $f_j(0)$ that was selected by D. From the property of A-ICP, for an *honest* $P_j \in WCORE$, a corrupted $P_k \in OKSetP_j$ can produce P_j's valid signature on incorrect $\overline{f_j(k)} \neq f_j(k)$ with negligible probability (see Lemma 9). Hence with very high probability $\overline{f_j(k)}$ is same as $f_j(k)$ for all $P_k \in ValidSetP_j$. Thus the polynomial $\overline{f_j(x)}$ reconstructed by P_α corresponding to an honest P_j in $WCORE$ is same as $f_j(x)$ that was selected by honest D. On the other hand, for a *corrupted* $P_j \in WCORE$, a corrupted $P_k \in OKSetP_j$ can produce P_j's valid signature on any $\overline{f_j(k)} \neq f_j(k)$ but P_k will fail to produce honest D's signature on $\overline{f_k(j)} = \overline{f_j(k)}$, with very high probability. Hence P_k will not be included in $ValidSetP_j$. Thus again the reconstructed polynomial $\overline{f_j(x)}$ corresponding to a corrupted P_j in $WCORE$ is same as $f_j(x)$. So P_α will correctly reconstruct $f_0(x) = F(x, 0)$ and hence the secret $s = f_0(0)$ with very high probability.

CORRECTNESS 2: Here we have to consider the case, when D is *corrupted*. Since in AWSS-Single-Share, every honest party agrees on a $WCORE$ and $OKSetP_j$ for $P_j \in WCORE$, a unique secret $s' \in \mathbb{F} \cup NULL$ is defined by (at least $t+1$) honest parties in $WCORE$ at the end of sharing phase. The committed secret s' is the constant term of the polynomial passing through points $(j, f_j(0))$'s, corresponding to *honest* P_j's in $WCORE$. If the points $(j, f_j(0))$ corresponding to honest P_j's in $WCORE$ define a unique t degree polynomial, say $f_0(x)$, then we say that D's committed secret is $s' = f_0(0)$. Otherwise, we say that D's committed secret is $s' = NULL$. Whatever may be case, we show that with very high probability, an honest P_α will either reconstruct s' or $NULL$.

We consider the first case when $s' = f_0(0)$. This implies that the points $(j, f_j(0))$ corresponding to honest P_j's in $WCORE$ define a unique t-degree polynomial $f_0(x)$. We now claim that with very high probability, the value $\overline{f_j(0)}$ (hence the polynomial $\overline{f_j(x)}$) corresponding to an honest $P_j \in WCORE$, that is IC-Reconstructed towards P_α, is same as $f_j(x)$ that P_j received in sharing phase. This claim follows from the argument given in CORRECTNESS 1. We next claim that the value $\overline{f_j(0)}$ (hence the polynomial $\overline{f_j(x)}$) corresponding to a *corrupted* $P_j \in WCORE$, which is IC-Reconstructed towards P_α can be any value. This is because for a *corrupted* P_j in $WCORE$, a corrupted $P_k \in OKSetP_j$ can produce a valid signature of P_j on any $\overline{f_j(k)}$ as well as a valid signature of D (who is corrupted as well) on $\overline{f_k(j)}$. Also adversary can delay the messages such that the values (along with the signatures) of all corrupted $P_k \in OKSetP_j$ are revealed to P_α before the values of honest parties in $OKSetP_j$. Thus the reconstructed polynomial $\overline{f_j(x)}$ can be any t-degree polynomial according to the choice of $\mathcal{A}_t$. Now there are two possibilities: if the points $(j, \overline{f_j(0)})$ corresponding to *honest* P_j's in $WCORE$, along with the points $(j, \overline{f_j(0)})$ corresponding to *corrupted* P_j's in $WCORE$ lie on $f_0(x)$, then s' will be reconstructed. Otherwise $NULL$ will be reconstructed. Notice that since for all *honest* parties in $WCORE$, $\overline{f_j(0)} = f_j(0)$, no other secret (other than s') can be reconstructed with very high probability.

We next consider the second case when D's committed secret is $NULL$. This implies that the points $(j, f_j(0))$ corresponding to honest P_j's in $WCORE$ do not define a unique t-degree polynomial. It is easy to see that in this case, irrespective of the behavior of the corrupted parties $NULL$ will be reconstructed. This is because the points $f_j(0)$ corresponding to each honest $P_j \in WCORE$ will be reconstructed correctly with very high probability. □

Theorem 4 *The pair (AWSS-Single-Share, AWSS-Single-Rec-Private) constitutes a valid $AWSS$ scheme for $n = 3t+1$ parties, which shares a single secret and satisfies the properties of $AWSS$.*

Note 5 (Important Remark:) *In AWSS-Single, the degree-t univariate polynomial $F(x,0) = f_0(x)$ is used to share the secret s. In the following we will say that D shares a degree-t polynomial $f(x)$ using protocol AWSS-Single by executing AWSS-Single-Share($D, \mathcal{P}, f(x)$). For this D selects a t-degree symmetric bivariate polynomial $F(x,y)$, such that $F(x,0) = f(x)$ and execute the protocol. The polynomial $f(x)$ is not random but only preserves the secrecy of the constant term; i.e., $s = f(0)$. Yet, this distribution of polynomials is sufficient to provide the secrecy requirements needed by our AVSS, where AWSS is used as a black box. If D indeed selects the bivariate polynomial in the above way and follows the protocol steps correctly, then as a result of the above execution, party P_i in WCORE will hold the i^{th} share $f(i) = F(i,0) = F(0,i)$ of s, the polynomial $f_i(x) = F(x,i)$ and the share-share $f_j(i) = F(i,j) = f_i(j)$ corresponding to every other party P_j. Similarly, AWSS-Single-Rec-Private($D, \mathcal{P}, f(x), P_\alpha$) can be used for the private reconstruction of $f(x)$ and the secret $s = f(0)$ for P_α. We call this reconstruction as P_α-weak-private-reconstruction of $f(x)$.* **Note that if D is corrupted, then P_α-weak-private-reconstruction of $f(x)$ may reconstruct either a t-degree polynomial $f(x)$ or $NULL$.**

We now extend protocol AWSS-Single (which shares a single secret) to protocol AWSS (given in Table 2) which concurrently shares a secret $S = (s^1 \dots s^\ell)$, containing ℓ secret field elements. The properties of AWSS follows from the properties of AWSS-Single.

Lemma 14 *Protocol AWSS-Share privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^2 \log n)$ bits. Protocol AWSS-Rec-Private privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits.*

PROOF: AWSS-Share executes at most $n + n^2 = \Theta(n^2)$ instances of Gen and Ver. Hence by Lemma 1, AWSS-Share privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits. A-casting of $WCORE$ and $OKSetP_j$'s require A-cast of $\mathcal{O}(n^2 \log n)$ bits. Protocol AWSS-Rec-Private executes $\Theta(n^2)$ instances of Reveal-Private and hence by Lemma 1, AWSS-Rec-Private privately communicates $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits. □

Theorem 5 *The pair (AWSS-Share, AWSS-Rec-Private) constitutes a valid AWSS scheme with $n = 3t+1$ parties, which shares ℓ secrets simultaneously and satisfies the properties of AWSS except with an error probability of $2^{-\mathcal{O}(\kappa)}$.*

Note 6 (Important Remark:) *As in AWSS-Single, in AWSS, the degree-t univariate polynomial $F^l(x,0) = f^l_0(x)$ is used to share the secret s^l for $l = 1, \dots, \ell$. In the following, we will say that D shares ℓ degree-t polynomials $f^1(x), \dots, f^\ell(x)$ simultaneously using protocol AWSS by executing AWSS-Share($D, \mathcal{P}, f^1(x), \dots, f^\ell(x)$). For this D selects ℓ t-degree symmetric bivariate polynomials $F^1(x,y), \dots,$ $F^\ell(x,y)$, such that for $l = 1, \dots, \ell, F^l(x,0) = f^l(x)$ and execute the protocol. The polynomials $f^l(x)$'s are not random but only preserve the secrecy of the constant terms $s^l = f^l(0)$'s. Yet, this distribution of polynomials is sufficient to provide the secrecy requirements needed by our AVSS, where AWSS is used as a building block. If D indeed selects the bivariate polynomial in the above way and follows the protocol steps correctly, then as a result of the above execution, party P_i in WCORE holds the i^{th} share $f^l(i) = F^l(i,0) = F^l(0,i)$ of s^l, the polynomial $f^l_i(x) = F^l(x,i)$ and the share-share $f^l_j(i) = F^l(i,j) = f^l_i(j)$ corresponding to every other party P_j. Similarly, AWSS-Rec-Private($D, \mathcal{P}, f^1(x), \dots, f^l(x), P_\alpha$) can be used for the private reconstruction of $f^1(x), \dots, f^l(x)$ and the secrets $s^1 = f^1(0), \dots, s^\ell = f^\ell(0)$ for P_α. We call this reconstruction as P_α-weak-private-reconstruction of $f^1(x), \dots, f^\ell(x)$.* **Note that if D is corrupted then P_α-weak-private-reconstruction of $f^1(x), \dots, f^\ell(x)$ may reconstruct either t-degree polynomials $f^1(x), \dots, f^\ell(x)$ or $NULL$.**

APPENDIX D: Protocol AVSS and Proof of Its Properties

Lemma 15 *Protocol AVSS-Single satisfies termination property.*

PROOF: **Termination 1:** When D is *honest*, the AWSS instances initiated by D, namely AWSS-Single-Share ($D, \mathcal{P}, f^i(x)$) for $i = 1, \dots, n$ will eventually terminate. Moreover, every P_i will privately reconstruct $f^i(x)$ during AWSS-Single-Rec-Private($D, \mathcal{P}, f^i(x), P_i$). Finally corresponding to every honest P_i, AWSS-Single-Share($P_i, \mathcal{P}, f^i(x)$) will eventually terminate with $WCORE^{P_i}$ containing all the honest parties. Hence D will eventually include all the honest parties in $VCORE$ and every honest party will eventually terminate AVSS-Single-Share after listening $VCORE$ from D and verifying the same.

Protocol $\mathsf{AWSS}(D, \mathcal{P}, S)$

$\mathsf{AWSS\text{-}Share}(D, \mathcal{P}, S)$

DISTRIBUTION: CODE FOR D

1. Select ℓ degree-t random symmetric bivariate polynomials $F^1(x,y), \ldots, F^\ell(x,y)$ such that for $l = 1, \ldots, \ell$, $F^l(0,0) = s^l$.
2. For $l = 1, \ldots, \ell$, deliver $f_i^l(x) = F^l(x,i)$ to P_i, along with *IC signature* by considering P_i as INT and executing $\mathsf{Gen}(D, P_i, \mathcal{P}, \Gamma_{ij})$ for every $j \in \{1, \ldots, n\}$ where $\Gamma_{ij} = (f_i^1(j), \ldots, f_i^\ell(j))$.

VERIFICATION: CODE FOR P_i

1. Wait until $\mathsf{Gen}(D, P_i, \mathcal{P}, \Gamma_{ij})$ for every $j \in \{1, \ldots, n\}$ are completed.
2. For $l = 1, \ldots, \ell$, check whether $(f_i^l(1), \ldots, f_i^l(n))$ is t-consistent. If yes, then acting as INT, execute $\mathsf{Ver}(D, P_i, \mathcal{P}, \Gamma_{ij})$ for every $j \in \{1, \ldots, n\}$.
3. If for all $j \in \{1, \ldots, n\}$, $\mathsf{Ver}(D, P_i, \mathcal{P}, \Gamma_{ij})$ gets over with $\mathsf{Ver} = 1$, then hand over Γ_{ij} to P_j, along with *IC signature* by executing $\mathsf{Gen}(P_i, P_j, \mathcal{P}, \Gamma_{ij})$ for all $j \in \{1, \ldots, n\}$. In addition, participate in $\mathsf{Gen}(P_j, P_i, \mathcal{P}, \Gamma_{ji})$ for all $j \in \{1, \ldots, n\}$ by acting as INT and considering P_j as dealer.
4. Wait until $\mathsf{Gen}(P_j, P_i, \mathcal{P}, \Gamma_{ji})$ is completed. If $\Gamma_{ij} = \Gamma_{ji}$ (i.e. for $l = 1, \ldots, \ell$ $f_i^l(j) = f_j^l(i)$), then execute $\mathsf{Ver}(P_j, P_i, \mathcal{P}, \Gamma_{ji})$. If $\mathsf{Ver}(P_j, P_i, \mathcal{P}, \Gamma_{ji})$ gets over with $\mathsf{Ver} = 1$, then $\mathsf{A\text{-}cast}$ $\mathtt{OK}(P_i, P_j)$. Here $j \in \{1, \ldots, n\}$.

CORE CONSTRUCTION : CODE FOR D– Same as in Protocol $\mathsf{AWSS\text{-}Single\text{-}Share}$

CORE VERIFICATION & AGREEMENT ON CORE : CODE FOR P_i– same as in Protocol $\mathsf{AWSS\text{-}Single\text{-}Share}$

$\mathsf{AWSS\text{-}Rec\text{-}Private}(D, \mathcal{P}, S, P_\alpha)$: P_α-weak-private-reconstruction of S:

SIGNATURE REVELATION: CODE FOR P_i

1. If P_i belongs to $OKSetP_j$ for some $P_j \in WCORE$, then participate in $\mathsf{Reveal\text{-}Private}(D, P_i, \mathcal{P}, \Gamma_{ij}, P_\alpha)$ and $\mathsf{Reveal\text{-}Private}(P_j, P_i, \mathcal{P}, \Gamma_{ji}, P_\alpha)$ as an INT.
2. Participate in $\mathsf{Reveal\text{-}Private}(D, P_k, \mathcal{P}, \Gamma_{kj}, P_\alpha)$ and $\mathsf{Reveal\text{-}Private}(P_j, P_k, \mathcal{P}, \Gamma_{jk}, P_\alpha)$ for all $P_k \in OKSetP_j$ and $P_j \in WCORE$ as a verifier.

LOCAL COMPUTATION: CODE FOR P_α

1. For every $P_j \in WCORE$, reconstruct P_j's *IC-Commitment* as follows:
   (a) Construct a set $ValidSetP_j = \emptyset$.
   (b) Add party $P_k \in OKSetP_j$ to $ValidSetP_j$ if both the following conditions are true:
      i. $\mathsf{Reveal\text{-}Private}(D, P_k, \mathcal{P}, \Gamma_{kj}, P_\alpha)$ and $\mathsf{Reveal\text{-}Private}(P_j, P_k, \mathcal{P}, \Gamma_{jk}, P_\alpha)$ are successfully completed, with $\mathsf{Reveal}_\alpha = \overline{\Gamma_{kj}} = (\overline{f_k^1(j)}, \ldots, \overline{f_k^\ell(j)})$ and $\mathsf{Reveal}_\alpha = \overline{\Gamma_{jk}} = (\overline{f_j^1(k)}, \ldots, \overline{f_j^\ell(k)})$ respectively;
      ii. $\overline{\Gamma_{kj}} = \overline{\Gamma_{jk}}$ i.e. For $l = 1, \ldots, \ell$, $\overline{f_k^l(j)} = \overline{f_j^l(k)}$.
   (c) Wait until $|ValidSetP_j| = t+1$. For $l = 1, \ldots, \ell$, construct a polynomial $\overline{f_j^l(x)}$ passing through the points $(k, \overline{f_j^l(k)})$ where $P_k \in ValidSetP_j$. Associate $\overline{f_j^l(0)}$ with $P_j \in WCORE$. We say that for $l = 1, \ldots, \ell$, $\overline{f_j^l(0)} = \overline{f_0^l(j)}$ are $\mathsf{IC\text{-}Reconstructed}$ towards P_α with the help of the parties in $OKSetP_j$.
2. For every P_j in $WCORE$, wait for $\overline{f_j^l(0)} = \overline{f_0^l(j)}$ to be $\mathsf{IC\text{-}Reconstructed}$ for $l = 1, \ldots, \ell$.
3. For every $l \in \{1, \ldots, \ell\}$ do the following: Check whether the points $(j, \overline{f_j^l(0)})$ for $P_j \in WCORE$ lie on a unique t degree polynomial $\overline{f_0^l(x)}$. If yes, then set $\overline{s^l} = \overline{f_0^l(0)}$, output $\overline{S} = (\overline{s^l}, \ldots, \overline{s^\ell})$ and terminate. Else if there exists an $l \in \{1, \ldots, \ell\}$, such that points $(j, \overline{f_j^l(0)})$ for $P_j \in WCORE$ does not lie on a unique t degree polynomial, then output $\overline{S} = NULL$ and terminate.

Table 2: **AWSS for Sharing Secret $S = (s^1, \ldots, s^\ell)$ containing ℓ field elements with $n = 3t+1$**

Termination 2: If an honest P_i has completed $\mathsf{AVSS\text{-}Single\text{-}Share}$, then he must have listened $VCORE$ and $WCORE^{P_j}$'s from the $\mathsf{A\text{-}cast}$ of D and is assured about their validity. By properties of $\mathsf{A\text{-}cast}$, each honest party will also listen the same and will eventually terminate $\mathsf{AVSS\text{-}Single\text{-}Share}$.

Termination 3: By **Termination 3** and **Correctness 1** of AWSS protocol (see Lemma 11 and Lemma 13), $\mathsf{AWSS\text{-}Single}(P_i, \mathcal{P}, f^i(x))$ initiated by an *honest* P_i in $VCORE$, will eventually lead to the reconstruction of $f^i(x)$ in its reconstruction phase with very high probability. But $\mathsf{AWSS\text{-}Single}(P_i, \mathcal{P}, f^i(x))$ initiated by a *corrupted* P_i in $VCORE$, may lead to the reconstruction of $NULL$ in its reconstruction phase. Since $|VCORE| = 2t+1$, for at least $t+1$ honest parties from $VCORE$, reconstruction of $f^i(x)$'s will be successful. This is enough to reconstruct the secret s. Hence if an honest party terminates AVSS-

Single-Share and every (honest) party starts AVSS-Single-Rec-Private, then an honest P_α will eventually terminate AVSS-Single-Rec-Private. □

Lemma 16 *Protocol AVSS-Single satisfies secrecy property.*

PROOF: Follows from the secrecy of AWSS-Single and properties of t-degree polynomials. □

Lemma 17 *Protocol AVSS-Single satisfies correctness property.*

PROOF: CORRECTNESS 1: We have to consider the case when D is *honest*. We now prove that if D is honest, then with very high probability, for every $P_j \in REC$, $\overline{f^j(x)}$ is equal to $f^j(x)$ that is selected by D at the beginning of AVSS-Single-Share. For an honest P_j, this is true. If P_j is corrupted then also our claim holds. This is because of the incorporation of step 4 in [RE-COMMITMENT BY INDIVIDUAL PARTY] of our protocol. The step 4 ensures that if a corrupted P_j completes his instance of re-commitment, namely AWSS-Single-Share($P_j, \mathcal{P}, f^j(x)$), with $WCORE^{P_j}$, then the honest parties (at least $t+1$) in $WCORE^{P_j}$ defines the same $f^j(x)$ as selected by D at the beginning of AVSS-Single-Share.

CORRECTNESS 2: Here we have to consider the case, when D is *corrupted* and some honest party has terminated AVSS-Single-Share. We now show that D has indeed committed a secret $s' \in \mathbb{F}$, which will be uniquely reconstructed in the reconstruction phase. The secret s' is defined by a unique $n \times n$ matrix M, whose each row and column is t-consistent (recall our discussion before the presentation of protocol AVSS-Single). If an honest $P_i \in WCORE^D$ has received $f^1(i), \ldots, f^n(i)$ from D's AWSS instances, then $f^1(i), \ldots, f^n(i)$ is the i^{th} row of M, where $f^1(i), \ldots, f^n(i)$ is t-consistent. The columns (at least $(t+1)$) of M are defined by the polynomials (of degree t) that the honest parties (at least $t+1$) in $VCORE$ have recommitted. So if an honest $P_i \in VCORE$ has recommitted $f^i(x)$, then $(f^i(1), \ldots, f^i(n))$ is the i^{th} column of M. It is easy to see that these (at least $t+1$) t-consistent rows and t-consistent columns of M (defined by the honest parties in $VCORE$ and $WCORE^D$) uniquely and completely fix the remaining elements of M. Let us denote the t degree polynomial defined by the values in j^{th} column of M as $f^j(x)$ for every $P_j \in \mathcal{P} \setminus VCORE$. It should be noted that if an honest party $P_i \in \mathcal{P} \setminus VCORE$ initiates recommitment (by initiating an AWSS instance as a dealer), then he is intending to recommit $f^i(x)$. We now show that s' defined by M will be reconstructed in the reconstruction phase.

To show this, it enough to show that every $P_i \in VCORE$ has recommitted $f^i(x)$ to the honest parties in $WCORE^{P_i}$. For an honest $P_i \in VCORE$, this is always true. Now to prevent a corrupted $P_i \in VCORE$ from recommitting $\overline{f^i(x)} \neq f^i(x)$ (i.e something other than i^{th} column of M), a corrupted P_i is forced to deliver $f^i(j)$ to every honest party $P_j \in WCORE^{P_i}$ during his recommitment. To prove the above statement, consider an honest $P_j \in WCORE^{P_i}$. P_j is allowed to be included in $WCORE^{P_i}$ only when P_j is in $ProbCORE$ of at least $2t+1$ parties. Out of these $2t+1$ parties at least $t+1$ P_k's are honest and thus their $f^k(x)$'s defines valid columns of M. Now since P_j is honest, it implies that P_j has correct points on these $t+1$ correct $f^k(x)$'s. This together with the fact that P_j have also checked t-consistency of $(f^1(j), \ldots, f^n(j))$ (since P_j is present in $ProbCORE^{P_k}$ and hence $WCORE^D_k$ of $t+1$ honest P_k's), implies that P_j has received a valid row of M from D during the AWSS instances of D. Finally since P_j has participated in P_i's recommitment and is considered to be part of $ProbCORE^{P_i}$, it is clear that P_i has passed on $f^i(j)$ to P_j during execution of his recommitment. This clearly implies that P_i has recommitted $f^i(x)$ to the honest in $WCORE^{P_i}$. In the reconstruction phase, for a corrupted P_i, AWSS-Single-Rec-Private($P_i, \mathcal{P}, f^i(x), P_\alpha$) can not reveal any t-degree polynomial other than $f^i(x)$. Therefore in the reconstruction phase s' will be reconstructed. □

We now present protocol AVSS, dealing with multiple secrets concurrently. The proofs of the properties of AVSS follows from the properties of AVSS-Single.

Protocol **AVSS**$(D, \mathcal{P}, S)$: $S = (s^1, \ldots, s^\ell)$

AVSS-Share$(D, \mathcal{P}, S)$

D's Commitment:

i. Code for D:

1. Select ℓ degree-t random univariate polynomials $g^1(x), \ldots, g^\ell(x)$ such that for $l = 1, \ldots, \ell$, $g^l(0) = s^l$. Compute $(f^{(l,1)}(x), \ldots, f^{(l,n)}(x)) = \mathsf{Generate}(g^l(x))$.
2. Denote $S_i = (f^{(1,i)}(x), \ldots, f^{(\ell,i)}(x))$. Initiate n instances of AWSS-Share, AWSS-Share$(D, \mathcal{P}, S_i)$ for $i = 1, \ldots, n$, by executing the code in [Distribution: Code for D] given in Table 2.

ii. Code for P_i:

1. For $j = 1, \ldots, n$, synchronize each step of [Verification: Code for P_i] in AWSS-Share$(D, \mathcal{P}, S_j)$. That is, execute step k of [Verification: Code for P_i] and if the requirements (if any) of k^{th} step are met for all the n instances, then proceed to step $k+1$ for all n instances.
2. As an additional requirement, after the completion of step 1 of [Verification: Code for P_i], check whether $(1, f^{(l,1)}(i)), (2, f^{(l,2)}(i)), \ldots, (n, f^{(l,n)}(i))$ lies on a degree-t polynomial, for each $l \in \{1, \ldots, \ell\}$. If yes proceed further to participate and execute step 2 of [Verification: Code for P_i] for all the n instances.
3. If step 4 of [Verification: Code for P_i] is executed successfully for all the n instances of D's AWSS-Share, then A-Cast a single $\mathtt{OK}(P_i, P_j)$ for all the n instances instead of n $\mathtt{OK}(P_i, P_j)$'s for each one of them.

iii. WCore Construction: Code for D– Construct only one copy of $WCORE$, called $WCORE^D$, common for all the n invocations of AWSS initiated by D, by following the steps for core construction in AWSS-Share.

iv. WCore verification & Agreement: Code for P_i– Similar to the description in AWSS-Share.

P_j-Weak-Private-Reconstruction of S_j for $j = 1, \ldots, n$: (Code for P_i:)

1. After executing step iv. of D's Commitment, participate in AWSS-Rec-Private$(D, \mathcal{P}, S_j, P_j)$, for $j = 1, \ldots, n$, for enabling P_j to privately reconstruct S_j. At the completion of AWSS-Rec-Private$(D, \mathcal{P}, S_i, P_i)$, obtain either $S_i = (f^{(1,i)}(x), \ldots, f^{(\ell,i)}(x))$ consisting of ℓ degree-t polynomials or $S_i = NULL$.

Re-Commitment by Individual Party

i. Code for P_i

1. If $S_i = (f^{(1,i)}(x), \ldots, f^{(\ell,i)}(x))$ is reconstructed in AWSS-Rec-Private$(D, \mathcal{P}, S_i, P_i)$, then acting as a dealer, initiate AWSS-Share$(P_i, \mathcal{P}, S_i)$.
2. Assign $WCORE_i^D = WCORE^D$, where $WCORE_i^D$ denotes the local copy of $WCORE^D$ for party P_i. Keep updating $WCORE_i^D$ locally with new parties, where a new party P_j will be included in $WCORE_i^D$ if at least $2t+1$ $\mathtt{OK}(., P_j)$s are A-casted in the AWSS instances initiated by D, i.e $|OKSetP_j| \geq 2t+1$.
3. If P_j is a new entrant in $WCORE^D$, participate (as INT or/and as verifier) in IC-Reconstructing $(f^{(1,k)}(j), \ldots, f^{(\ell,k)}(j))$ towards P_k for $k = 1, \ldots, n$.
4. Participate in AWSS-Share$(P_j, \mathcal{P}, S_j)$ if (A) $P_i \in WCORE_i^D$ and (B) $f^{(1,j)}(i), \ldots, f^{(\ell,j)}(i)$ received from D in AWSS-Share$(D, \mathcal{P}, S_j)$ is same as $f^{(1,j)}(i), \ldots, f^{(\ell,j)}(i)$ now received from P_j in AWSS-Share$(P_j, \mathcal{P}, S_j)$.
5. $WCORE^{P_i}$ Construction for AWSS-Share$(P_i, \mathcal{P}, S_i)$: Here P_i as a dealer, constructs $WCORE$ for AWSS-Share$(P_i, \mathcal{P}, S_i)$ in a different way (this is not same as in protocol AWSS-Share) to prove that *he has indeed re-committed* S_i.
   (a) Construct a set $ProbCORE^{P_i}$ ($= \emptyset$ initially). Include a party P_j in $ProbCORE^{P_i}$ and A-cast $\mathtt{Message}(P_j, ProbCORE^{P_i})$ if (A) At least $2t+1$ A-casts of the form $\mathtt{OK}(., P_j)$ are heard in the instance AWSS-Share$(P_i, \mathcal{P}, S_i)$, (B) $P_j \in WCORE_i^D$ and (C) P_j's points on $f^{(1,i)}(x), \ldots, f^{(\ell,i)}(x)$ (namely $(f^{(1,i)}(j), \ldots, f^{(\ell,i)}(j))$) which are IC-Reconstructed towards P_i are consistent with the re-committed polynomials $(f^{(1,i)}(x), \ldots, f^{(\ell,i)}(x))$.
   (b) On listening $\mathtt{Message}(P_j, ProbCORE^{P_k})$, consider it as *valid* if there are at least $2t+1$ A-casts of the form $\mathtt{OK}(., P_j)$ in the instance AWSS-Share$(P_k, \mathcal{P}, S_k)$.
   (c) Construct $WCORE^{P_i}$. Add P_j in $WCORE^{P_i}$ if (A) $P_j \in ProbCORE^{P_i}$ and (B) for at least $2t+1$ P_k's, *valid* $\mathtt{Message}(P_j, ProbCORE^{P_k})$ is obtained. A-cast $WCORE^{P_i}$ when $|WCORE^{P_i}| = 2t+1$.

ii. Final CORE Construction: Code for D

1. Create a list $VCORE$. Include P_i in $VCORE$ if *a valid* $WCORE^{P_i}$ is listened from P_i. Here $WCORE^{P_i}$ is *valid*, if P_i has indeed constructed it by following the steps in 5 (a(A), c(B)).
2. A-cast $VCORE$ and $WCORE^{P_i}$'s for each P_i in $VCORE$ when $|VCORE| = 2t+1$.

iii. CORE Verification & Agreement on CORE: Code for P_i

1. Terminate after listening $VCORE$ and $WCORE^{P_j}$'s from D's A-Cast, where $|VCORE| = 2t+1$, such that for each P_j in $VCORE$, $WCORE^{P_j}$ is *valid.*

Protocol **AVSS**$(D, \mathcal{P}, S)$: $S = (s^1, \ldots, s^\ell)$

AVSS-Rec-Private$(D, \mathcal{P}, S, P_\alpha)$**:** Private reconstruction of S by party P_α:

P_α-WEAK-PRIVATE-RECONSTRUCTION OF S_j FOR EVERY $P_j \in VCORE$: CODE FOR P_i

1. Participate in AWSS-Rec-Private$(P_j, \mathcal{P}, S_j, P_\alpha)$ for every $P_j \in VCORE$.

LOCAL COMPUTATION: CODE FOR P_α

1. For every $P_j \in VCORE$, obtain either $\overline{S_j} = (\overline{f^{(1,j)}(x)}, \ldots, \overline{f^{(\ell,j)}(x)})$ or $NULL$ from P_α-weak-private-reconstruction of S_j.
2. Add party $P_j \in VCORE$ to REC if $\overline{S_j} \neq NULL$.
3. Wait until $|REC| = t + 1$. For every $l = \{1, \ldots, \ell\}$, construct polynomial $\overline{g^l(x)}$ passing through the points $(k, \overline{f^{(l,k)}(0)})$ where $P_k \in REC$.
4. Compute $\overline{s^l} = g^l(0)$ for $l = 1, \ldots, \ell$ and terminate.

We presented AVSS with private reconstruction, where the secrets are available only to a specific party P_α at the end of reconstruction phase. We call the private reconstruction as *P_α-private-reconstruction*. Thus though we have an implementation of public reconstruction for AVSS, we skip it here as our implementation of ACVSS protocol (where AVSS is used as a black-box), requires private reconstruction of secrets. So we have the following theorem:

Theorem 6 *The pair (AVSS-Share, AVSS-Rec-Private) constitutes a valid AVSS scheme with $n = 3t + 1$, which shares $\ell \geq 1$ secret values concurrently and satisfies the properties of AVSS except with an error probability of $2^{-\mathcal{O}(\kappa)}$.*

Lemma 18 *Protocol AVSS-Share privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^3 \log n)$ bits. Protocol AVSS-Rec-Private privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits.*

PROOF: AVSS-Share executes $2n$ instances of AWSS-Share and n instances of AWSS-Rec-Private. Hence by Lemma 14, AVSS-Share privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits and A-casts $\mathcal{O}(n^3 \log n)$ bits. AVSS-Rec-Private executes n instances of AWSS-Rec-Private and hence by Lemma 14, AVSS-Rec-Private privately communicates $\mathcal{O}((\ell n^3 + n^4\kappa)\kappa)$ bits. □

Note 7 (Important Remark:) *As in AVSS-Single, we may invoke AVSS as AVSS-Share($D, \mathcal{P}, g^1(x), \ldots, g^\ell(x)$) (instead of AVSS-Share($D, \mathcal{P}, S$)) and by doing so, we mean that D executes AVSS-Share with ℓ degree-t polynomials $g^1(x), \ldots, g^\ell(x)$, such that for $l = 1, \ldots, \ell$, $g^l(0) = s^l$. As a result of this, party P_i in $VCORE$ gets the shares $g^1(i), \ldots, g^\ell(i)$. The polynomials $g^1(x), \ldots, g^\ell(x)$ are not completely random but preserves the secrecy of $s^1 = g^1(0), \ldots, s^\ell = g^\ell(0)$.*

APPENDIX E: Protocol **ACVSS**

Protocol $\mathsf{ACVSS}(D, \mathcal{P}, S)$: $S = (s^1, \ldots, s^\ell)$

ACVSS-Share($D, \mathcal{P}, S$)

D'S COMMITMENT:

i. CODE FOR D:

1. For $l = 1, \ldots, \ell$, select a degree-t random polynomial $h^l(x)$ with $h^l(0) = s^l$ and compute $(g^{(l,1)}(x), \ldots, g^{(l,n)}(x)) = \mathsf{Generate}(h^l(x))$.
2. Denote $\Delta_i = (g^{(1,i)}(x), \ldots, g^{(\ell,i)}(x))$. For $i = 1, \ldots, n$, initiate $\mathsf{AVSS\text{-}Share}(D, \mathcal{P}, \Delta_i)$.

ii. CODE FOR P_i:

1. Participate in $\mathsf{AVSS\text{-}Share}(D, \mathcal{P}, \Delta_j)$ for $j = 1, \ldots, n$.
2. Wait to construct $WCORE^{P_i}$ in each of the n instances of AVSS-Share. For $j = 1, \ldots, n$, let $WCORE^{(P_i,j)}$ be the $WCORE^{P_i}$, constructed by P_i in $\mathsf{AVSS\text{-}Share}(D, \mathcal{P}, \Delta_j)$. Wait until $|WCORE^{(P_i,*)}| \geq 2t+1$, where $WCORE^{(P_i,*)} = \cap_{i=1}^{n} WCORE^{(P_i,j)}$.
3. For every $l = 1, \ldots, \ell$, check whether the points $((1, g^{(l,1)}(i)), \ldots, (n, g^{(l,n)}(i)))$ lie on a t degree polynomial, where $g^{(l,j)}(i)$ is obtained during execution of $\mathsf{AVSS\text{-}Share}(D, \mathcal{P}, \Delta_j)$. If yes then A-cast $WCORE^{(P_i,*)}$ which will be considered as common $WCORE^{P_i}$ for all the n instances of AVSS-Share initiated by D.

iii. VCORE CONSTRUCTION: CODE FOR D: Create a single copy of $VCORE$ for all the n instances of AVSS-Share by following the steps for core construction and A-cast the same along with $WCORE^{(P_i,*)}$ for each $P_i \in VCORE$.

iv. VERIFICATION & AGREEMENT ON VCORE: CODE FOR P_i: Similar as in Protocol AVSS-Share.

P_j-PRIVATE-RECONSTRUCTION OF Δ_j FOR $j = 1, \ldots, n$: CODE FOR P_i:

1. For $j = 1, \ldots, n$, participate in $\mathsf{AVSS\text{-}Rec\text{-}Private}(D, \mathcal{P}, \Delta_j, P_j)$ for P_j-private-reconstruction of Δ_j, after terminating all the n instances of AVSS-Share.
2. Obtain $\Delta_i = (g^{(1,i)}(x), \ldots, g^{(\ell,i)}(x))$ from $\mathsf{AVSS\text{-}Rec\text{-}Private}(D, \mathcal{P}, \Delta_i, P_i)$, compute $h^l(i) = g^{(l,i)}(0)$, the i^{th} shares of s^l for $l = 1, \ldots, \ell$ and terminate **ACVSS-Share**.

ACVSS-Rec-Private($D, \mathcal{P}, S, P_\alpha$): Private reconstruction of S by party P_α:

CODE FOR P_i 1. For $l = 1, \ldots, \ell$, send $h^l(i)$, the i^{th} shares of the secrets to P_α.

LOCAL COMPUTATION: CODE FOR P_α

1. For $l = 1, \ldots, \ell$, upon receiving at least $2t+1$ t-consistent shares, $\overline{h^l(i)}$'s, interpolate a degree-t polynomial $\overline{h^l(x)}$, compute the secret $\overline{s^l} = \overline{h^l(0)}$ and terminate **ACVSS-Rec-Private**.

ACVSS-Rec-Public($D, \mathcal{P}, S, \mathcal{P}$): Public reconstruction of S for $\mathcal{P}$:

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

APPENDIX F: Proof of the Properties of **t-2D-Share**

Lemma 19 *In protocol t-2D-Share, if D is honest, then each honest party will eventually terminate with correct t-2D-sharing. If D is corrupted, then with very high probability, the honest parties will terminate only if D has done correct t-2D-sharing.*

PROOF: The first part is easy to proof. For the second part, we consider the case when D is corrupted. For $i = 0, \ldots, n$, $\mathsf{ACVSS\text{-}Share}^i$ ensures that D has correctly t-1D-shared some S^i. But it may happen that some S^i t-1D-shared by D does not contain the correct i^{th} shares of S^0. Assume that D has t-1D-shared $\overline{S^j} \neq S^j$ in $\mathsf{ACVSS\text{-}Share}^j$. This implies that in $\mathsf{ACVSS\text{-}Share}^j$, D has used polynomials $\overline{q^{(0,j)}(x), \ldots, q^{(\ell,j)}(x)}$ to share $\overline{S^j}$, such that for at least one $l \in \{0, \ldots, \ell\}$, $\overline{q^{(l,j)}(0)} \neq q^l(j) = s^l_j$. That is, $\overline{q^{(l,j)}(0)} = \overline{s^l_j} \neq s^l_j$. Now consider $q^*_j(0) = s^0_j + rs^1_j + \ldots + r^l\overline{s^l_j} + \ldots + r^\ell s^\ell_j$. We claim that with very high probability $q^*(j) \neq q^*_j(0)$. The probability that $q^*(j) = q^*_j(0)$ is same as the probability that two different ℓ degree polynomials with coefficients $(s^0_j, \ldots, \overline{s^l_j}, \ldots, s^\ell_j)$ and $(s^0_j, \ldots, s^l_j, \ldots, s^\ell_j)$ intersect at a random value r. Since any two ℓ degree polynomial can intersect each other at most at ℓ values, r has to be one of the ℓ values. But r is chosen randomly after the completion of all $\mathsf{ACVSS\text{-}Share}^j$ for $j = 0, \ldots, n$ (so during executions of $\mathsf{ACVSS\text{-}Share}^i$'s D is unaware of r). So the above event can happen with probability at most $\frac{\ell}{|\mathbb{F}|} \approx 2^{-\mathcal{O}(\kappa)}$. Thus with probability at least $1 - 2^{-\mathcal{O}(\kappa)}$, $q^*(j) \neq q^*_j(0)$. So Ver_i will be 0 for every honest $P_i \in \mathcal{P}$ and thus no honest party will terminate the protocol. □

Lemma 20 *If D is honest, then $s^1, \ldots, s^\ell$ will remain information theoretically secure in t-2D-Share.*

PROOF: Without loss of generality, let $P_1, \ldots, P_t$ be under the control of $\mathcal{A}_t$. If D is honest then from the properties of ACVSS-Share, the secrets $s^0, s^1, \ldots, s^\ell$ and shares $S^{t+1}, \ldots, S^n$ will remain secure after the execution of ACVSS-Share^0 and ACVSS-Share$^{t+1}, \ldots,$ ACVSS-Share^n respectively. It is easy to see that even if s^* and s_i^*'s are publicly reconstructed, the secrets $s^1, \ldots, s^\ell$ will remain information theoretic secure. □

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

PROOF: Protocol t-2D-Share calls $n+1$ instances of ACVSS-Share with $\ell+1$ secrets each. Hence by Theorem 1, it incurs a private communication of $\mathcal{O}((\ell n^5 + n^6\kappa)\kappa)$ bits and A-cast of $\mathcal{O}(n^5\log(n))$ bits. The protocol for random number generation, privately communicates $\mathcal{O}(n^6\kappa^2)$ bits and A-casts $\mathcal{O}(n^5\log n)$ bits . The instance of ACVSS-Rec-Public requires $\mathcal{O}(n^3\kappa)$ bits of private communication. Finally, n instances of ACVSS-Rec-Private requires $\mathcal{O}(\ell n^2\kappa)$ bits of private communication. □

APPENDIX G: Protocols used in Preparation Phase

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

CODE FOR P_i:

1. Select $L = \frac{\ell}{n-2t}$ random secret elements $(s^{(i,1)}, \ldots, s^{(i,L)})$. As a dealer, invoke t-2D-Share$(P_i, \mathcal{P}, S^i)$ to generate t-2D-sharing of $S^i = (s^{(i,1)}, \ldots, s^{(i,L)})$.
2. For $j = 1, \ldots, n$, participate in t-2D-Share$(P_j, \mathcal{P}, S^j)$.

AGREEMENT ON A CORE-SET: CODE FOR P_i

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

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

1. Wait until ACS completes with output C containing $n-t$ parties. For every $P_j \in C$, obtain the i^{th} shares $s_i^{(j,1)}, \ldots, s_i^{(j,L)}$ of S^j and i^{th} share-share $s_{ki}^{(j,1)}, \ldots, s_{ki}^{(j,L)}$ of shares $s_k^{(j,1)}, \ldots, s_k^{(j,L)}$, corresponding to each P_k, for $k = 1, \ldots, n$. Without loss of generality, let $C = \{P_1, \ldots, P_{n-t}\}$.
2. Let V denotes a $(n-t) \times (n-2t)$ publicly known *Vandermonde Matrix* [16] over $\mathbb{F}$.
   (a) For every $k \in \{1, \ldots, L\}$, let $(r^{(1,k)}, \ldots, r^{(n-2t,k)}) = (s^{(1,k)}, \ldots, s^{(n-t,k)})V$.
   (b) Locally compute i^{th} share of $r^{(1,k)}, \ldots, r^{(n-2t,k)}$ as $(r_i^{(1,k)}, \ldots, r_i^{(n-2t,k)}) = (s_i^{(1,k)}, \ldots, s_i^{(n-t,k)})V$.
   (c) For each $1 \leq j \leq n$, locally compute the i^{th} share-share of share $(r_j^{(1,k)}, \ldots, r_j^{(n-2t,k)})$ as $(r_{ji}^{(1,k)}, \ldots, r_{ji}^{(n-2t,k)}) = (s_{ji}^{(1,k)}, \ldots, s_{ji}^{(n-t,k)})V$ and terminate.

The values $r^{(1,1)}, \ldots, r^{(n-2t,1)}, \ldots, r^{(1,L)}, \ldots, r^{(n-2t,L)}$ denotes the ℓ random secrets which are t-2D-shared.

Table 3: Protocol for Collectively Generating t-2D-Sharing of ℓ secrets, $n = 3t+1$

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

SHARING BY D:

1. CODE FOR D: (a) Select ℓ non-zero random elements $\beta^1, \ldots, \beta^\ell$ from $\mathbb{F}$. For $1 \leq l \leq \ell$, let $c^l = a^l b^l$ and $d^l = b^l\beta^l$. Let $\mathcal{B} = (\beta^1, \ldots, \beta^\ell)$, $\mathcal{C} = (c^1, \ldots, c^\ell)$ and $\Lambda = (d^1, \ldots, d^\ell)$.
   (b) Invoke ACVSS-Share$(D, \mathcal{P}, \mathcal{B})$, ACVSS-Share$(D, \mathcal{P}, \mathcal{C})$ and ACVSS-Share$(D, \mathcal{P}, \Lambda)$.
2. CODE FOR P_i: Participate in the ACVSS-Share protocols initiated by D to obtain the i^{th} share $(\beta_i^1, \ldots, \beta_i^\ell)$, $(c_i^1, \ldots, c_i^\ell)$ and $(d_i^1, \ldots, d_i^\ell)$ of $\mathcal{B}, \mathcal{C}$ and Λ respectively.

VERIFYING WHETHER $c^l = a^l.b^l$: CODE FOR P_i

1. Once the three instances of ACVSS-Share initiated by D are terminated, participate in protocol RNG (given in Note 4 in section 3.4) to jointly generate a random non-zero value $r \in \mathbb{F}$.
2. For $l = 1, \ldots, \ell$, locally compute $p_i^l = ra_i^l + \beta_i^l$, the i^{th} share $p^l = ra^l + \beta^l$. Participate in ACVSS-Rec-Public$(D, \mathcal{P}, (p^1, \ldots, p^\ell), \mathcal{P})$ to publicly reconstruct p^l for $l = 1, \ldots, \ell$.
3. Upon reconstruction of p^l's, locally compute $q_i^l = p^l b_i^l - d_i^l - rc_i^l$ for $1 \leq l \leq \ell$, to get the i^{th} share of $q^l = p^l b^l - d^l - rc^l$. Participate in ACVSS-Rec-Public$(D, \mathcal{P}, (q^1, \ldots, q^\ell), \mathcal{P})$ to publicly reconstruct q^l for $l = 1, \ldots, \ell$.
4. Upon reconstruction of q^l's, locally check whether for $l = 1, \ldots, \ell$, $q^l \stackrel{?}{=} 0$. If yes then terminate.

Table 4: Protocol for Generating t-1D-sharing of $[c^1]_t = [a^1]_t.[b^1]_t, \ldots, [c^\ell]_t = [a^\ell]_t.[b^\ell]_t$, $n = 3t+1$

Protocol PreparationPhase($\mathcal{P}$)

Code for P_i:

1. Participate in two instances of Random-t-2D-Share($\mathcal{P}, c_M + c_R$) to generate t-2D-sharing of $a^1, \ldots, a^{c_M+c_R}$ and $b^1, \ldots, b^{c_M+c_R}$. Obtain the i^{th} shares $a_i^1, \ldots, a_i^{c_M+c_R}, b_i^1, \ldots, b_i^{c_M+c_R}$ and share-shares $a_{ji}^1, \ldots, a_{ji}^{c_M+c_R}, b_{ji}^1, \ldots, b_{ji}^{c_M+c_R}$.
2. For $1 \leq k \leq c_M + c_R$, let $c^k = a^k b^k$. Upon termination of both the instances of Random-t-2D-Share, invoke ProveCeqAB($P_i, \mathcal{P}, [a_i^1]_t, \ldots, [a_i^{c_M+c_R}]_t, [b_i^1]_t, \ldots, [b_i^{c_M+c_R}]_t$) as a dealer, to generate t-1D-sharing of $c_i^1, \ldots, c_i^{c_M+c_R}$, where c_i^k is the i^{th} share of c^k.
3. For $j = 1, \ldots, n$, participate in ProveCeqAB($P_j, \mathcal{P}, [a_j^1]_t, \ldots, [a_j^{c_M+c_R}]_t, [b_j^1]_t, \ldots, [b_j^{c_M+c_R}]_t$).

Agreement on a Core-Set: Code for P_i

1. Create an accumulative set $C^i = \emptyset$. 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, add P_j in C^i.
2. Take 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. Wait until ACS completes with output C containing $2t+1$ parties. For simplicity, assume that $C = \{P_1, \ldots, P_{2t+1}\}$.
2. For $k = 1, \ldots, c_M + c_R$, locally compute $c_i^k = \sum_{j=1}^{2t+1} r_j c_{ji}^k$ the i^{th} share of $c^k = r_1 c_1^k + \ldots + r_{2t+1} c_{2t+1}^k$, where $(r_1, \ldots, r_{2t+1})$ is the publicly known recombination vector.

Table 5: Protocol for Generating t-1D-sharing of $c_M + c_R$ secret random multiple triples

APPENDIX H: Protocol for Input Phase

Protocol **InputPhase**($\mathcal{P}$)

Secret Sharing: Code for P_i

1. On input X_i, invoke ACVSS-Share($P_i, \mathcal{P}, X_i$) to generate t-1D-sharing of X_i.
2. For every $j = 1, \ldots, n$, participate in ACVSS-Share($P_j, \mathcal{P}, X_j$).

Agreement on a Core-Set: Code for P_i

1. Create an accumulative set $C^i = \emptyset$. Upon completing ACVSS-Share($P_j, \mathcal{P}, x_j$) with dealer P_j, add P_j in C^i.
2. Participate in ACS with the accumulative set C^i as input.
3. Output core-set C containing $2t + 1$ parties and local shares of all inputs corresponding to parties in C.

Table 6: Protocol for Sharing Inputs, $n = 3t + 1$

APPENDIX I: Protocol for Computation Phase

Protocol **ComputationPhase**($\mathcal{P}$)

For every gate in the circuit: Code for P_i
Wait until the i^{th} share of each of the inputs of the gate is available. Now depending on the type of the gate, proceed as follows:

1. **Input Gate:** $[s]_t = \mathsf{IGate}([s]_t)$: There is nothing to be done here.
2. **Linear Gate:** $[z]_t = \mathsf{LGate}([x]_t, [y]_t, \ldots)$: Compute $z_i = LGate(x_i, y_i, \ldots)$, the i^{th} share of $z = \mathsf{LGate}(x, y, \ldots)$, where $x_i, y_i, \ldots$ denotes i^{th} share of $x, y, \ldots$.
3. **Multiplication Gate:** $[z]_t = \mathsf{MGate}([x]_t, [y]_t, ([a^k]_t, [b^k]_t, [c^k]_t))$:
   (a) Let $([a^k]_t, [b^k]_t, [c^k]_t)$ be the random triple associated with the multiplication gate.
   (b) Compute $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$, the i^{th} share of $\alpha = (x - a)$ and $\beta = (y - b)$ respectively.
   (c) Participate in ACVSS-Rec-Public($\mathcal{P}, \alpha, \mathcal{P}$) and ACVSS-Rec-Public($\mathcal{P}, \beta, \mathcal{P}$) to reconstruct α and β.
   (d) Upon completion of both the instances of ACVSS-Rec-Public, compute $z_i = \alpha\beta + \alpha b_i + \beta a_i + c_i$, the i^{th} share of $z = \alpha\beta + \alpha b + \beta a + c = xy$.
4. **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 random triple associated with the random gate. Compute $r_i = a_i^k$ as the i^{th} share of r.
5. **Output Gate:** $x = \mathsf{OGate}([x]_t)$: Participate in ACVSS-Rec-Public($\mathcal{P}, x, \mathcal{P}$) and output x.

Table 7: Protocol for Evaluating the Circuit, $n = 3t + 1$