Communication Optimal Multi-Valued Asynchronous Byzantine Agreement with Optimal Resilience

Arpita Patra[1] ⋆ and C. Pandu Rangan[2]

[1] Department of Computer Science, Aarhus University, Denmark
arpita@cs.au.dk,arpitapatra10@gmail.com
[2] Department of Computer Science and Engg., IIT Madras, India
prangan55@gmail.com

Abstract. Byzantine Agreement (BA) and Broadcast (BC) are considered to be the most fundamental primitives for fault-tolerant distributed computing and cryptographic protocols. An important variant of BA and BC is Asynchronous Byzantine Agreement (ABA) and Asynchronous Broadcast (called as A-cast) respectively. Most often in the literature, protocols for ABA and A-cast were designed for a single bit message. But in many applications, these protocols may be invoked on *long message* rather than on single bit. Therefore, it is important to design efficient *multi-valued* protocols (i.e. protocols with *long message*) which extract advantage of directly dealing with long messages and are far better than multiple invocations to existing protocols for single bit. In synchronous network settings, this line of research was initiated by Turpin and Coan [27] and later it is culminated in the result of Fitzi et al. [15] who presented the first ever *communication optimal* (*i.e. the communication complexity is minimal in asymptotic sense*) multi-valued BA and BC protocols with the help of BA and BC protocols for short message. It was left open in [15] to achieve the same in asynchronous settings.
In [21], the authors presented a communication optimal multi-valued A-cast using existing A-cast [6] for small message. Here we achieve the same for ABA which is known to be harder problem than A-cast. Specifically, we design a *communication optimal*, optimally resilient (*allows maximum fault tolerance*) multi-valued ABA protocol, based on the existing ABA protocol for short message.

Keywords: Asynchronous Byzantine Agreement, Multi-valued, Unbounded Computing Power.

1 Introduction

The problem of Byzantine Agreement (BA) (also popularly known as consensus) was introduced in [22] and since then it has emerged as the most fundamental problem in distributed computing. It has been used as a building block for several important secure distributed computing tasks such as Secure Multiparty

⋆ Financial Support from Center for Research in Foundations of Electronic Markets (CFEM) Denmark Acknowledged. The work was done when the author was a PhD student at IIT Madras, India.

Computation (MPC) [4, 5, 26], Verifiable Secret Sharing (VSS) [10, 4, 26] etc. In practice, BA is used in almost any task that involves multiple parties, like voting, bidding, secure function evaluation, threshold key generation etc [14]. Informally, a BA protocol allows a set of parties, each holding some input bit, to agree on a common bit, even though some of the parties may act maliciously in order to make the honest parties disagree.

An important variant of BA problem is asynchronous BA (known as ABA) that studies the BA problem in asynchronous network which is known to be more realistic than synchronous network. The works of [3, 25, 6, 11, 8, 7, 1, 23] have reported different ABA protocols. In this paper, we focus on ABA, specifically on the communication complexity of the problem.

Our Model. We follow the network model of [8, 7]. Specifically, our ABA protocol is carried out among a set of n parties, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, where every two parties are directly connected by a secure channel and t out of the n parties can be under the influence of a *computationally unbounded Byzantine (active) adversary*, denoted as $\mathcal{A}_t$. We assume $n = 3t + 1$ which is the minimum number of parties required to design any ABA protocol [17]. The adversary $\mathcal{A}_t$, completely dictates the parties under its control and can force them to deviate from the protocol in any arbitrary manner. The parties not under the influence of $\mathcal{A}_t$ are called *honest or uncorrupted.*

The underlying network is asynchronous, where the communication channels between the parties have arbitrary, yet finite delay (i.e. the messages are guaranteed to reach eventually). To model this, we assume that $\mathcal{A}_t$ controls the network and may delay messages between any two honest parties. However, it cannot read or modify these messages as the links are private and authenticated, and it also has to eventually deliver all the messages by honest parties. In asynchronous network, the inherent difficulty in designing a protocol comes from the fact that a party can not distinguish between a slow sender (whose message is simply delayed in the network) and a corrupted sender (who did not send the message at all). So a party can not wait for the values sent by all parties, as waiting for all of them may turn out to be endless. Hence the values of up to t (potentially honest) parties may have to be ignored for computation at any step.

Definitions. We now define ABA and its variant formally.

Definition 1 (ABA [8]). *Let Π be an asynchronous protocol executed among the set of parties $\mathcal{P}$, with each party having a private binary input. We say that Π is an ABA protocol tolerating $\mathcal{A}_t$ if the following hold, for every possible behavior of $\mathcal{A}_t$ and every possible input: (a)* **Termination***: All honest parties eventually terminate the protocol. (b)* **Correctness***: All honest parties who have terminated the protocol hold identical outputs. Furthermore, if all honest parties had same input, say ρ, then all honest parties output ρ.*

We now define (ϵ, δ)-ABA protocol, where both ϵ and δ are negligibly small values and are called error probabilities of the ABA protocol. Throughout our paper, we assume $\epsilon = 2^{-\Omega(\kappa)}$ and $\delta = 2^{-\Omega(\kappa)}$, where κ is called as the error parameter. To achieve the above bounds for error probabilities, our protocol will operate on finite Galois field $\mathbb{F} = GF(2^\kappa)$.

Definition 2 **((ϵ, δ)-ABA).** *An ABA protocol Π is called (ϵ, δ)-ABA if: (a) Π satisfies* **Termination** *described in Definition 1, except with an error probability of ϵ and (b) Conditioned on the event that every honest party terminates Π, protocol Π satisfies* **Correctness** *property described in Definition 1, except with error probability δ.*

The ABA and (ϵ, δ)-ABA can be executed for long messages and these type of protocols will be referred as *multi-valued* protocols. The important parameters of any ABA protocol are: (a) **Resilience**: It is the maximum number of corrupted parties (t) that the protocol can tolerate and still satisfy its properties; (b) **Communication Complexity**: It is the total number of bits communicated by the *honest* parties in the protocol; (c) **Computation Complexity:** It is the computational resources required by the honest parties during a protocol execution; and (d) **Running Time**: An informal, but standard definition of the running time of an asynchronous protocol is provided in [8, 7].

The History of ABA. From [22, 17], any ABA protocol tolerating $\mathcal{A}_t$ is possible if and only if $n \geq 3t + 1$. Thus any ABA protocol designed with $n = 3t+1$ parties is called as *optimally resilient.* By the seminal result of [13], any ABA protocol, irrespective of the value of n, must have some *non-terminating* runs/executions, where some honest party(ies) may not terminate at all. So in any (ϵ, δ)-ABA protocol with non-zero ϵ, the probability of the occurrence of a non-terminating execution is at most ϵ (these type of protocols are called $(1-\epsilon)$-terminating [8, 7]). On the other hand in any $(0, \delta)$-ABA protocol, the *probability* of the occurrence of a non-terminating execution is *asymptotically zero* (these type of protocols are called *almost-surely terminating*, a term coined in [1]). In Table 1, we summarize the best known ABA protocols in the literature.

Table 1. Summary of the Best Known Existing ABA Protocols

Ref.	Type	Resilience	Communication Complexity (CC) in bits	Expected Running Time (ERT)
[6]	$(0,0)$-ABA	$t < n/3$	$\mathcal{O}(2^n)$	$\mathcal{C} = \mathcal{O}(2^n)$
[11, 12]	$(0,0)$-ABA	$t < n/4$	$\mathcal{O}((nt + t^7) \log \lvert\mathbb{F}\rvert)$[a]	$\mathcal{C} = \mathcal{O}(1)$
[8, 7]	$(\epsilon,0)$-ABA	$t < n/3$	Private[b]: $\mathcal{O}(\mathcal{C}n^{11} (\log \kappa)^4)$[c] A-cast[d]: $\mathcal{O}(\mathcal{C}n^{11} (\log \kappa)^2 \log n)$	$\mathcal{C} = \mathcal{O}(1)$
[1]	$(0,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^6 \log \lvert\mathbb{F}\rvert)$ A-cast: $\mathcal{O}(\mathcal{C}n^6 \log \lvert\mathbb{F}\rvert)$	$\mathcal{C} = \mathcal{O}(n^2)$
[20]	$(\epsilon,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^6 \log \kappa)$ A-cast: $\mathcal{O}(\mathcal{C}n^6 \log \kappa)$	$\mathcal{C} = \mathcal{O}(1)$
[20]	Multi-valued[e] $(\epsilon,0)$-ABA	$t < n/3$	Private: $\mathcal{O}(\mathcal{C}n^5 \log \kappa)$ A-cast: $\mathcal{O}(\mathcal{C}n^5 \log \kappa)$	$\mathcal{C} = \mathcal{O}(1)$

[a] Here $\mathbb{F}$ is the finite field over which the ABA protocol of [11, 12] works. It is enough to have $|\mathbb{F}| \geq n$ and therefore $\log |\mathbb{F}|$ can be replaced by $\log n$. In fact in the remaining table, $\mathbb{F}$ bears the same meaning.
[b] Communication over private channels between pair of parties in $\mathcal{P}$.
[c] In this table, κ is the error parameter of the protocols.
[d] Total number of bits that needs to be A-casted (see more discussion on A-cast in subsection 2.1 under section 2).
[e] This protocol allows to reach agreement on $(t + 1)$ bits concurrently.

Multi-valued ABA. In many applications, ABA protocols are invoked on *long messages* rather than on single bit. For example, in asynchronous MPC (AMPC) [5, 7, 19], where typically lot of ABA invocations are required, many of the invocations can be parallelized and optimized to a single invocation with a long message. All existing protocols for ABA [25, 3, 6, 11, 12, 8, 7, 1, 20] are designed for single bit message. A naive approach to design multi-valued ABA for $\ell > 1$ bit message is to parallelize ℓ invocations of existing ABA protocols dealing with single bit. This approach requires a communication complexity that is ℓ times the communication complexity of the existing protocols for single bit and hence is not very efficient. More intelligent techniques need to be called for in order to gain in terms of communication complexity.

In synchronous network, Turpin and Coan [27] are the first to report a multi-valued BC protocol based on the access to a BC protocol for short message. Recently, Fitzi et al. [15] have designed *communication optimal* BA and BC protocols for large message using BA and BC protocol (respectively) for small message. While all existing synchronous BA protocols required a communication cost of $\Omega(\ell n^2)$ bits, the BA protocols of [15] communicate $\mathcal{O}(\ell n + poly(n, \kappa))$ bits to agree on an ℓ bit message. For a sufficiently large ℓ, the communication complexity expression reduces to $\mathcal{O}(n\ell)$, which is a clear improvement over $\Omega(\ell n^2)$. A brief discussion on the approach used in [15] for designing BA protocol is presented in **Appendix A**.

Designing *communication optimal*, multi-valued ABA and A-cast protocol was left as an interesting open question in [15]. The problem of A-cast has been resolved in [21]. In this paper, we settle the case for ABA which is known to be harder than A-cast. Our ABA calls for much more involved techniques than A-cast of [21]. To the best of our knowledge, ours is the first ever attempt to design multi-valued ABA.

Our Contribution. We propose a *communication optimal*, *optimally resilient*, *multi-valued* (ϵ, δ)-ABA protocol that attains a communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits to agree on an ℓ bit message. Our protocol requires $\mathcal{O}(n^3)$ invocations to ABA protocol for small messages (we may use any one of the ABA protocols listed in Table 1; the most communication efficient ABA is listed in the last row of the same table). For sufficiently large ℓ, the communication complexity of our protocol becomes $\mathcal{O}(\ell n)$ bits. From the result of [15], any BA protocol in synchronous networks with $t \in \Omega(n)$, requires to communicate $\Omega(n\ell)$ bits for an ℓ bit message. The same lower bound holds for asynchronous networks as well. Therefore our ABA is *communication optimal* for large enough ℓ. The degree of n and κ (and therefore the bound on ℓ for which our protocol is communication optimal) in the term $poly(n, \kappa)$ depends on the ABA for short message under use.

In our ABA protocol, we employ *player-elimination* framework introduced in [16] in the context of MPC. So far player-elimination was used only in MPC and AMPC. Hence our result shows the first non-MPC application of the technique. Apart from this, we present a novel idea to expand a set of $t + 1$ parties, with all the honest party(ies) in it holding a common message m, to a set of $2t + 1$

parties with all honest parties in it holding m. Moreover, the expansion process requires a communication complexity of $\mathcal{O}(\ell n + poly(n, \kappa))$ bits, where $|m| = \ell$. This technique may be useful in designing communication efficient protocols for many other form of consensus problems.

2 Communication Optimal (ϵ, δ)-ABA Protocol

We now present our novel (ϵ, δ)-ABA protocol with $n = 3t + 1$, called Optimal-ABA. The protocol allows the honest parties in $\mathcal{P}$, each having input message of ℓ bits, to reach agreement on a common message $m^* \in \{0,1\}^\ell$ containing ℓ bits. Moreover, if all the honest parties have same input m, then they agree on m at the end. We first describe the existing tools used in Optimal-ABA.

2.1 Tools Used

Hash Function [15, 9]. A *keyed* hash function $\mathcal{U}_\kappa$ maps arbitrary strings in $\{0,1\}^*$ to κ bit string with the help of a κ bit random key. So $\mathcal{U}_\kappa : \{0,1\}^* \rightarrow \{0,1\}^\kappa$. $\mathcal{U}_\kappa$ can be implemented as follows: Let m and r be the input to $\mathcal{U}_\kappa$, where m is an ℓ bit string that need to be hashed/mapped and r is the hash key selected from $\mathbb{F}$. Without loss of generality, we assume that $\ell = \text{poly}(\kappa)$. Then m is interpreted as a polynomial $f_m(x)$ over $\mathbb{F}$, where the degree of $f_m(x)$ is $\lceil \ell/\kappa \rceil - 1$. For this, m is divided into blocks of κ bits and each block of κ bits is interpreted as an element from $\mathbb{F}$. Then these field elements are considered as the coefficients of $f_m(x)$ over $\mathbb{F}$. Finally, $\mathcal{U}_\kappa(m, r) = f_m(r)$. We now have the following important well-known theorem.

Theorem 1 (Collision Theorem [15]). *Let m_1 and m_2 be two ℓ bit messages. The probability that $\mathcal{U}_\kappa(m_1, r) = \mathcal{U}_\kappa(m_2, r)$ for a randomly chosen hash key r is $\frac{\ell 2^{-\kappa}}{\kappa} = 2^{-\Omega(\kappa)}$ which is negligible.*

A-cast or Asynchronous Broadcast. In brief, an A-cast protocol allows a sender $S \in \mathcal{P}$ to send some message M identically to all the parties in $\mathcal{P}$. An A-cast protocol satisfies two properties: (1) **Termination:** If S is honest, then all honest parties in $\mathcal{P}$ will eventually terminate; If any honest party terminates, then all honest parties will eventually terminate. (2) **Correctness:** If the honest parties terminate, then they do so with a common output M^*; (b) Furthermore, if the sender S is honest then $M^* = M$. The first ever protocol for A-cast is due to Bracha [6] and the protocol is error free in both termination and correctness. The A-cast protocol of [6] is t resilient with $t < n/3$ and communicates $\mathcal{O}(n^2)$ bits to A-cast a *single bit* in constant running time. The other protocol for A-cast is reported in [21]. This protocol has error in both correctness and termination; but it communicates $\mathcal{O}(\ell n)$ bits for an ℓ bit input where $\ell = \omega(n^2(n \log n + \kappa))$. For simplicity, in our ABA protocol we will prefer to use A-cast of [6]. We use the following syntax to invoke A-cast: A-cast$(S, \mathcal{P}, M)$. The description of Bracha's A-cast protocol is available in [7].

Notation 1 (Convention for Using A-cast:) *By saying that 'P_i A-casts M', we mean that P_i as a sender, initiates A-cast$(P_i, \mathcal{P}, M)$. Similarly 'P_j receives M from the A-cast of P_i' will mean that P_j terminates A-cast$(P_i, \mathcal{P}, M)$, with*

M as the output. By the property of A-cast*, if some honest party P_j terminates* A-cast$(P_i, \mathcal{P}, M)$ *with M as the output, then every other honest party will eventually do so, irrespective of the behavior of the sender P_i.*

Asynchronous Verifiable Secret Sharing (AVSS). An AVSS scheme consisting of two phases, namely sharing phase and reconstruction phase, can be viewed as a *distributed commitment mechanism* where a (possibly corrupted) special party in $\mathcal{P}$, called *dealer* (denoted as D), *commits* a secret $s \in \mathbb{F}$ in the sharing phase, where commitment information is distributed among the parties in $\mathcal{P}$. Later in reconstruction phase, the commitment s can be *uniquely* and privately reconstructed by any specific party, say $P_\alpha \in \mathcal{P}$ (we may call it as P_α-private-reconstruction) even in the presence of $\mathcal{A}_t$. Here P_α is called *receiver* party. Moreover, if D and P_α are honest, then secrecy of s from $\mathcal{A}_t$ is maintained throughout. AVSS is implemented by a pair of protocols (Sh, Rec) and it has three properties called, **Termination**, **Correctness** and **Secrecy** (for a formal definition of AVSS, refer to [7, 19]).

If an AVSS satisfies its **Termination** and/or **Correctness**, except with error probability $\rho = 2^{-\Omega(\kappa)}$, then we arrive at the notion of statistical AVSS. From [8], statistical AVSS tolerating $\mathcal{A}_t$ is possible iff $n \geq 3t + 1$. The best known communication efficient statistical AVSS with $n = 3t + 1$ is due to [19]. We use the following syntax to invoke AVSS: AVSS-Share$(D, \mathcal{P}, s)$ (protocol for sharing phase) and AVSS-Rec$(D, \mathcal{P}, s, P_\alpha)$ (protocol for reconstruction phase).

Agreement on a Common Subset (ACS). In our (ϵ, δ)-ABA protocol, we come across the following situation: There exists a set of parties $\mathcal{R} \subseteq \mathcal{P}$ with $|\mathcal{R}| \geq t + 1$, such that each party in $\mathcal{R}$ is asked to A-cast (AVSS-Share) some value(s). While the honest parties in $\mathcal{R}$ will eventually do the A-cast (AVSS-Share), the corrupted parties in $\mathcal{R}$ may or may not do the same. So the (honest) parties in $\mathcal{P}$ want to agree on a *common* set $\mathcal{T} \subset \mathcal{R}$, with $1 \leq |\mathcal{T}| \leq |\mathcal{R}| - t$, such that A-cast (AVSS-Share) instance of each party in $\mathcal{T}$ will be eventually terminated by the (honest) parties in $\mathcal{P}$. For this, the parties use ACS primitive (stands for Agreement on Common Subset), presented in [5]. The ACS protocol will use $|\mathcal{R}|$ instances of ABA invoked on single bit. We may use the best known communication efficient $(\epsilon, 0)$-ABA of [20] for this purpose. We use the following syntax for invoking ACS: ACS$(\mathcal{R}, |\mathcal{T}|)$.

Theorem 2. *The communication complexity of ACS is equal to $|\mathcal{R}|$ executions of ABA protocol each invoked on a single bit.*

2.2 Protocol Optimal-ABA

Our protocol Optimal-ABA uses the so-called *player-elimination framework*, along with several novel ideas. So far player-elimination [16] has been used *only* in the context of synchronous and asynchronous MPC [16, 2, 24]. Ours is the first non-MPC application of player-elimination. We would refer it by *party-elimination*, rather than player-elimination in our context (as we use the term party in place of player). In the party-elimination framework, the computation of Optimal-ABA is divided into t segments, where in each segment the parties agree on an $\frac{\ell}{t}$ bit, considering $\frac{\ell}{t}$ bits of their original input as the *input message* of the segment. In particular, the parties divide their original message into t blocks, each of size $\frac{\ell}{t}$ bits and in α^{th} segment $\mathcal{S}_\alpha$, the parties reach agreement on an $\frac{\ell}{t}$ bit message, considering *only* the α^{th} block as the input message. Each segment terminates eventually with the parties having common output of $\frac{\ell}{t}$ bits; moreover if the honest parties start a segment with the same block of $\frac{\ell}{t}$ bits, then they agree on that common input.

The computation of a segment is carried out in a *non-robust* fashion, in the sense that if all the parties including the corrupted parties behave according to the protocol then the segment successfully achieves its task; otherwise the segment may fail in which case it outputs a *triplet of parties* among which *at least one is corrupted*. In the former case, the next segment will be taken up for computation for reaching agreement with next block of $\frac{\ell}{t}$ bits as input. In the latter case, the same segment will be repeated among the set of parties after excluding the parties in the triplet and this continues until the segment becomes successful. It is to be noted that though the computations in a segment may be done among a *subset* of parties from $\mathcal{P}$ (as parties in triplet might be eliminated from $\mathcal{P}$), the agreement in the segment is finally attained over all honest parties in $\mathcal{P}$. It is now easy to see that the t segments may fail at most t times *in total* as t is the upper bound on the number of corrupted parties. After t failures, all the corrupted parties will be removed and therefore there will be no more failure.

We denote the input of party P_i by $m_i \in \{0,1\}^\ell$, which is divided into t blocks, with α^{th} block being denoted by $m_{i\alpha}$, for $\alpha = 1, \ldots, t$. At the beginning of our protocol, we initialize two dynamic variables $n' = n$ and $t' = t$ and one dynamic set $\mathcal{P}' = \mathcal{P}$. $\mathcal{P}'$ denotes the *set of non-eliminated* parties and contains n' parties, out of which at most t' can be corrupted. In every segment $\mathcal{S}_\alpha$ the computation is structured into three main phases: (a) **Checking Phase**, (b) **Expansion Phase** and (c) **Output Phase**. The segment failure may occur *only in the second phase* and hence *only* the first two phases of a segment may be repeated several times (bounded by t); once the first two phases are successful for a segment, the segment will always be successfully completed after robustly executing the third phase. So at the end of segment $\mathcal{S}_\alpha$, every honest party will agree on a common $\frac{\ell}{t}$ bits, denoted by m^*_α. Moreover if the honest parties start with common input (i.e. $m_{i\alpha}$'s are equal for all honest parties), then m^*_α will be same as that common input.

1. **Checking Phase:** Here the parties, on having private input message of $\frac{\ell}{t}$ bits each (i.e. $m_{i\alpha}$'s), jointly perform some computation in order to determine and agree on a set of $t' + 1$ parties called $\mathcal{P}'_{ch} \subseteq \mathcal{P}'$, such that the honest parties in $\mathcal{P}'_{ch}$ hold a common ℓ/t bit message, say m^*_α. In case of failure due to the inconsistencies among the inputs of the honest parties, the parties abort any further computation for current segment and agree on a predefined message $m^\dagger_\alpha$. So in this case current segment terminates with all honest parties agreeing on common output $m^*_\alpha = m^\dagger_\alpha$. On the other hand, if $\mathcal{P}'_{ch}$ is generated and agreed among the parties, then the computation for current

segment proceeds to the next phase. It is to be noted that $\mathcal{P}'_{ch}$ will be *always obtained* if the initial messages of the honest parties in $\mathcal{P}'$ are same.

2. **Expansion Phase:** Here the parties in $\mathcal{P}'_{ch}$ on holding a common message m^*_α help other parties to receive m^*_α. Specifically here the parties jointly perform some computation in conjunction with the parties in $\mathcal{P}'_{ch}$ to expand $\mathcal{P}'_{ch}$ to a set of $2t'+1$ parties, denoted by $\mathcal{P}'_{ex}$ (with $\mathcal{P}'_{ch} \subset \mathcal{P}'_{ex} \subseteq \mathcal{P}'$) such that all honest parties in $\mathcal{P}'_{ex}$ hold m^*_α. *The expansion technique is the most crucial and novel part of our protocol.* But the computation of this phase is non-robust and hence either one of the following is guaranteed: (a) $\mathcal{P}'_{ex}$ is constructed successfully or (b) a triplet of parties (P_i, P_j, P_k) is obtained, such that at least one of the three parties is corrupted. If the former case happens, then parties proceed to execute **Output Phase**. If the latter case happens, then n' and t' are reduced by 3 and 1 respectively and the current segment is repeated from the beginning (from the **Checking Phase**) with updated n' and t' and $\mathcal{P}' = \mathcal{P}' \setminus \{P_i, P_j, P_k\}$. Note that n', t' and $\mathcal{P}'$ always satisfy: $n' = 3t'+1$ and $|\mathcal{P}'| = n'$.
3. **Output Phase:** Here the parties in $\mathcal{P}'_{ex}$ help the parties in $\mathcal{P} \setminus \mathcal{P}'_{ex}$ (**not** $\mathcal{P}' \setminus \mathcal{P}'_{ex}$) to learn the common ℓ/t message m^*_α held by the honest parties in $\mathcal{P}'_{ex}$. After this phase, current segment terminates with common output m^*_α and the parties proceed to the computation of next segment. The implementation of this phase is very similar to the implementation of the Output Phase of [21] and the Claiming Stage of the BA protocol of [15].

Now the overall structure of Optimal-ABA is presented below.

Protocol Optimal-ABA($\mathcal{P}$)

Code for P_i: Every party in $\mathcal{P}$ executes this code.

1. Set $n' = n$, $t' = t$ and $\mathcal{P}' = \mathcal{P}$. Initialize $\alpha = 1$.
2. While $\alpha \leq t$, do the following for segment $\mathcal{S}_\alpha$ with input $m_{i\alpha}$ and with n', t' and $\mathcal{P}'$ to agree on m^*_α:
   (a) **Checking Phase:** Participate in the code Checking, presented in Fig. 1 to determine and agree on $\mathcal{P}'_{ch} \subseteq \mathcal{P}'$ of size $t'+1$ such that all the honest parties in $\mathcal{P}'_{ch}$ hold common $\frac{\ell}{t}$ bits, say m^*_α. If $\mathcal{P}'_{ch}$ is generated then proceed to the next phase. Otherwise set m^*_α to some predefined value $m^\dagger_\alpha \in \{0,1\}^{\frac{\ell}{t}}$, set $\alpha = \alpha + 1$ and terminate the current segment with output m^*_α.
   (b) **Expansion Phase:** Participate in code Expansion presented in Fig. 2 to expand $\mathcal{P}'_{ch}$ to contain $2t'+1$ parties, denoted by $\mathcal{P}'_{ex}$ such that $\mathcal{P}'_{ch} \subset \mathcal{P}'_{ex} \subseteq \mathcal{P}'$ and all honest parties in $\mathcal{P}'_{ex}$ hold m^*_α. If $\mathcal{P}'_{ex}$ is generated successfully then proceed to the next phase. Otherwise output a triplet (P_m, P_l, P_k), set $n' = n'-3$, $t' = t'-1$ and $\mathcal{P}' = \mathcal{P}' \setminus \{P_m, P_l, P_k\}$ and repeat the current segment.
   (c) **Output Phase:** Participate in code Output presented in Fig. 3 and output m^*_α upon termination, set $\alpha = \alpha + 1$ and terminate the current segment.
3. Output m^* which is the concatenation of $m^*_1, \ldots, m^*_t$ and terminate the protocol.

In the sequel, we will pursue an in-depth discussion on the implementation and properties of each of the above three phases.

Checking Phase. As mentioned before, the aim of this phase is to either agree on a set $\mathcal{P}'_{ch}$ of size $t'+1$ such that all the honest parties in $\mathcal{P}'_{ch}$ hold common message, say m^*_α, or decide that such set may not exist. When all the honest parties start with same input message, $\mathcal{P}'_{ch}$ can be always found out and agreed upon. To achieve the above task, every party hashes his message with a random key and A-casts the (random key, hash value) pair. The parties then agree on a set $\mathcal{I}$ of $n'-t'$ parties whose A-cast will be eventually received by every honest party. This can be achieved by executing an instance of ACS.

Now every party P_i prepares a response vector $\overrightarrow{v_i}$, indicating whether the hash value of every $P_j \in \mathcal{I}$ is indeed the hash value of his own message $m_{i\alpha}$ with respect to P_j's hash key (this should ideally be the case, when P_i and P_j are honest and their input messages are identical, i.e. $m_{i\alpha} = m_{j\alpha}$). P_i A-casts $\overrightarrow{v_i}$. Now the parties again agree on a set of $n'-t'$ parties, say $\mathcal{J}$ whose A-cast with their $\overrightarrow{v_i}$ has been terminated. *Now notice that if all honest parties start with common input, then the vectors of the honest parties in $\mathcal{J}$ would be identical and would have at least $t'+1$ 1's at the locations corresponding to the $t'+1$ honest parties in $\mathcal{I}$.* So now the parties try to find a set of at least $t'+1$ parties in $\mathcal{J}$, whose vectors are identical and have *at least $t'+1$ 1's in them.* If found, then any subset of $t'+1$ parties from that set (say $t'+1$ parties with smallest index) will be considered as $\mathcal{P}'_{ch}$. It is easy to show that $\mathcal{P}'_{ch}$ will be *always obtained* if the initial messages of the honest parties in $\mathcal{P}'$ are same. Moreover it can be shown that the honest parties in $\mathcal{P}'_{ch}$ hold common message, say m^*_α with very high probability (see Lemma 2). But if $\mathcal{P}'_{ch}$ is not found, then the honest parties know that their input messages are inconsistent and hence they agree that such set can not be found. The steps performed so far are enough for achieving the goal of our current phase.

But we need to do some more task for the requirement of next phase i.e. **Expansion Phase**. In **Expansion Phase**, we require that every honest party P_j in $\mathcal{P}'$ should hold a distinct *secret* random hash key and hash value of the message corresponding to every party P_i in $\mathcal{I}$, such that the hash key and hash value that P_j has received from P_i should not be known to anybody other than P_i and P_j. Though achieving this in synchronous network is easy, it needs some amount of effort in asynchronous network. We do this by using AVSS-Share and AVSS-Rec. The code that implements this phase is now given in Fig. 1.

Before proving the properties of **Checking Phase**, we define the following:

<u>Event E</u>: *Let E be an event in an execution of Checking, defined as follows: All invocations of AVSS scheme initiated by the parties in $\mathcal{I}$ have been terminated with correct output. More clearly, E means that all the invocations of AVSS initiated by the parties in $\mathcal{I}$ will satisfy termination and correctness property. It is easy to see that E occurs with very high probability of* $(1-2^{-\Omega(\kappa)})$. □

In the sequel, all the lemmas for all the three phases are proved conditioned on event E.

Lemma 1 (Termination of Checking Phase). *In a segment $\mathcal{S}_\alpha$, an execution of* **Checking Phase** *will be terminated, except with probability* $2^{-\Omega(\kappa)}$,

Fig. 1. Code for Checking Phase.

Checking

To avoid notational clutter, we assume that $\mathcal{P}'$ is the set of first n' parties

Code for $P_i \in \mathcal{P}'$: Every party in $\mathcal{P}'$ executes this code

1. On having input $m_{i\alpha}$,
   (a) choose a random hash key r_i from $\mathbb{F}$ and A-cast $(r_i, \mathcal{V}_i)$ where $\mathcal{V}_i = \mathcal{U}_\kappa(m_{i\alpha}, r_i)$;
   (b) choose n' random hash keys $r_{i1}, \ldots, r_{in'}$ from $\mathbb{F}$ and commit $(r_{ij}, \mathcal{V}_{ij})$ where $\mathcal{V}_{ij} = \mathcal{U}_\kappa(m_{i\alpha}, r_{ij})$, by executing AVSS-Share$(P_i, \mathcal{P}', r_{ij})$ and AVSS-Share$(P_i, \mathcal{P}', \mathcal{V}_{ij})$.
2. Participate in AVSS-Share$(P_j, \mathcal{P}', r_{jk})$ and AVSS-Share$(P_j, \mathcal{P}', \mathcal{V}_{jk})$ for every $P_j \in \mathcal{P}'$ and $k = 1, \ldots, n'$.
3. Participate in ACS$(\mathcal{P}', n' - t')$ to agree on a set of $n' - t'$ parties from $\mathcal{P}'$, denoted as $\mathcal{I}$, whose A-cast and the $2n'$ instances of AVSS-Share will be eventually terminated (by the honest parties in $\mathcal{P}'$).
4. Wait to receive $(r_j, \mathcal{V}_j)$ from the A-cast of every $P_j \in \mathcal{I}$.
5. Wait to terminate all $2n'$ instances of AVSS-Share of every party in $\mathcal{I}$. Participate in AVSS-Rec$(P_j, \mathcal{P}', r_{jk}, P_k)$ and AVSS-Rec$(P_j, \mathcal{P}', \mathcal{V}_{jk}, P_k)$ for every $P_j \in \mathcal{I}$ and every $P_k \in \mathcal{P}'$ for P_k-private-reconstruction of $(r_{jk}, \mathcal{V}_{jk})$.
6. Obtain $(r_{ji}, \mathcal{V}_{ji})$ pair from AVSS-Rec$(P_j, \mathcal{P}', r_{ji}, P_i)$ and AVSS-Rec$(P_j, \mathcal{P}', \mathcal{V}_{ji}, P_i)$ corresponding to every $P_j \in \mathcal{I}$.
7. Construct n length vector $\overrightarrow{v_i}$, where $\overrightarrow{v_i}[j] = \begin{cases} \perp & \text{If } P_j \notin \mathcal{I} \\ 1 & \text{If } P_j \in \mathcal{I} \text{ and } \mathcal{V}_j = \mathcal{U}_\kappa(m_{i\alpha}, r_j). \\ 0 & \text{If } P_j \in \mathcal{I} \text{ and } \mathcal{V}_j \neq \mathcal{U}_\kappa(m_{i\alpha}, r_j). \end{cases}$
   A-cast $\overrightarrow{v_i}$.
8. Participate in ACS$(\mathcal{P}', n' - t')$ to agree on a set of $n' - t'$ parties from $\mathcal{P}'$, denoted as $\mathcal{J}$, whose A-cast with an n length vector has been terminated.
9. Check whether there is a unique set of at least $t' + 1$ parties in $\mathcal{J}$ such that their vectors are identical and have at least $t' + 1$ 1's in them (Note that this can be done in polynomial time).
   (a) If yes, then let $\mathcal{P}'_{ch}$ be the set containing exactly $t' + 1$ parties (say the parties with first $t' + 1$ smallest indices) out of those parties. Let $\overrightarrow{v}$ be the n length vector, where $\overrightarrow{v}[i] = 1$ if the i^{th} location of the vectors of all parties in $\mathcal{P}'_{ch}$ is 1, otherwise $\overrightarrow{v}[i] = \perp$. Moreover, let $\mathcal{I}_1 = \{P_i \in \mathcal{I} \text{ such that } \overrightarrow{v}[i] = 1\}$. Assign $m^*_\alpha = m_{i\alpha}$ if $P_i \in \mathcal{P}'_{ch}$.
   (b) If not, then decide that $\mathcal{P}'_{ch}$ can not be found.

where termination means that the code either outputs a set $\mathcal{P}'_{ch}$ of size $t' + 1$ or decide that such set can not be constructed.

Proof: Conditioned on event E, an execution of **Checking Phase** will always terminate if both the executions of ACS terminate and all the instances of A-cast terminate. Since A-cast has no error in termination and each execution of ACS terminates except with probability $2^{-\Omega(\kappa)}$, an execution of **Checking Phase** will terminate except with negligible probability. □

Lemma 2 (Correctness of Checking Phase). *In an execution of* **Checking Phase** *in a segment $\mathcal{S}_\alpha$, the honest parties in $\mathcal{P}'_{ch}$ (if it is found) hold a common message m^*_α, except with probability $2^{-\Omega(\kappa)}$. Moreover, if the honest parties start $\mathcal{S}_\alpha$ with common message m_α, then $\mathcal{P}'_{ch}$ will always be found with $m^*_\alpha = m_\alpha$.*

Proof: We prove the first part of the lemma. If $\mathcal{P}'_{ch}$ contains exactly one honest party, then first part is trivially true with m^*_α being the input message of the sole honest party in $\mathcal{P}'_{ch}$. So let $\mathcal{P}'_{ch}$ contain at least two honest parties. We now show that the messages of every pair of honest parties (P_i, P_j) in $\mathcal{P}'_{ch}$ are same. Recall that the response vectors $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$ of P_i and P_j are identical and have at least $t' + 1$ 1's in them. Moreover, $\mathcal{I}_1$ contains all P_k's such that $\overrightarrow{v_i}[k] = \overrightarrow{v_j}[k] = 1$. Evidently, $|\mathcal{I}_1| \geq t + 1$. So there is at least one *honest* party in $\mathcal{I}_1$, say P_k, such that $\overrightarrow{v_i}[k] = \overrightarrow{v_j}[k] = 1$. This implies that $\mathcal{V}_k = \mathcal{U}_\kappa(m_{i\alpha}, r_k)$ and $\mathcal{V}_k = \mathcal{U}_\kappa(m_{j\alpha}, r_k)$ holds for P_i and P_j respectively, where P_i has received $(\mathcal{V}_k, r_k)$ from P_k (by A-cast) and P_j has received $(\mathcal{V}_k, r_k)$ from P_k (by A-cast). Now by **Collision Theorem** (Theorem 1), it follows that $m_{i\alpha} = m_{k\alpha}$ and $m_{j\alpha} = m_{k\alpha}$, except with probability $2^{-\Omega(\kappa)}$. Consequently $m_{i\alpha} = m_{j\alpha}$, except with probability $2^{-\Omega(\kappa)}$. Now let us fix an honest party, say P_i in $\mathcal{P}'_{ch}$. If P_i's value is equal to every honest P_j's value in $\mathcal{P}'_{ch}$, then it means that all honest parties in $\mathcal{P}'_{ch}$ hold a common message m^*_α, except with negligible error probability.

We now prove the second part. When all honest parties start with same input m_α, the vectors of all honest parties in $\mathcal{J}$ will have 1 at the locations corresponding to the honest parties in $\mathcal{I}$. Since there are at least $t' + 1$ honest parties in both $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{P}'_{ch}$ can always be found and now it is easy to see that all honest parties in $\mathcal{P}'_{ch}$ will hold m_α. □

Expansion Phase. If $\mathcal{P}'_{ch}$ is found and agreed upon in previous phase, then the parties proceed to expand $\mathcal{P}'_{ch}$ in order to obtain $\mathcal{P}'_{ex}$. For that we first initiate $\mathcal{K} = \mathcal{P}'_{ch}$ and $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$. Then $\mathcal{K}$ will be expanded to contain $2t' + 1$ parties and we will assign $\mathcal{K}$ to $\mathcal{P}'_{ex}$ when $\mathcal{K}$ contains $2t' + 1$ parties. We call the $\mathcal{K}$ containing $t' + 1$ parties as 'initial' $\mathcal{K}$ and likewise the $\mathcal{K}$ containing $2t' + 1$ parties as 'final' $\mathcal{K}$. The expansion (transition from 'initial' $\mathcal{K}$ to 'final' $\mathcal{K}$) takes place in a sequence of t' iterations. In each iteration, either $\mathcal{K}$ is expanded by one or in case of failure a conflict triplet is returned. In the latter case, the current segment fails and it is again repeated (from **checking phase**) with renewed value of n', t' and $\mathcal{P}'$ (i.e. after excluding the parties in the triplet from $\mathcal{P}'$).

So this phase starts as follows: First an injective mapping $\varphi : \mathcal{K} \to \overline{\mathcal{K}}$ is defined. Now a party $P_i \in \mathcal{K}$ sends his message m^*_α to party $\varphi(P_i) \in \overline{\mathcal{K}}$. A party $P_i \in \overline{\mathcal{K}}$ on receiving a message m^*_α from $\varphi^{-1}(P_i) \in \mathcal{K}$, calculates vector $\overrightarrow{v_i}$ with the (key, hash value) pair of the parties only in $\mathcal{I}_1$ ($\perp$ is placed at all other locations) and with m^*_α as the message. P_i then A-casts Matched-P_i if $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ (which was calculated in Checking). Otherwise let k be the minimum index in $\overrightarrow{v_i}$ such that $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then P_i A-casts a conflict triplet $(\varphi^{-1}(P_i), P_i, P_k)$. Clearly, one of the three parties in the triplet must be corrupted. The parties now invoke an instance of ACS to agree on a single party, say P_l from $\overline{\mathcal{K}}$ whose A-cast has been terminated. Such a party from $\overline{\mathcal{K}}$

can always be found as there exists at least one honest $P_m \in \mathcal{K}$ which will be mapped to another honest $P_l = \varphi(P_m) \in \overline{\mathcal{K}}$ and P_l will eventually receive m^*_α from P_m and successfully A-cast some message (see Lemma 4).

Now there are two cases. If $(\varphi^{-1}(P_l), P_l, P_k)$ is received from the A-cast of P_l, then the computation stops here and the triplet $(\varphi^{-1}(P_l), P_l, P_k)$ is returned. If Matched-P_l is received from the A-cast of P_l, then P_l is included in $\mathcal{K}$ and excluded from $\overline{\mathcal{K}}$. P_l now finds a unique party from the set of parties in $\overline{\mathcal{K}}$ that was never mapped before (say the unmapped party with smallest index) and sends m^*_α to it. Again the party who receives the message, calculates response vector with the received message and A-casts either a conflict triplet or Matched signal. Then the parties invoke an instance of ACS to agree on a single party from $\overline{\mathcal{K}}$ whose A-cast has been terminated and this process continues until either $|\mathcal{K}|$ becomes $2t' + 1$ or the segment is failed with some triplet in some iteration. Though it is non-intuitive that in every iteration the parties will be able to agree on a single party from $\overline{\mathcal{K}}$ by executing ACS, this will indeed happen and we prove this in Lemma 4. If $\mathcal{K}$ becomes of size $2t' + 1$, it is assigned to $\mathcal{P}'_{ex}$. The code for this phase is given in Fig. 2. We now prove the properties of **Expansion Phase**.

Lemma 3. *In a segment $\mathcal{S}_\alpha$, in any iteration of* `while` *loop (in an execution of* **Expansion Phase***), no two different parties in $\mathcal{K}$ are mapped to the same party in $\overline{\mathcal{K}}$. Also in case* `while` *loop is completed with $\mathcal{K}$ containing $2t' + 1$ parties, only the last entrant in 'final' $\mathcal{K}$ is not mapped to any party.*

Proof: From the protocol steps, it is clear that a party in $\mathcal{K}$ is mapped only once. Now we show that no pair (P_i, P_j) in $\mathcal{K}$ is mapped to same party. This is true as φ is injective and also every time a party P_i from $\mathcal{K}$ is mapped to a party P_k in $\overline{\mathcal{M}}$ (set of unmapped parties), P_k is never mapped again as it is immediately transferred to $\mathcal{M}$ (set of mapped parties).

Now we show that there will be enough number of parties in $\overline{\mathcal{M}}$ to be mapped in all iterations, except the last one. We consider the worst case, where the while loop is executed completely for t' iterations (as 'initial' $|\mathcal{K}|$ is $t' + 1$ and t' more parties have to enter to make 'final' $\mathcal{K}$ of size $2t' + 1$), without outputting any triplet. Now as per the protocol, at the beginning of the `while` loop, $\mathcal{K} = t' + 1$, $\overline{\mathcal{K}} = 2t'$, $\mathcal{M} = t' + 1$ and $\overline{\mathcal{M}} = 2t' - (t' + 1) = t' - 1$. In i^{th} iteration, a party, say P_l from $\mathcal{M}$ (hence from $\overline{\mathcal{K}}$) enters into $\mathcal{K}$ and gets mapped to an unmapped party in $\overline{\mathcal{M}}$ (hence in $\overline{\mathcal{K}}$). As a result: (a) $|\mathcal{K}|$ increases by 1, (b) $|\overline{\mathcal{K}}|$ decreases by 1, (c) $|\mathcal{M}|$ remains same and (d) $|\overline{\mathcal{M}}|$ decreases by 1. So after $t' - 1$ iterations, the following hold: (a) $|\mathcal{K}| = 2t'$, (b) $|\overline{\mathcal{K}}| = t' + 1$, (c) $|\mathcal{M}| = t' + 1$ and (d) $|\overline{\mathcal{M}}| = 0$. Hence $\overline{\mathcal{M}}$ becomes empty only after the mapping is done in $(t' - 1)^{th}$ iteration. In the last iteration (t'^{th}), another party from $\mathcal{M}$ (hence from $\overline{\mathcal{K}}$) is finally included in $\mathcal{K}$ which need not be mapped to any more party as $\mathcal{K}$ becomes exactly $2t' + 1$ at this point. □

Lemma 4. *In a particular execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, $|\mathcal{K}|$ will increase by one with probability at least $(1 - 2^{-\Omega(\kappa)})$, in every iteration of* `while` *loop until the* `while` *loop is completed due to $|\mathcal{K}| = 2t' + 1$ or broken due to the output of a triplet.*

Fig. 2. Code for the Expansion Phase.

```
Expansion
Code for $P_i \in \mathcal{P}'$: Every party in $\mathcal{P}'$ executes this code

1. Assign $\mathcal{K} = \mathcal{P}'_{ch}$ and $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$.
2. Define an injective mapping $\varphi : \mathcal{K} \rightarrow \overline{\mathcal{K}}$ where $\overline{\mathcal{K}} = \mathcal{P}' \setminus \mathcal{K}$ as follows: the party
   with smallest index in $\mathcal{K}$ is associated with the party with smallest index in $\overline{\mathcal{K}}$. Let
   $\mathcal{M} = \varphi(\mathcal{K})$ ($\subset \overline{\mathcal{K}}$, as $|\mathcal{K}|$ is exactly $t'+1$) be the set of currently *mapped* parties in
   $\overline{\mathcal{K}}$. Let $\overline{\mathcal{M}} = \overline{\mathcal{K}} \setminus \mathcal{M}$ be the set of currently *unmapped* partied in $\overline{\mathcal{K}}$.
3. If $P_i \in \mathcal{K}$, then send $m^*_\alpha$ to $\varphi(P_i)$.
4. If $P_i \in \overline{\mathcal{K}}$ and has received message $m^*_\alpha$ from $\varphi^{-1}(P_i) \in \mathcal{K}$, then calculate vector $\overrightarrow{v_i}$
   of length $n$ as follows: $\overrightarrow{v_i}[j] = \begin{cases} \perp & \text{If } P_j \notin \mathcal{I}_1 \\ 1 & \text{If } P_j \in \mathcal{I}_1 \text{ and } \mathcal{V}_{ji} = \mathcal{U}_\kappa(m^*_\alpha, r_{ji}). \\ 0 & \text{If } P_j \in \mathcal{I}_1 \text{ and } \mathcal{V}_{ji} \neq \mathcal{U}_\kappa(m_\alpha, r_{ji}). \end{cases}$ Recall that
   $(r_{ji}, \mathcal{V}_{ji})$ pair was obtained by $P_i$ in Checking from AVSS-Rec($P_j, \mathcal{P}', r_{ji}, P_i$) and
   AVSS-Rec($P_j, \mathcal{P}', \mathcal{V}_{ji}, P_i$). If $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ then A-cast Matched-$P_i$; otherwise
   let $k$ be the minimum index in $\overrightarrow{v_i}$ such that $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then A-cast $(P_j, P_i, P_k)$,
   where $P_j = \varphi^{-1}(P_i)$.
5. while $|\mathcal{K}| < 2t'+1$ do:
   (a) Participate in an instance of ACS($\mathcal{M}, 1$) to agree on a single party from $\mathcal{M}$
       whose A-cast has been terminated. Let the party be $P_l$.
   (b) If $(P_m, P_l, P_k)$ is received from A-cast of $P_l$, then stop any further computation
       and output the triplet $(P_m, P_l, P_k)$.
   (c) If Matched-$P_l$ is received from A-cast of $P_l$, then set $\mathcal{K} = \mathcal{K} \cup \{P_l\}$, $\overline{\mathcal{K}} = \overline{\mathcal{K}} \setminus \{P_l\}$
       and $\mathcal{M} = \mathcal{M} \setminus \{P_l\}$.
   (d) Define a mapping, which maps $P_l$ to the party in $\overline{\mathcal{M}}$ with the smallest index,
       say $P_m$. Set $\overline{\mathcal{M}} = \overline{\mathcal{M}} \setminus \{P_m\}$ and $\mathcal{M} = \mathcal{M} \cup \{P_m\}$.
   (e) If $P_i = P_l$, then send $m^*_\alpha$ to $P_m$.
   (f) If $P_i = P_m$ and $P_i$ has received message $m^*_\alpha$ from $P_l$, then calculate vector
       $\overrightarrow{v_i}$ of length $n$ in the same way as in step 4. If $\overrightarrow{v_i}$ is identical to $\overrightarrow{v}$ then
       A-cast Matched-$P_i$; otherwise let $k$ be the minimum index in $\overrightarrow{v_i}$ such that
       $\overrightarrow{v_i}[k] \neq \overrightarrow{v}[k]$, then A-cast $(P_l, P_i, P_k)$.
6. Set $\mathcal{P}'_{ex} = \mathcal{K}$. If $P_i \in \mathcal{P}'_{ex}$, then consider $m^*_\alpha$ as the final message.
```

Proof: To prove the lemma, we show that in every iteration of the `while` loop, the parties will be able to agree on a single party (using ACS) from $\overline{\mathcal{K}}$ (except with negligible probability, as the instance of ACS may not terminate with negligible probability), whose A-cast will be terminated. In other words, we assert that in every iteration of the `while` loop, there will exist one party from $\overline{\mathcal{K}}$ who will eventually A-cast a response. Moreover, this will be true, until the `while` loop is either over or broken due to the output of a triplet. For this, we claim that in every iteration of `while` loop, there must be an honest party, say P_i, belonging to $\mathcal{K}$, such that P_i is mapped to another honest party, say P_j, belonging to $\overline{\mathcal{K}}$. Moreover, honest P_i's message will eventually reach to honest P_j, who will then A-cast his response, which is either an n length vector or a triplet of parties.

At the time of entering into the loop for the first time, assume that among $t'+1$ parties in $\mathcal{K}$ there are $0 \leq c \leq t'$ corrupted parties. So the remaining $t'-c$ corrupted parties are in 'initial' $\overline{\mathcal{K}}$. In the worst case, c corrupted parties and $t'-c$ honest parties from $\mathcal{K}$ may be mapped to c honest parties and $t'-c$ corrupted parties, respectively from $\overline{\mathcal{K}}$. Still $\mathcal{K}$ contains at least one honest party which is bound to be mapped to another honest party from $\overline{\mathcal{K}}$, as there is no other unmapped corrupted party in $\overline{\mathcal{K}}$. So our claim holds for first iteration. In general in i^{th} iteration, there are $t'+i$ parties in $\mathcal{K}$ out of which say c with $0 \leq c \leq t'$ are corrupted parties. So extending the previous argument for this general case, there are i honest parties in $\mathcal{K}$ who are mapped to i honest parties in $\overline{\mathcal{K}}$. Among these i mappings, $i-1$ might correspond to previous $i-1$ iterations. But still one mapping is left for i^{th} iteration. Now let the mapping be from honest $P_j \in \mathcal{K}$ to honest $P_k \in \overline{\mathcal{K}}$.

So P_j's message reaches to P_k eventually and P_k tries to prepare $\overrightarrow{v_k}$ with received message and the (key, hash value) of the parties in $\mathcal{I}_1$. Conditioned on event E, P_k will receive the (key, hash value) of the parties in $\mathcal{I}_1$. Once P_k prepares his vector, he A-casts his response (which could be either Matched-P_k, if $\overrightarrow{v_k} = \overrightarrow{v}$ or a triplet of parties if $\overrightarrow{v_k} \neq \overrightarrow{v}$). If P_k's response is Matched-P_k, then $|\mathcal{K}|$ will be incremented by 1; otherwise, the loop will be broken with a triplet as output. □

Lemma 5 (Termination of Expansion Phase). *In a segment $\mathcal{S}_\alpha$, an execution of* **Expansion Phase** *will terminate, except with probability $2^{-\Omega(\kappa)}$, where termination means that the code either outputs a triplet or a set $\mathcal{P}'_{ex}$ of size $2t'+1$.*

Proof: From Lemma 4, in every iteration of the `while` loop, there will exist one party from $\overline{\mathcal{K}}$ who will eventually A-cast a response. Now conditioned on event E, the termination of an execution of **Expansion Phase** depends on the termination of the invoked ACS protocols and the A-casts. A-cast has no error in termination. The invocations of ACS (there can be at most t' invocations corresponding to t' iterations of while loop) will terminate, except with probability $2^{-\Omega(\kappa)}$. Therefore, an execution of **Expansion Phase** terminates, except with probability $2^{-\Omega(\kappa)}$. □

Lemma 6 (Correctness-I of Expansion Phase). *In an execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, all the honest parties in $\mathcal{P}'_{ex}$ (if found) will hold a common message m^*_α, which was also the common message held by the honest parties in $\mathcal{P}'_{ch}$, except with probability $2^{-\Omega(\kappa)}$. Moreover if the honest parties start $\mathcal{S}_\alpha$ with same input message m_α, then $m^*_\alpha = m_\alpha$.*

Proof: Let us consider party P_f, who is the first *honest* party to enter into 'initial' $\mathcal{K}$ during the **Expansion phase**. Recall that P_f enters into $\mathcal{K}$ (hence $\mathcal{P}'_{ex}$) when it receives a message $\overline{m^*_\alpha}$ from some already existing (possibly corrupted) party P_j in $\mathcal{K}$ and P_f's generated $\overrightarrow{v_f}$ is identical to $\overrightarrow{v}$. We claim that $\overline{m^*_\alpha} = m^*_\alpha$, except with error probability $2^{-\Omega(\kappa)}$. Consider an honest $P_k \in \mathcal{K}$ and an honest P_l in $\mathcal{I}_1$ with $\overrightarrow{v}[l] = 1$ (there is at least one such honest P_l as $|\mathcal{I}_1| \geq t'+1$). By **Collision Theorem**, $m_{k\alpha} = m_{l\alpha} = m^*_\alpha$, except with error probability $2^{-\Omega(\kappa)}$. Now since $\overrightarrow{v_f} = \overrightarrow{v}$, it implies that $\overrightarrow{v}_f[l] = 1$, as $\overrightarrow{v}[l] = 1$. This further implies that $\overline{m^*_\alpha} = m_{l\alpha}$. Hence it implies that $\overline{m^*_\alpha} = m^*_\alpha$ holds, with very high probability. This is because the key and hash value pair $(r_{lf}, \mathcal{V}_{lf})$ is not known to anyone (including possibly corrupted P_j) other than P_f and P_l. Hence with very high probability, P_f has received m^*_α from P_j.

Now let P_s be the second honest party to enter into 'initial' $\mathcal{K}$. P_s may receive its message either from P_f or from any party belonging to 'initial' $\mathcal{K}$. In both cases, P_s's message will be m^*_α, except with negligible error probability. In general, if an honest party P_i enters into 'initial' $\mathcal{K}$ at sometime, then its message will be equal to m^*_α, except with negligible error probability. □

Lemma 7 (Correctness-II of Expansion Phase). *In an execution of* **Expansion Phase** *in a segment $\mathcal{S}_\alpha$, if a triplet (P_m, P_l, P_k) is A-casted by P_l then at least one of P_m, P_l and P_k is corrupted, except with error probability $2^{-\Omega(\kappa)}$ where $P_m \in \mathcal{K}, P_l \in \overline{\mathcal{K}}$ and $P_k \in \mathcal{I}_1$.*

Proof: Let P_m, P_l and P_k be honest, where $P_m \in \mathcal{K}, P_l \in \overline{\mathcal{K}}$ and $P_k \in \mathcal{I}_1$. Since $P_k \in \mathcal{I}_1$, it implies that $\overrightarrow{v}(k) = 1$ holds. Also $P_m \in \mathcal{K}$ implies that $\overrightarrow{v}_m(k) = 1$. This further implies that m^*_α held by P_m is same as $m_{k\alpha}$ held by P_k, except with error probability $2^{-\Omega(\kappa)}$ (from **Collision Theorem**). Now during **Expansion phase**, P_m sends his m^*_α to P_l and P_l computes $\overrightarrow{v}_l$ with respect to the received m^*_α and the pairs $(r_{jl}, \mathcal{V}_{jl})$, corresponding to every $P_j \in \mathcal{I}_1$. On computing $\overrightarrow{v}_l$, party P_l will find that $\overrightarrow{v}_l(k) = \overrightarrow{v}(k)$, except with negligible error probability. This is because P_k is honest and hence $\mathcal{V}_{kl}$ is the hash value of $m_{k\alpha}$, with respect to the hash key r_{kl}. However, as shown above, m^*_α received by P_l from P_m is same as $m_{k\alpha}$, except with negligible error probability. So P_l will find that $\mathcal{V}_{kl} = \mathcal{U}_\kappa(m^*_\alpha, r_{kl})$. Hence P_l will not A-cast triplet (P_m, P_l, P_k). So if at all P_l A-casts (P_m, P_l, P_k), then at least one of P_m, P_l and P_k is corrupted. □

Output Phase. Once the parties agree on $\mathcal{P}'_{ex}$, with all honest parties in it holding some common m^*_α, we need to ensure that m^*_α propagates to all (honest) parties in $\overline{\mathcal{P}_{ex}} = \mathcal{P} \setminus \mathcal{P}'_{ex}$, in order to reach agreement on m^*_α. This is achieved in code Output (presented in Fig. 3) with the help of the parties in $\mathcal{P}'_{ex}$. A simple solution could be to ask each party in $\mathcal{P}'_{ex}$ to send his m^*_α to all the parties in $\overline{\mathcal{P}_{ex}}$, who can wait to receive $t'+1$ same m^*_α and then accept m^*_α as the message. This solution will work as there are at least $t'+1$ honest parties in $\mathcal{P}'_{ex}$. But clearly, this requires a communication complexity of $\mathcal{O}(\frac{\ell}{n} \cdot n^2) = \mathcal{O}(\ell n)$ bits for each segment (and thus $\mathcal{O}(\ell n^2)$ bits for our ABA protocol; this violates our promised communication complexity bound for Optimal-ABA). Hence, we adopt a technique proposed in [15] for designing a BA protocol in synchronous settings with $n = |\mathcal{P}| = 2t+1$ parties. Now the technique proposed in [15] requires a set of parties, say $\mathcal{H} \subset \mathcal{P}$ such that all the honest parties in $\mathcal{H}$ hold the same message and the majority of the parties in $\mathcal{H}$ are honest. Under this condition the technique allows the set of honest parties in $\mathcal{P} \setminus \mathcal{H}$ to obtain the common message of the honest parties in $\mathcal{H}$ with a communication cost of $\mathcal{O}(\ell n)$ bits. In our context $\mathcal{P}'_{ex}$ has all the properties of $\mathcal{H}$. Hence we adopt the technique of [15] in our context in the following way: Every $P_i \in \mathcal{P}'_{ex}$ sets $d = t'+1$ and $c = \lceil \frac{\ell+1}{td} \rceil$

and transforms his message m^*_α (with $|m^*_\alpha| = \frac{\ell}{t}$) into a polynomial $p(x)$ of degree $d-1$ over $GF(2^c)$. Now if somehow a party $P_j \in \overline{\mathcal{P}_{ex}}$ receives d values on $p(x)$, then he can interpolate $p(x)$ and receive m^*_α. For this, party $P_i \in \mathcal{P}'_{ex}$ sends i^{th} value on $p(x)$, namely $p_i = p(i)$ to every $P_j \in \overline{\mathcal{P}_{ex}}$. As the corrupted parties in $\mathcal{P}'_{ex}$ may send wrong p_i, P_j should be able to detect correct values. For this, every $P_i \in \mathcal{P}'_{ex}$ also sends hash values of $(p_1, \ldots, p_n)$ for a random hash key to every $P_j \in \overline{\mathcal{P}_{ex}}$. Now P_j can detect 'clean' values with the help of the hash values and eventually P_j will receive d 'clean' values (possibly from $d = t'+1$ honest parties in $\mathcal{P}'_{ex}$) using which he can compute m^*_α.

Fig. 3. Code for **Output Phase**

```
                                   Output

i. Code for $P_i$: Every party in $\mathcal{P}$ (not $\mathcal{P}'$) will execute this code.

  1. If $P_i \in \mathcal{P}'_{ex}$, do the following to help the parties in $\overline{\mathcal{P}_{ex}} = \mathcal{P} \setminus \mathcal{P}'_{ex}$ to compute $m^*_\alpha$:
     (a) Set $d = t'+1$ and $c = \lceil \frac{\ell+1}{td} \rceil$.
     (b) Interpret $m^*_\alpha$ as a polynomial $p(x)$ of degree $d-1$ over $GF(2^c)$. For this, divide
         $m^*_\alpha$ into blocks of $c$ bits and interpret each block as an element from $GF(2^c)$.
         These elements from $GF(2^c)$ are the coefficients of $p(x)$.
     (c) Send $p_i = p(i)$ to every $P_j \in \overline{\mathcal{P}_{ex}}$, where $p_i$ is computed over $GF(2^c)$.
     (d) For every $P_j \in \overline{\mathcal{P}_{ex}}$, choose a random distinct hash key $R_{ij}$ from $\mathbb{F}$ and send
         $(R_{ij}, \mathcal{X}_{ij1}, \ldots, \mathcal{X}_{ijn})$ to $P_j$, where for $k = 1, \ldots, n$, $\mathcal{X}_{ijk} = \mathcal{U}_\kappa(p_k, R_{ij})$. Here,
         to compute $\mathcal{X}_{ijk}$, interpret $p_k$ as a $c$ bit string.
     (e) Terminate this code with $m^*_\alpha$ as output.
  2. If $P_i \in \overline{\mathcal{P}_{ex}}$, do the following to compute $m^*_\alpha$:
     (a) Call $p_k$ received from party $P_k \in \mathcal{P}'_{ex}$ as 'clean' if there are at least
         $t'+1$ $P_j$'s in $\mathcal{P}'_{ex}$, corresponding to which $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ holds, where
         $(R_{ji}, \mathcal{X}_{ji1}, \ldots, \mathcal{X}_{jin})$ is received from $P_j \in \mathcal{P}'_{ex}$.
     (b) Wait to receive $d$ 'clean' $p_k$'s and upon receiving, interpolate $d-1$ degree poly-
         nomial $p(x)$ using those 'clean' values, interpret $m^*$ from $p(x)$ and terminate
         this protocol with $m^*_\alpha$ as the output.
```

Lemma 8 (Termination of Output Phase). *An execution of* **Output Phase** *in any segment $\mathcal{S}_\alpha$ will terminate, except with probability $2^{-\Omega(\kappa)}$.*

PROOF: From the steps of the code Output, the parties in $\mathcal{P}'_{ex}$ always terminate after performing the steps as mentioned in step 1(a)-1(d) of the code. So we now have to prove that the parties in $\mathcal{P} \setminus \mathcal{P}'_{ex}$ terminate, except with negligible error probability. To show this, we first assert that if all the honest parties in $\mathcal{P}'_{ex}$ hold common m^*_α, then the above event happens with no error; but the parties in $\mathcal{P}'_{ex}$ hold common m^*_α, except with negligible error probability (from Lemma 6). Hence it will follow that the parties in $\mathcal{P} \setminus \mathcal{P}'_{ex}$ terminate, except with negligible error probability.

Now we are left to show that the parties in $\mathcal{P} \setminus \mathcal{P}'_{ex}$ terminate without error when all the honest parties in $\mathcal{P}'_{ex}$ hold common m^*_α. Consider an honest party P_i in $\overline{\mathcal{P}_{ex}}$. Clearly, P_i terminates if it receives $d = t'+1$ 'clean' values eventually. To assert that P_i will indeed receive $d = t'+1$ 'clean' values, we first show that the value p_k received from every honest P_k in $\mathcal{P}'_{ex}$ will be considered as 'clean' by P_i. Consequently, since there are $t'+1$ honest parties in $\mathcal{P}'_{ex}$, P_i will eventually receive $t'+1$ 'clean' values even though the corrupted parties in $\mathcal{P}'_{ex}$ may never send any value to P_i. If the honest parties in $\mathcal{P}'_{ex}$ have common m^*_α, they will generate same $p(x)$ and therefore same $p_k = p(k)$. Hence, $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $\mathcal{P}'_{ex}$. As there are at least $d = t'+1$ honest parties in $\mathcal{P}'_{ex}$, the p_k received from honest $P_k \in \mathcal{P}'_{ex}$ will be considered as 'clean' by P_i. This proves our claim. □

Lemma 9 (Correctness of Output Phase). *Every honest party in $\mathcal{P}$ will output a common message m^*_α in an execution of* **Output Phase** *in a segment $\mathcal{S}_\alpha$, except with probability $2^{-\Omega(\kappa)}$. Moreover, if the honest parties start $\mathcal{S}_\alpha$ with same input m_α, then $m^*_\alpha = m_\alpha$.*

Proof: Lemma 6 shows that all the honest parties in $\mathcal{P}'_{ex}$ will output same m^*_α with very high probability. So we are left to prove that all the honest parties in $\overline{\mathcal{P}_{ex}}$ will output same m^*_α as well.

So let $P_i \in \overline{\mathcal{P}_{ex}}$ be an honest party. Now the p_k value of each honest $P_k \in \mathcal{P}'_{ex}$ will be eventually considered as 'clean' value by honest P_i. This is because there are at least $t'+1$ honest parties in $\mathcal{P}'_{ex}$, who hold same m^*_α and therefore same $p(x)$ (and hence $p(k)$). So $\mathcal{X}_{jik} = \mathcal{U}_\kappa(p_k, R_{ji})$ will hold, with respect to $(R_{ji}, \mathcal{X}_{jik})$ of every honest P_j in $\mathcal{P}'_{ex}$. A corrupted $P_k \in \mathcal{P}'_{ex}$ may send $\overline{p_k} \neq p_k$ to P_i, but $\overline{p_k}$ will not be considered as a 'clean' value with very high probability. This is because, in order to be considered as 'clean' value, $\overline{p_k}$ should satisfy $\mathcal{X}_{jik} = \mathcal{U}_\kappa(\overline{p_k}, R_{ji})$ with respect to $(R_{ji}, \mathcal{X}_{jik})$ of at least $t+1$ P_j's from $\mathcal{P}'_{ex}$. The test will fail with respect to an honest party from $\mathcal{P}'_{ex}$ with very high probability according to **Collision Theorem** (see Theorem 1). Thus though the test may pass with respect to all corrupted parties in $\mathcal{P}'_{ex}$ (at most t), the test will fail for every honest party from $\mathcal{P}'_{ex}$ with high probability. Hence, honest P_i will reconstruct $p(x)$ using d 'clean' values (which he is bound to get eventually), with very high probability. The second part is easy to follow. □

Properties of Optimal-ABA. We now prove the properties of Optimal-ABA.

Lemma 10. *In Optimal-ABA, in total there can be t segment failures. The* **Checking Phase** *and* **Expansion Phase** *may be executed for at most $2t$ times. But* **Output Phase** *may be executed at most t times, once for each segment.*

Proof: Since there are t corrupted parties, in total there can be t segment failures. These t failures may occur within a single segment or they may be distributed across t segments. After t failures, all corrupted parties will be removed from $\mathcal{P}$ and hence segment failure can not occur any more.

Since a segment may fail in **Expansion Phase**, there can be $2t$ executions of **Checking Phase** and **Expansion Phase** of which at most t may be non-robust executions (conflict triplet is found) and remaining t may be robust executions. Since segment can not fail in **Output Phase**, this phase may be executed at most t times, once for each segment. □

Lemma 11 (Termination of Optimal-ABA). *Protocol Optimal-ABA will terminate eventually, except with probability* $2^{-\Omega(\kappa)}$.

Proof: This follows from Lemma 1, Lemma 5 and Lemma 8 and the fact that event E occurs with very high probability. □

Lemma 12. *Conditioned on the event that segment* $\mathcal{S}_\alpha$ *terminates, every honest party outputs common* m^*_α *at the end of* $\mathcal{S}_\alpha$*, except with probability* $2^{-\Omega(\kappa)}$*. Moreover if the honest parties start* $\mathcal{S}_\alpha$ *with same input message* m_α*, then* $m^*_\alpha = m_\alpha$.

Proof: $\mathcal{S}_\alpha$ may terminate at the end of **Checking Phase** or at the end of **Output Phase**. If $\mathcal{S}_\alpha$ terminates at the end of **Checking Phase**, then every party assigns $m^*_\alpha = m^\dagger_\alpha$, where $m^\dagger_\alpha$ is a predefined value. Hence in this case the first part of the lemma holds without any error. Now let $\mathcal{S}_\alpha$ terminate at the end of **Output Phase**. Here we show that every party in $\mathcal{P}$ outputs common m^*_α at the end of **Output Phase** of $\mathcal{S}_\alpha$. By **Correctness of the Output Phase** (Lemma 9), given event E, all honest parties in $\mathcal{P}$ will hold common m^*_α, except with negligible error probability. Now since event E happens with very high probability, it follows that the parties in $\mathcal{P}$ will hold common m^*_α, except with negligible error probability.

The second part of the lemma follows from Lemma 2, 6 and 9. □

Lemma 13 (Correctness of Optimal-ABA). *Conditioned on the event that Optimal-ABA terminates, every honest party outputs common* m^* *at the end of Optimal-ABA, except with probability* $2^{-\Omega(\kappa)}$*. Moreover if the honest parties start Optimal-ABA with same input message* m*, then* $m^* = m$.

Proof: This follows from Lemma 12 and the fact that m^* is the concatenation of $m^*_1, \ldots, m^*_t$. □

Theorem 3. *Optimal-ABA is a* (ϵ, δ)*-ABA protocol.*

Proof: Follows from Lemma 11 and Lemma 13. □

Theorem 4. *Optimal-ABA privately communicates* $\mathcal{O}(\ell n + n^4\kappa)$ *bits and requires* $\mathcal{O}(n^3)$ *invocations to ABA (for single bit) and AVSS protocols (for one field element) to agree on an* ℓ *bit message.*

Proof: In Optimal-ABA, **Checking Phase** and **Expansion Phase** may be executed for at most $2t$ times and **Output Phase** may be executed t times (by Lemma 10).

In a single execution of **Checking Phase**, there are at most $2n'^2$ instances of AVSS. Moreover, there are two executions of ACS to agree on a set of parties of size $t' + 1$ and n' A-cast of n length response vectors. Since $n' = \mathcal{O}(n)$, the total communication complexity during one execution of **Checking Phase** is $2n^2 \cdot \mathsf{AVSS} + 2n \cdot \mathsf{ABA} + n^4$ bits.

During the execution of **Expansion Phase**, the most expensive step in terms of communication complexity is the execution of ACS, which will be executed t' times (the maximum number of iterations of `while` loop) in the `while` loop.

Since $t' = \mathcal{O}(n)$, this step requires a communication complexity of $n^2 \cdot \mathsf{ABA}$. Moreover, during **Expansion Phase** each party in $\mathcal{K}$ will privately send his ℓ/t bit message to exactly one party in $\overline{\mathcal{K}}$ to which it is mapped. As $|\mathcal{K}| = \mathcal{O}(n)$, this step requires a communication cost of $\mathcal{O}(n\ell/t)$ bits.

A single execution of **Output Phase** requires $\mathcal{O}(n'^2c + n'^3\kappa)$ bits of private communication. Now $\mathcal{O}(n'^2c + n'^3\kappa) = \mathcal{O}(\ell + n'^3\kappa)$ as $c = \lceil \frac{\ell+1}{td} \rceil = \lceil \frac{\ell+1}{tt'} \rceil$ and $n' = \mathcal{O}(n)$, $t' = \mathcal{O}(n)$.

So executing **Checking Phase** and **Expansion phase** $2t = \Theta(n)$ times and executing **Output Phase** t times require a communication complexity of $\mathcal{O}(\ell n + n^4\kappa)$ bits plus $\mathcal{O}(n^3)$ invocations to ABA and AVSS protocols. □

3 Open Problems

The communication complexity of Optimal-ABA shows that the protocol is communication optimal for sufficiently large ℓ and the bound on ℓ depends on the communication complexity of the underlying ABA and AVSS protocols. One may try to design communication optimal ABA protocol for all values of ℓ (if possible) using completely different approach.

References

1. I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine Agreement with optimal resilience. In *PODC*, pages 405–414, 2008.
2. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, pages 213–230, 2008.
3. M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *PODC*, pages 27–30, 1983.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.
5. M. BenOr, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *PODC*, pages 183–192, 1994.
6. G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *PODC*, pages 154 – 162, 1984.
7. R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute, Israel, 1995.
8. R. Canetti and T. Rabin. Fast asynchronous Byzantine Agreement with optimal resilience. In *STOC*, pages 42–51, 1993.
9. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(4):143–154, 1979.
10. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *STOC*, pages 383–395, 1985.
11. P. Feldman and S. Micali. An optimal algorithm for synchronous Byzantine Agreemet. In *STOC*, pages 639–648, 1988.
12. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine Agreement. *SIAM Journal of Computing*, 26(4):873–933, 1997.
13. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.
14. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement.* PhD thesis, ETH Zurich, 2002.

15. M. Fitzi and M. Hirt. Optimally efficient multi-valued Byzantine Agreement. In *PODC*, pages 163–168, 2006.
16. M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multiparty computation. In *ASIACRYPT*, pages 143–161, 2000.
17. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
18. A. Patra, A. Choudhary, T. Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *CRYPTO*, pages 487–504, 2009.
19. A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing and multiparty computation with optimal resilience. In ICITS, pages 74–92, 2009.
20. A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient asynchronous Byzantine Agreement with optimal resilience. In PODC, pages 92-101, 2009.
21. A. Patra and C. Pandu Rangan. Communication Optimal Multi-valued Asynchronous Broadcast Protocol. In LATINCRYPT, pages 162-177, 2010.
22. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.
23. B. Pfitzmann and M. Waidner. Unconditional Byzantine Agreement for any number of faulty processors. In *STACS*, pages 339–350, 1992.
24. B. Prabhu, K. Srinathan, and C. Pandu Rangan. Trading players for efficiency in unconditional multiparty computation. In *SCN*, pages 342–353, 2002.
25. M. O. Rabin. Randomized Byzantine generals. In *FOCS*, pages 403–409, 1983.
26. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, pages 73–85, 1989.
27. R. Turpin and B. A. Coan. Extending binary Byzantine Agreement to multivalued Byzantine Agreement. *Information Processing Letters*, 18(2):73–76, 1984.

Appendix A: Approach Used in the BA of [15]

Here we briefly recall the approach used in [15] for designing the communication optimal multi-valued BA protocol in synchronous settings. The protocol of [15] requires $n = 2t + 1$ parties. So $|\mathcal{P}| = 2t + 1$. The BA protocol was structured into three stages: (a) Checking, (b) Consolidation and (c) Claiming Stage. In the Checking Stage, the parties in $\mathcal{P}$ compare their respective messages and jointly determine an accepting subset $\mathcal{P}_{acc} \subseteq \mathcal{P}$ of size at least $n - t$, such that all 'accepting' parties hold the same message, and all (honest) parties holding this message are 'accepting'. This stage can be aborted when inconsistencies among honest parties are detected. If this stage is not aborted then the BA protocol proceeds to Consolidation Stage where the parties in $\mathcal{P}_{acc}$ help to decide on a happy subset $\mathcal{P}_{ok} \subseteq \mathcal{P}$, such that all 'happy' parties hold the same message, and the majority of 'happy' parties are honest. Also this stage may be aborted in case of inconsistencies among the honest parties' inputs. Consolidation Stage is very important and introduces new ideas. But a careful checking will reveal that the same ideas can not be implemented in asynchronous network even for $n = 3t + 1$ parties. That is why we introduce a new sets of ideas in our ABA protocol. Finally, if Consolidation Stage is not aborted then BA protocol of [15] proceeds to the last stage called Claiming Stage. In the Claiming Stage, the parties in $\mathcal{P}_{ok}$ distribute their common message to the unhappy parties i.e. the parties in $\mathcal{P} \setminus \mathcal{P}_{ok}$. This stage will never be aborted and hence at the end every party will output a common value. If the BA protocol aborts during Checking and Consolidation Stages then every party decides on a predefined default value.