Concurrent Asynchronous Byzantine Agreement in Expected-Constant Rounds, Revisited*

Ran Cohen† Pouyan Forghani‡ Juan Garay§ Rutvik Patel¶ Vassilis Zikas‖

Abstract

Without randomization, Byzantine agreement (BA) requires a linear number of rounds in the synchronous setting, while it is flat out impossible in the asynchronous setting. The standard primitive used to bypass these limitations is *oblivious common coin* (OCC), which enables parties to agree with constant probability on a random coin in an oblivious manner, i.e., without knowing whether agreement has been achieved.

The starting point of our work is the observation that no known protocol exists for information-theoretic multi-valued OCC (where the coin takes a value from a domain of size greater than 2) that achieves optimal resiliency in the asynchronous setting with eventual message delivery. This gap is particularly problematic because multi-valued OCC is used, implicitly or explicitly, in several constructions. In fact, it is often falsely attributed to the asynchronous BA result of Canetti and Rabin [STOC '93], which only achieves binary OCC and does not translate to the multi-valued setting.

In this paper, we fill this gap by presenting the first information-theoretic multi-valued OCC protocol in the asynchronous setting with optimal resiliency, tolerating $t < n/3$ corruptions. Further, our protocol efficiently supports an exponential-size domain, a feature not achieved even by existing protocols in the synchronous setting.

We then re-examine the problem of round-preserving parallel composition of asynchronous BA. Ben-Or and El-Yaniv [Distributed Computing '03] proposed a protocol for this task, but their construction suffers from several flaws: it relies on the non-existent multi-valued OCC of Canetti and Rabin, and even when instantiated with our new OCC protocol, its proof of expected constant round complexity is incorrect. To resolve this, we provide a simpler, modular protocol for parallel asynchronous BA. Finally, as a contribution of independent interest, we prove our results in Canetti's Universal Composability (UC) framework, providing the first composability guarantees for these essential building blocks.

*A preliminary version of this work appeared at *TCC 2023* [CFG+23].

†Efi Arazi School of Computer Science, Reichman University. E-mail: cohenran@runi.ac.il.

‡School of Cybersecurity and Privacy, Georgia Tech. E-mail: pforghani3@gatech.edu. Work partly done while the author was at Texas A&M University.

§Department of Computer Science and Engineering, Texas A&M University. E-mail: garay@tamu.edu.

¶E-mail: rpatel4554@gmail.com. Work partly done while the author was at Texas A&M University.

‖School of Cybersecurity and Privacy, Georgia Tech. E-mail: vzikas@gatech.edu. Work partly done while the author was at Purdue University.

Contents

1 Introduction **1**
- 1.1 Concurrent A-BA in Expected-Constant Rounds: Cracks in the Concrete 3
- 1.2 Overview of Our Results 4
- 1.3 Additional Related Work 9
- 1.4 Organization of the Paper 10

2 Model and Preliminaries **10**
- 2.1 UC Basics 11
- 2.2 Modeling Eventual Message Delivery in UC 11

3 Ideal Functionalities for a Few Standard Primitives **13**
- 3.1 Modeling Delayed Participation 13
- 3.2 Ideal Functionalities for a Few Standard Primitives 14

4 Asynchronous Oblivious Common Coin **18**
- 4.1 A-OCC Ideal Functionality 18
- 4.2 The A-OCC Protocol 19
- 4.3 Security Proof 21

5 Concurrent A-BA in Expected-Constant Rounds **30**
- 5.1 Concurrent A-BA Ideal Functionality 30
- 5.2 The Simplified Concurrent A-BA Protocol 31
- 5.3 Security Proof 35

Bibliography **40**

A Attack on Ben-Or and El-Yaniv's Select Protocol **45**
- A.1 Static Attack 49
- A.2 Adaptive Attack 51

B The Asynchronous Turpin-Coan Extension **51**

1 Introduction

Byzantine agreement (BA) [PSL80, LSP82] enables n parties to reach agreement on one of their inputs in an adversarial setting, facing up to t colluding and cheating parties. The core properties require all honest parties to eventually terminate with the same output (*agreement*), which equals their input value in case they all begin with the same common input (*validity*). BA and its closely related single-sender variant, *broadcast*, are fundamental building blocks in the construction of cryptographic protocols, in particular for *secure multi-party computation* (MPC) [Yao82, GMW87, BGW88, CCD88], in which parties wish to privately compute a joint function over their inputs.

We consider a complete network of authenticated and private point-to-point (P2P) channels, which enables every pair of parties to communicate directly. The central settings in which BA has been studied are:

The *synchronous* setting. Here the protocol proceeds in a round-by-round fashion, and messages sent in a given round are guaranteed to be delivered by the start of the next round. The round structure can be achieved given synchronized clocks and a known bound on message delivery, and enables the use of *timeouts*. This clean abstraction allows for simpler analyses of protocols, but comes at the cost that parties must wait for the worst-case delay in each round before they can proceed.

The *asynchronous* setting. Here no assumptions are made on the clocks or on the bound of message delivery (other than that each message will be eventually delivered), and messages may be adversarially delayed by an arbitrary (yet finite) amount of time. The main challenge is that timeouts cannot be used, implying the inability to distinguish a slow honest party from a silent, noncooperative corrupted party. On the positive side, asynchronous protocols can advance as fast as the network allows, irrespectively of the worst-case delay, and each party can proceed to the next step as soon as it gets sufficiently many messages.

The *feasibility* of BA is inherently related to the synchrony assumptions of the system. Synchronous BA with *perfect* security is achievable with deterministic protocols for $t < n/3$ [PSL80, BGP89, GM98]; this bound is tight in the plain model[1] even when considering weaker, computational security [PSL80, FLM86, Bor96], but can be overcome with setup assumptions, yielding computationally secure BA [DS83], and even information-theoretically secure BA [PW92], for $t < n/2$.[2] On the other hand, deterministic asynchronous BA (A-BA) is impossible even facing a single crash failure [FLP85], and randomized solutions are a necessity. Following Ben-Or [Ben83] and Rabin [Rab83], information-theoretic randomized A-BA has been achieved with $t < n/3$ corruptions [Bra87, CR93, ADH08], which, as opposed to the synchronous case, is a tight bound even given setup and cryptographic assumptions [DLS88, BT85].

Round complexity of BA. In the synchronous setting, it is known that *deterministic* BA requires $t + 1$ rounds [FL82, DS83], a bound that is matched by early feasibility results [PSL80, DS83, GM98]. It is also known that t-secure randomized BA for $t \in \Theta(n)$ cannot terminate in a *strict* constant number of rounds [CMS89, KY86, CPS19]; yet, *expected* constant-round protocols have been constructed both for $t < n/3$ in the plain model [FM97] and for $t < n/2$ in the PKI

[1] That is, without setup assumptions and without imposing resource restrictions on the adversary.

[2] The bound $t < n/2$ is tight for BA [Fit03]; however, under the same setup assumptions, broadcast can be solved for any number of corruptions.

model [FG03, KK09].[3] The latter protocols follow the approach of Rabin [Rab83] and rely on an *oblivious common coin* (OCC), that is, a distributed coin-tossing protocol over a domain V such that the output of every honest party is a common random value $v \in V$ with constant probability p, but with probability $1 - p$ is independently and adversarially chosen; the coin toss is "oblivious" since the parties cannot distinguish between a successful coin toss and an adversarial one.[4]

Loosely speaking, round complexity in the asynchronous setting can be defined based on the expected number of times an honest party has to alternate between sending and receiving messages that are "causally related" [CGHZ16]. In an unpublished manuscript, Feldman [Fel89] generalized the expected-constant-round synchronous protocol of [FM88] to asynchronous networks with $t < n/4$; at the core of the construction lies a **binary** (asynchronous) OCC protocol which actually has resiliency $\min(t', \lceil n/3 \rceil - 1)$, where t' is the corruption threshold of an *asynchronous verifiable secret sharing* (A-VSS) scheme.[5] Canetti and Rabin [CR93] constructed A-VSS for $t < n/3$, thereby obtaining A-BA in expected-constant rounds for the optimal threshold $t < n/3$.

Concurrent BA. All of the above constructions are for solving a *single* instance of BA. However, most applications, particularly MPC protocols, require composing multiple instances of BA: sequentially, in parallel, or concurrently. While the composition of synchronous, deterministic protocols is relatively simple (although care must be taken in the cryptographic setting [LLR06]), composing expected-constant-round protocols with probabilistic termination is a much more challenging task.

Indeed, Ben-Or and El-Yaniv [BE88, BE03] observed that running m instances of a probabilistic-termination protocol in parallel may incur a blow-up in the expected number of rounds until they all terminate. The technical reason is that the expectation of the maximum of m independent, identically distributed random variables does not necessarily equal the maximum of their expectations. In particular, for expected-constant-round BA with a geometric termination probability (which is the case in all known protocols), the parallel composition of m instances terminates after expected $\Theta(\log m)$ rounds [CCGZ19]. Further, even when all parties start the protocol together, simultaneous termination is not guaranteed as the adversary can force some honest parties to terminate after the others; although this gap can be reduced to a single round, inaccurate sequential re-synchronization of ℓ instances of BA may lead to an exponential blow up in ℓ if not done with care [LLR02]. Following [BE03, LLR02, LLR06, KK09], recent works have shown how to compose synchronous BA in a round-preserving way with simulation-based security in Canetti's framework for universal composability (UC) [Can20], with optimal resiliency and *perfect* security in the plain model for $t < n/3$ [CCGZ19], and with cryptographic security in the PKI model for $t < n/2$ [CCGZ21].

In the asynchronous setting, the parties are not assumed to begin the protocol at the same time, so intuitively, sequential composition is not problematic. However, running m instances of expected-constant-round A-BA concurrently would yield a $\Theta(\log m)$ blowup as in the synchronous case. Ben-Or and El-Yaniv [BE03] further showed how to execute m instances of A-BA in expected-constant rounds and with optimal resiliency $t < n/3$; however, their solution is more complicated than the synchronous one. In addition, they only prove a property-based security definition of A-BA that does not necessarily address modern security requirements such as security under composition, or facing adaptive adversaries.

[3] Fitzi and Garay [FG03] devised expected-constant-round BA for $t < n/2$ in the PKI model under number-theoretic assumptions. Katz and Koo [KK09] established a similar result from the minimal assumption of digital signatures, which yields an information-theoretic variant using pseudo-signatures [PW92].

[4] This primitive is sometimes known as a "weak" common coin in the literature.

[5] Feldman's A-VSS suffers from a negligible error probability. An *errorless* A-VSS scheme for $t < n/4$ is given in [BCG93] and used to construct a *perfectly* secure asynchronous MPC protocol with resiliency $t < n/4$.

1.1 Concurrent A-BA in Expected-Constant Rounds: Cracks in the Concrete

The underlying idea behind the synchronous protocol of Ben-Or and El-Yaniv [BE03] for parallel BA is to execute each BA instance multiple times over the same inputs, but only for a constant number of rounds. For a suitable choice of parameters, this ensures that with high probability each party will have obtained at least one output value in each such batch. To coordinate these outputs, the parties then run an *oblivious leader election* (OLE) protocol, which guarantees that with constant probability, a random leader is elected. In the event that the leader is honest and the parties obtained an output in each batch (which, again, occurs with constant probability) the parties will terminate; otherwise they repeat the process.

The same general technique underlies Ben-Or and El-Yaniv's asynchronous protocol, but great care is needed to deal with the low message dispersion inherent in asynchronous networks while maintaining optimal resiliency $t < n/3$. A closer look at their security proof indeed raises a number of subtle issues. First, they point to Canetti and Rabin [CR93] for instantiating the (asynchronous) OLE primitive used in their construction (called *A-Election()*). (Recall that Canetti and Rabin construct an OCC to obtain A-BA in expected-constant rounds; an n-valued OCC would indeed imply OLE.) As it turns out, the OCC construction in [CR93, Sec. 8] (as well as the more detailed versions in [Can96, Sec. 5.7] and [CR98, Sec. 8]) is only *binary* (i.e., it only works for $V = \{0, 1\}$), and it does not seem straightforward to generalize to larger (non-constant-sized) domains.

Further, running $\log n$ executions of a binary OCC, with agreement probability p, in parallel to make it multi-valued yields only $p^{\log n} \in 1/\mathsf{poly}(n)$ probability of agreement, and as long as the coin is not *perfectly* fair (i.e., guaranteeing that all honest parties output 0 with probability 1/2, and 1 with probability 1/2, which is impossible in the asynchronous setting [dSKT22]), that would not imply OLE with constant success probability. We can also consider a different type of "strong" common coin: one which achieves arbitrarily small bias, possibly with guaranteed agreement. Such an ϵ-biased binary coin would in fact achieve arbitrarily small bias even when run in parallel, and there are even known protocols for this task that simultaneously enjoy almost-sure termination and optimal resiliency ($t < n/3$) [ADS22], but these coins do not have constant round complexity, since at the very least there will be a dependence on ϵ.[6]

We note that Patra *et al.* [PCR14] claim to construct a $(t + 1)$-bit asynchronous OCC, but their main focus is on communication complexity, and the agreement probability of their protocol is no better than would be obtained by running $t + 1$ executions of a binary OCC protocol (i.e., exponentially small). Techniques used to get OLE in the synchronous setting [FM88, KK09] do not seem to extend in asynchrony for $t < n/3$ (we elaborate on this in Section 1.2). Thus, to the best of our knowledge, no existing OCC construction is *simultaneously* optimally resilient, multi-valued, and asynchronous, without relying on computational assumptions (in Section 1.3 we discuss solutions in the cryptographic setting).

Second, there is a subtle issue in the logic of one of the proofs in [BE03]. This issue raises concerns about the validity of the proof claiming an expected-constant round complexity of one of the main subroutines—namely, the Π_{select} subroutine, which handles the shortcomings of their message-distribution mechanism. Specifically, in the analysis of Π_{select} it is claimed that if the leader is chosen from a certain set, the protocol will terminate. However, further examination reveals that there are scenarios in which the protocol may not terminate for certain leaders from that set. As a result, this issue casts doubt on the promised expected-constant round complexity of their concurrent A-BA protocol. (Refer to Appendix A for more details on this issue.)

Finally, the concurrent asynchronous (resp., synchronous) BA protocol in [BE03] relies on *multi-*

[6] The protocol in [ADS22] actually incurs an additional overhead of several factors of n, due to its use of ACS and almost-surely terminating A-BA.

valued asynchronous (resp., synchronous) BA in expected-constant rounds.[7] Recall that running the binary protocols in [FM97] or [CR93] $\omega(1)$ times in parallel would terminate in expected $\omega(1)$ rounds, so they cannot be naïvely used for this task. In the synchronous setting, Turpin and Coan [TC84] extended binary BA to multi-valued BA for $t < n/3$ with an overhead of just two rounds. Ben-Or and El-Yaniv claim that this technique can be adapted for asynchronous networks by using Bracha's "A-Cast" primitive [Bra87] for message distribution. However, a closer look reveals that although the Turpin-Coan extension works (with appropriate modifications) in the asynchronous setting for $t < n/5$, it **provably does not work** when $t \geq n/5$, regardless of the specific choice of the underlying binary A-BA protocol and even when the adversary is limited to static corruptions (see Appendix B).

More recently, an optimally resilient multi-valued A-BA protocol with expected-constant round complexity was proposed by Patra [Pat11], but it relies on the *Agreement on a Common Subset* (ACS) protocol in [BKR94]; however, as we explain below, this ACS protocol does not achieve expected-constant round complexity without some modifications, which require either expected-constant-round concurrent A-BA (this would be circular), or information-theoretic asynchronous OLE with optimal resiliency. Fortunately, Mostéfaoui and Raynal [MR17] recently gave a black-box, constant-round reduction from multi-valued to binary A-BA for $t < n/3$,[8] using just one invocation of the underlying binary protocol as in [TC84].

We emphasize that the asynchronous protocol of Ben-Or and El-Yaniv [BE88, BE03] lies, either explicitly or implicitly, at the core of virtually every round-efficient asynchronous MPC construction [BCG93, BKR94, HNP05, BH07, HNP08, Coh16, CGHZ16, BZL20, LLM+20]. The concerns raised above regarding the result of [BE03] render this extensive follow-up work unsound. In this paper, we revisit this seminal result and rectify these issues.

1.2 Overview of Our Results

We now present an overview of our results, which are three-fold, including a detailed exposition of our techniques.

Multi-valued and asynchronous oblivious common coin. As a starting point, we look at the *binary* asynchronous OCC protocol of Canetti and Rabin [CR93]. The idea (following the approach of [FM97] in the synchronous setting and [Fel89] in the asynchronous setting) is that each party secret-shares a random vote for each party, using an optimally resilient A-VSS scheme. Each party accepts $t + 1$ of the votes cast for him (at least one of which must have come from an honest party), and once it is determined that enough secrets have been fixed (based on several rounds of message exchange, using Bracha's A-Cast primitive [Bra87] for message distribution), the parties begin reconstructing the accepted votes. The sum or "tally" of these votes becomes the value associated with the corresponding party. After computing the values associated with an appropriate set of at least $n - t$ parties, an honest party outputs 0 if at least one of those tallies is 0, and outputs 1 otherwise. Note that each tally must be uniformly random, and the properties of A-Cast guarantee that no two honest parties can disagree on the value of any given tally (although up to t tallies may be "missing" in an honest party's local view); moreover, Canetti and Rabin show using a counting argument that at least $n - 2t > n/3$ of the tallies are known to all honest parties

[7] This is true even if one is interested only in *binary* concurrent BA (i.e., when the input vectors consist of bits). Multi-valued BA is needed to agree on the leader's output vector.

[8] They are also concerned with obtaining $O(n^2)$ message complexity. The novelty of their result, even without this more stringent requirement, does not seem to be acknowledged in the paper.

(i.e., common to their local views).[9] This can be used to show that with probability at least 1/4 all honest parties output 0, and similarly for 1.

In the conference version of Feldman and Micali's paper [FM88], there is a brief remark suggesting that the synchronous version of the above protocol can be modified to obtain (oblivious) leader election.[10] Rather than outputting a bit, parties output the index of the party whose tally is minimum; when the domain of secrets is large enough, with constant probability the same *honest* party is chosen. This approach was fully materialized by Katz and Koo [KK09] for $t < n/3$ in the information-theoretic setting and $t < n/2$ in the computational setting. We note that what is obtained is not exactly OLE as we have defined it, as the adversary can of course bias the index of the elected party, but nevertheless, this is sufficient for the concurrent BA protocols of Ben-Or and El-Yaniv [BE03].

Unfortunately, the approach effective in the synchronous setting cannot be directly applied in the asynchronous setting while maintaining optimal resiliency. The challenge arises from the fact that the adversary can selectively remove t coordinates from the honest parties' local views of the tallies, ensuring that the leader is not chosen from the parties corresponding to those missing coordinates. This poses a significant obstacle as the original concurrent A-BA protocol by Ben-Or and El-Yaniv [BE03] terminates successfully when the leader is honest and selected from an adversarially chosen subset of $n - 2t$ parties. Consequently, when $t < n/3$, the size of this subset is only $t + 1$, allowing the adversary to reduce the probability of choosing an appropriate leader to as low as $\frac{1}{n-t}$. The same issue arises in our simplified concurrent A-BA protocol, where we also require the leader to be selected from an adversarially chosen subset of parties with size $n - 2t$. Again, when $t < n/3$, this subset can be of size $t + 1$, leading to the same challenge. However, when $t < n/4$, the set of potential appropriate leaders becomes larger in both concurrent A-BA protocols, enabling us to circumvent this issue. Therefore, in addition to the inherent value in obtaining a true OCC (where a successful coin toss produces a uniform value), we specifically need such a primitive to obtain optimal resiliency for concurrent A-BA.

In summary, no OLE construction—and therefore no concurrent A-BA protocol in expected-constant rounds—exists with optimal resiliency $t < n/3$ in asynchronous networks. We now describe our solution to this problem, which is based on the following simple combinatorial observation. If we work over a field of size $N \in \Theta(n^2)$, then with *constant* probability, at least one value will be repeated in the global view of tallies, and, moreover, all repeats will occur within the subset of indices known to all honest parties. The intuition for this fact is that when $N \in O(n^2)$ there is, due to the birthday paradox, at least one repeat in any constant fraction of the indices with constant probability, and when $N \in \Omega(n^2)$, there are no repeats at all with constant probability. There is a "sweet spot" between these extremes that can be leveraged to extract shared randomness: Honest parties output the value that is repeated in their local views of the tallies and has the minimum index in the vector of tallies. With this modification of Canetti and Rabin's protocol [CR93], we obtain the first n^2-valued asynchronous OCC for $t < n/3$ in the information-theoretic setting. We remark that the above combinatorial observation at the heart of our construction may be of independent interest.

It is straightforward to extend our OCC protocol to accommodate arbitrary domains V. One approach is to work over a prime field of size at least $\mathrm{lcm}(n^2, |V|)$. In this setting, we can still identify "repeats" by considering the tallies modulo n^2. Once the repeat with minimum index is determined, we can generate a common random output by reducing the corresponding (original)

[9] An enhanced version of this counting argument, using the "Gather" protocol of Abraham *et al.* [AJM+23], can be used to increase the size of this intersection to $n - t > 2n/3$.

[10] This claim no longer appears in the ICALP [FM89] or journal [FM97] versions of the paper, or in Feldman's thesis [Fel88].

field element modulo $|V|$. In particular, when $|V| = n$, we get asynchronous OLE with optimal resiliency.

Proving the security of our multi-valued OCC protocol in a *simulation-based* manner is not without its own challenges. The issue is that the simulator must expose a view of the vector of tallies that both adheres to the distribution in the real world and is consistent with the random value chosen by the OCC functionality (in the case of a successful coin toss) in the ideal world. While the simulator can easily determine the exact set of indices known to all honest parties from its internal execution with the real-world adversary, properly sampling the "repeat pattern" according to these constraints is a delicate task; furthermore, since the functionality is only parameterized by a (constant) lower bound on the probability of a successful coin toss, the simulator must handle the complementary case carefully in order to avoid skewing the distribution. It is not immediately clear how to perform this inverse sampling efficiently.

A heavy-handed solution is to simply have the functionality sample the tallies itself, and then determine the output based on the location of repeats relative to the subset of indices (supplied by the simulator) that are known to all honest parties. This protocol-like functionality would certainly allow for simulation—and would in fact be sufficient for our purposes—but its guarantees are more difficult to reason about and, more importantly, it cannot be realized by other protocols! Instead, we construct a simulator that, given the exact probabilities of certain events in the protocol, can also efficiently sample from those events, potentially conditioned on the output of a successful coin toss. By selecting the appropriate sampling procedure, the simulator can derive a vector of tallies that preserves the (perfect) indistinguishability of the real and ideal worlds. Equipped with the means to carry out the inverse sampling, we can now realize a more abstract (and natural) OCC functionality, which is ready to be plugged into higher-level protocols.

Simplified concurrent A-BA. Ben-Or and El-Yaniv [BE03] devised an expected-constant-round concurrent A-BA protocol. However, in addition to relying on unspecified building blocks, as mentioned above, it suffers from logical issues in its proof. Although our new OCC protocol can instantiate the missing primitive, doubts remain about the expected-constant round complexity due to a lingering issue in the proof. This issue stems from the steps taken to address low message dispersion, and the possibility of resolving it without changing the protocol is unclear. To tackle this, we redesign the message-distribution phase, avoiding the problem in the proof and obtaining stronger guarantees. These guarantees simplify the protocol structure, achieving a level of simplicity comparable to the synchronous solution.

In more detail, we build on Ben-Or and El-Yaniv's work, where parties initiate multiple executions for each A-BA instance that are "truncated" after a fixed number of iterations. However, we adopt a different approach to message distribution, allowing us to establish the rest of the protocol based on the simpler structure of their synchronous solution. Similarly to [BE03], we set parameters to ensure that each party receives at least one output from each batch of truncated A-BA executions (for each instance) with constant probability. Based on their local results from those truncated A-BAs, parties create suggestions for the final output. Subsequently, the message-distribution phase begins with parties running a binary A-BA to verify a precondition, ensuring the ability to validate each other's suggestions. If they choose to proceed, they A-Cast their suggestions and only accept those they can validate using their local results from the truncated A-BAs. This validation relies on the property that honest parties terminate within two consecutive iterations in each A-BA execution, ensuring the validity of suggestions even when provided by corrupted parties. Once parties receive $n - t$ suggestions, they proceed by A-Casting the set of the first $n - t$ suggestions they receive, along with the identities of their providers. The checked precondition

guarantees that everyone can move on to the next step.

Parties then wait until they receive enough suggestions and enough sets, ensuring that at least $n - t$ sets are completely contained within the set of suggestions they received. By employing a counting argument similar to [Fel89, CR93], we can ensure the delivery of suggestions from at least $n - 2t > n/3$ parties to all honest participants. Moreover, the validation process applied to the suggestions guarantees that each honest party only accepts valid suggestions, even if they originate from corrupted sources. These robust guarantees in the distribution of suggested outputs establish that the $n - 2t$ suggestions commonly received by honest parties are legitimate outputs. This enables us to directly utilize the synchronous protocol in the asynchronous setting.

We employ our new asynchronous OCC protocol to elect a leader for party coordination. Every party adopts the leader's suggestion and runs a multi-valued A-BA to handle the obliviousness of the leader-election mechanism. If the parties successfully agree on a random leader who happens to be chosen from the $n-2t > n/3$ parties whose suggestions are commonly received, which occurs with constant probability, all honest parties initiate the A-BA protocol with the leader's suggestion and output the same value. If the leader is not among those parties (or there is disagreement on the leader), precautions are taken to ensure no malicious value is output. For this purpose, we utilize an "intrusion-tolerant" A-BA protocol that guarantees the output to be either a default value $\perp$ or one of the honest parties' inputs.[11] Thus, in case the output is not $\perp$ the parties can safely output it, and otherwise they can simply start over. Unlike the synchronous solution in [BE03], which includes a validation step in between the agreement on the leader's suggestion and a final agreement on termination, our concurrent A-BA protocol involves validation in the message-distribution phase, and so consensus on a non-default value can directly be used to agree on termination.

By following the above approach, we sidestep the issues in Ben-Or and El-Yaniv's proof and achieve a significantly simpler protocol structure for the asynchronous setting compared to the one presented in [BE03]. Somewhat surprisingly, the resulting protocol is conceptually as simple as its synchronous counterpart.

Our concurrent A-BA protocol also yields an expected-constant-round solution to agreement on a core set (ACS), the input-selection task underlying asynchronous MPC [BCG93]. Applying concurrent A-BA to an existing ACS construction requires care. In the construction of [BKR94], a party supplies input 1 to an A-BA instance upon completing the corresponding input sharing, but supplies input 0 to the remaining instances only after sufficiently many other instances have decided 1. As observed in [ZD22, AAPS24], these dependencies prevent a direct replacement of the individual A-BA instances by a concurrent A-BA invocation requiring a complete input vector. The same issue arises in constructions that use this ACS procedure, including [CGHZ16, BZL20].

We eliminate these dependencies by first running the information-theoretic party-gather protocol of [AAPS24, Section 4]. Let $R_i \subseteq [n]$ be the growing set of dealers whose A-VSS sharing phases have completed at P_i, and let P_i validate dealer j when $j \in R_i$. The termination and totality properties of A-VSS provide the required validation conditions: every honest dealer is eventually validated by every honest party, and any dealer validated by one honest party is eventually validated by all honest parties. Gather produces a set $G_i \subseteq R_i$ at each honest party, with the guarantee that there exists a common set $G^\star$, of size at least $n - t$, contained in every honest party's output G_i. The sets G_i need not be identical, and the parties need not identify $G^\star$.

[11] Ben-Or and El-Yaniv [BE03] introduced and used a strengthened property for (A-)BA without naming it, which was later called "non-intrusion" validity in [MR17]. Non-intrusion validity lies between standard validity and "strong" validity [FG03], as it requires that a value decided by an honest party is either an honest party's input or a special symbol $\perp$ (i.e., the adversary cannot intrude malicious values into the output).

Upon obtaining G_i, party P_i fixes the entire vector

$$b_{i,j} = \begin{cases} 1 & \text{if } j \in G_i, \\ 0 & \text{otherwise,} \end{cases} \qquad j \in [n],$$

and submits it to a single invocation of our concurrent A-BA protocol. Let d_j be the common decision for coordinate j, and define

$$C = \{j \in [n] : d_j = 1\}.$$

If the concurrent A-BA interface also permits the default output $\perp$, that output is treated as exclusion. Parties continue the underlying sharing protocols and, before proceeding with the MPC computation, wait until the sharing phase of every dealer in C has completed locally.

Agreement on C follows from coordinate-wise A-BA agreement. For every $j \in G^\star$, all honest parties supply input 1, so validity implies $d_j = 1$; hence $|C| \geq n - t$. Conversely, if $d_j = 1$, at least one honest party must have supplied input 1, since otherwise validity would force decision 0. That party has completed dealer j's sharing phase, so A-VSS totality ensures that every honest party eventually completes it. Thus C is a common set of at least $n - t$ dealers whose shared inputs are available for the subsequent computation.

Gather contributes only a constant number of asynchronous rounds. With constant-round A-VSS, including its completion-propagation guarantee, and our expected-constant-round concurrent A-BA protocol, the resulting ACS has expected $O(1)$ round complexity. Different parties may enter concurrent A-BA at different times, but each enters with all of its inputs fixed; no A-BA input depends on another A-BA decision. This supplies the input-selection stage needed by the asynchronous MPC construction. The gather step tolerates $t < n/3$ and introduces no computational assumptions; the security and resilience of the overall MPC protocol remain subject to the guarantees of its other building blocks.

Expected-constant-round ACS can also be obtained directly from other constructions. Abraham *et al.* [AAPS24] provide information-theoretic ACS protocols with perfect security for $t < n/4$ and statistical security for $t < n/3$; their work also supplies the information-theoretic gather primitive used in the reduction above. Duan *et al.* [DWZ23] give an expected-constant-round ACS construction based on multivalued validated Byzantine agreement, assuming suitable common-coin primitives. These constructions provide alternative implementations of the input-selection stage of asynchronous MPC

Composable treatment of expected-constant-round concurrent A-BA. We choose to work in Canetti's Universal Composability (UC) framework [Can20], and as such, we prove the security of our protocols in a *simulation-based* manner. (See Section 2.1 for a high-level overview.) The UC framework provides strong composability guarantees when secure protocols are run as a subroutine in higher-level protocols (this is absolutely critical in our context given that expected-constant-round concurrent A-BA is a key building block in many round-efficient cryptographic protocols, as mentioned above), and even in *a priori* unknown or highly adversarial environments (such as asynchronous networks). Moreover, it enables us to provide a modular, bottom-up security analysis. However, obtaining a composable and round-preserving treatment of "probabilistic-termination" BA is non-trivial, as pointed out by Cohen *et al.* [CCGZ19, CCGZ21] in the synchronous setting. In the following, we discuss some unique issues in asynchrony and how we address them.

To model eventual message delivery, we follow [KMTZ13, CGHZ16, Coh16] and require parties to repeatedly attempt fetching messages from the network. The first D such requests are ignored

by the functionality, where D is a value provided by the adversary in unary so that it remains bounded by the adversary's running time (i.e., so that messages cannot be delayed *indefinitely*). It is straightforward then to derive a formal notion of asynchronous rounds in UC, based on this mechanism. We remark that unlike in the synchronous setting [KMTZ13], (asynchronous) rounds cannot be used by the environment to distinguish the real and ideal worlds in the asynchronous setting. Thus, as opposed to [CCGZ19, CCGZ21], our functionalities are round-unaware. Similarly, we do not need to deal with standard issues in sequential composition, namely, non-simultaneous start/termination ("slack"), since asynchronous protocols are already robust to slack. Indeed, it is entirely possible that some parties receive output from a (secure) asynchronous protocol before other parties have even started the protocol!

On the other hand, the issue of *input incompleteness* is trickier to address. This refers to the problem that in the asynchronous setting, the inputs of up to t honest parties may not be considered in the result of the computation; the remaining $n - t$ parties form a "core set" of input providers. Note that in the worst case, the core set is adversarially chosen and includes all corrupted parties. Prior work [CGHZ16, Coh16] allowed the adversary to send an explicit core set to the functionality; however, this approach does not always accurately model what happens in the real world, and can cause difficulties in the simulation. Instead, our solution is to allow the adversary to define the core set *implicitly*, by delaying the submission of inputs to the functionality in the same way that it delays the release of outputs from the functionality. Using this novel modeling approach, we obtain updated functionalities for some standard asynchronous primitives that more accurately capture realizable security guarantees.

1.3 Additional Related Work

As mentioned earlier, the $t + 1$ lower bounds for deterministic BA [FL82, DS83] were extended to rule out *strict*-constant-round t-secure randomized BA for $t = \Theta(n)$ [CMS89, KY86, CPS19]; these bounds show that any such r-round BA must fail with probability at least $(c \cdot r)^{-r}$ for a constant c, a result that is matched by the protocol of [GGL22]. Cohen *et al.* [CHM+22] showed that for $t > n/3$, two-round BA is unlikely to reach agreement with constant probability, implying that the *expected* round complexity must be larger; this essentially matches Micali's BA [Mic17] that terminates in three rounds with probability 1/3. Attiya and Censor-Hillel [AC10] extended the results on worst-case round complexity for $t = \Theta(n)$ from [CMS89, KY86] to the asynchronous setting, showing that any r-round A-BA must fail with probability $1/c^r$ for some constant c.

In the dishonest-majority setting, expected-constant-round broadcast protocols were initially studied by Garay *et al.* [GKKO07], who established feasibility for $t = n/2 + O(1)$ as well as a negative result. A line of work [FN09, CPS20, WXDS20, WXSD20, SLM+23] established expected-constant-round broadcast for any constant fraction of corruptions under cryptographic assumptions.

Synchronous and (binary) asynchronous OCC protocols in the information-theoretic setting were discussed earlier. Using the synchronous protocol in [BE03], Micali and Rabin [MR90] showed how to realize a *perfectly unbiased* common coin in expected-constant rounds for $t < n/3$ over secure channels (recall that this task is impossible in asynchronous networks [dSKT22]). In the cryptographic setting, both synchronous and asynchronous OCC protocols with optimal resiliency are known, relying on various computational assumptions; we mention a few here. Beaver and So [BS93] gave two protocols tolerating $t < n/2$ corruptions in synchronous networks, which are secure under the quadratic residuosity assumption and the hardness of factoring, respectively. Cachin *et al.* [CKS05] presented two protocols for $t < n/3$ and asynchronous networks, which are secure in the random oracle model based on the RSA and Diffie-Hellman assumptions, respectively. Nielsen [Nie02] showed how to eliminate the random oracle and construct an asynchronous OCC

protocol relying on standard assumptions alone (RSA and DDH). Although these constructions are for asynchronous networks, they can be readily extended to work in synchronous networks for $t < n/2$ (e.g., so that they can be used in the computationally secure BA protocol from [FG03]). We also note that while the resiliency bounds for asynchronous OCC protocols coincide in the information-theoretic and computational settings, working in the latter typically yields expected-constant-round A-BA protocols that are much more efficient in terms of communication complexity (i.e., *interaction*) than the unconditionally secure protocol of Canetti and Rabin [CR93].

Lastly, we discuss solutions to multi-valued A-BA in the computational setting. Cachin *et al.* [CKPS01] studied a more general version of this problem, in which the validity property is replaced with "external" validity, where the output domain can be arbitrarily large but the agreed-upon value must only satisfy an application-specific predicate. They gave a construction for multi-valued "validated" A-BA that runs in expected-constant rounds for $t < n/3$, assuming a PKI and a number of threshold cryptographic primitives (including an asynchronous OCC), and used it to obtain an efficient protocol for asynchronous *atomic* broadcast. Recently, Abraham *et al.* [AMS19] (and follow-ups, e.g., [LLTW20, AJM+23, GLL+22]) have improved the communication complexity. The work of Fitzi and Garay [FG03] also considered *strong* (A-)BA. Here we require "strong" validity: the common output must have been one of the honest parties' inputs (note that this is equivalent to standard validity in the binary case). When the size of the input domain is $m > 2$, neither a Turpin-Coan-style reduction [TC84] nor the obvious approach of running $\log m$ parallel executions of a binary protocol would suffice to realize this stronger notion of agreement; indeed, Fitzi and Garay showed that strong A-BA is possible if and only if $t < n/(m+1)$. This bound holds in both the information-theoretic and computational settings. Their unconditionally secure asynchronous protocol actually involves oblivious coin flipping on the domain, but $m > 2$ forces $t < n/4$ and the binary asynchronous OCC protocols in [Fel89, CR93] can be extended to multi-valued in this regime, as discussed in Section 1.2.

1.4 Organization of the Paper

The rest of the paper is organized as follows. We start in Section 2 with some preliminaries, including a brief overview of the UC framework. In Section 3, we discuss our model in greater depth, formally specifying the asynchronous primitives we use as building blocks and motivating our novel modeling choices along the way. Section 4 contains our multi-valued asynchronous OCC construction with optimal resiliency $t < n/3$, and its security proof. Finally, in Section 5 we present the impactful result that our OCC construction enables: a simplified and sound protocol for concurrent A-BA in expected-constant rounds. Appendix A contains our attack on the asynchronous construction in [BE03] (along with the required background material), and Appendix B demonstrates that an asynchronous version of the binary to multi-valued BA extension in [TC84] works **if and only if** $t < n/5$.

2 Model and Preliminaries

For $m \in \mathbb{N}$, we use $[m]$ to denote the set $\{1, \ldots, m\}$. Our protocols often use dynamically growing sets S; when $|S| \geq k$, we denote by $S^{(k)} \subseteq S$ the set containing the first k elements that were added to S. We also denote by $\mathbb{F}$ the finite field from which the secrets shared in our protocols are drawn; its size is fixed by the calling protocol, and is instantiated in Section 4. Our statements (in particular, the ones about statistical security) assume an (often implicit) security parameter κ which is assumed to be linear in $\max\{\log |\mathbb{F}|, n\}$.

We write our protocols using a series of numbered steps. The interpretation is that a party repeatedly goes through them in sequential order, attempting each instruction. Of course, in the asynchronous setting, the necessary precondition for carrying out an instruction might only be met much later in the protocol, and furthermore messages may be delayed. We discuss the latter below, in Section 2.2. Regarding the former, we frequently use statements of the form "Wait until [condition]. Then, [instruction]," which is to be executed at most once, or "If [condition], then [instruction]," which can be executed multiple times.

2.1 UC Basics

We prove our constructions secure in the UC framework [Can20] and we briefly summarize it here. Protocol machines, ideal functionalities, the adversary, and the environment are all modeled as interactive Turing machine (ITM) instances, or ITIs. An execution of protocol Π consists of a series of activations of ITIs, starting with the environment $\mathcal{Z}$ who provides inputs to and collects outputs from the parties and the adversary $\mathcal{A}$; parties can also give input to and collect output from sub-parties (e.g., hybrid functionalities), and $\mathcal{A}$ can communicate with parties via "backdoor" messages. Although parties can send messages to one another, $\mathcal{A}$ is responsible for delivering them, making this model *completely* asynchronous (see Section 2.2 for the way we capture eventual-delivery channels). Protocol instances are delineated using a unique session identifier (SID).

Corruption of parties is modeled by a special `corrupt` message sent from $\mathcal{A}$ to the party; upon receipt of this message, the party sends its entire local state to $\mathcal{A}$, and in all future activations follows the instructions of $\mathcal{A}$. We emphasize that corruptions can occur at any point during the protocol (i.e., $\mathcal{A}$ is assumed to be adaptive). Denote by $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ the probability distribution ensemble corresponding to the (binary) output of $\mathcal{Z}$ at the end of an execution of Π with adversary $\mathcal{A}$.

The ideal-world process for functionality $\mathcal{F}$ is simply defined as an execution of the ideal protocol $\text{IDEAL}_{\mathcal{F}}$, in which the so-called "dummy" parties just forward inputs from $\mathcal{Z}$ to $\mathcal{F}$ and forward outputs from $\mathcal{F}$ to $\mathcal{Z}$ (in particular, the dummy parties do not communicate with the adversary, but rather the adversary is expected to send backdoor messages directly to $\mathcal{F}$, including corruption requests). Our functionalities implicitly respond to corruption requests in the standard way (e.g., all previous inputs of the newly corrupted party are leaked, and the adversary has the option to replace its output); we refer to [Can20] for further details. The adversary in the ideal world is also called the *simulator* and denoted $\mathcal{S}$; the corresponding ensemble is denoted by $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

We are interested in unconditional (aka information-theoretic) security. Thus, we say that a protocol Π (statistically) *UC-realizes* an ideal functionality $\mathcal{F}$ if for any computationally unbounded adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ (which is polynomial in the complexity of $\mathcal{A}$), such that for any computationally unbounded environment $\mathcal{Z}$, we have $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. In case the distribution ensembles are perfectly indistinguishable we say that Π UC-realizes $\mathcal{F}$ with *perfect* security.

When Π is a $(\mathcal{G}_1,\ldots,\mathcal{G}_\ell)$-hybrid protocol (i.e., making subroutine calls to $\text{IDEAL}_{\mathcal{G}_1},\ldots,\text{IDEAL}_{\mathcal{G}_\ell}$), we say that Π UC-realizes $\mathcal{F}$ in the $(\mathcal{G}_1,\ldots,\mathcal{G}_\ell)$-hybrid model.

2.2 Modeling Eventual Message Delivery in UC

As mentioned above, the base communication model in UC is completely unprotected. To capture asynchronous networks with eventual message delivery, we work in a hybrid model with access to the multi-use *asynchronous secure message transmission* functionality $\mathcal{F}_{\mathsf{a\text{-}smt}}$, shown in Figure 1, which was introduced in [CGHZ16] and is itself based on the (single-use) *eventual-delivery secure*

channel functionality $\mathcal{F}_{\mathsf{ed\text{-}sec}}$ from [KMTZ13].

Functionality $\mathcal{F}_{\mathsf{a\text{-}smt}}$

The functionality proceeds as follows. At the first activation, verify that $\mathsf{sid} = (P_s, P_r, \mathsf{sid}')$, where P_s and P_r are player identities. Initialize the current delay $D := 1$ and the message buffer $M := (\mathtt{eom})$, where $\mathtt{eom}$ is a special "end-of-messages" symbol.

- Upon receiving $(\mathtt{send}, \mathsf{sid}, m)$ from P_s, update $M := (m, \mathsf{mid}) \parallel M$ (i.e., insert (m, mid) at the beginning of M), where mid is a unique message ID, and send $(\mathtt{leakage}, \mathsf{sid}, \mathsf{mid})$ to the adversary.
- Upon receiving $(\mathtt{delay}, \mathsf{sid}, T)$ from the adversary for $T \in \mathbb{Z}$ represented in unary notation, update $D := \max(1, D + T)$ and send $(\mathtt{delay\text{-}set}, \mathsf{sid})$ to the adversary (thereby returning the activation to the adversary).
- Upon receiving $(\mathtt{fetch}, \mathsf{sid})$ from P_r, do:
  1. Update $D := \max(0, D - 1)$ and send $(\mathtt{fetched}, \mathsf{sid})$ to the adversary.
  2. If $D = 0$ and $M = ((m_1, \mathsf{mid}_1), \ldots, (m_l, \mathsf{mid}_l), \mathtt{eom})$, then update $M := ((m_2, \mathsf{mid}_2), \ldots, (m_l, \mathsf{mid}_l), \mathtt{eom})$ and send $(\mathtt{output}, \mathsf{sid}, m_1)$ to P_r.
- Upon receiving $(\mathtt{permute}, \mathsf{sid}, \pi)$ from the adversary, if $M = ((m_1, \mathsf{mid}_1), \ldots, (m_l, \mathsf{mid}_l), \mathtt{eom})$ and π is a permutation over $[l]$, then update $M := ((m_{\pi(1)}, \mathsf{mid}_{\pi(1)}), \ldots, (m_{\pi(l)}, \mathsf{mid}_{\pi(l)}), \mathtt{eom})$.
- (Adaptive Message Replacement) Upon receiving $(\mathtt{replace}, \mathsf{sid}, ((m_1, \mathsf{mid}_1), \ldots, (m_{l'}, \mathsf{mid}_{l'})), T')$ from the adversary: If P_s is corrupted, $D > 0$, and T' is a valid delay, then update $M := ((m_1, \mathsf{mid}_1), \ldots, (m_{l'}, \mathsf{mid}_{l'}), \mathtt{eom})$ and $D := \max(1, T')$.

Figure 1: The asynchronous secure message transmission functionality.

The functionality $\mathcal{F}_{\mathsf{a\text{-}smt}}$ models a *secure* eventual-delivery channel between a sender P_s and receiver P_r.[12] To reflect the adversary's ability to delay the message by an arbitrary finite duration (even when P_s and P_r are not corrupted), the functionality operates in a "pull" mode, managing a message buffer M and a counter D that represents the current message delay. This counter is decremented every time P_r tries to fetch a message, which is ultimately sent once the counter hits 0. The adversary can at any time provide an additional integer delay T, and if it wishes to immediately release the messages, it needs only to submit a large negative value. It is important to note that T must be encoded in unary; this ensures that the delay, while arbitrary, remains bounded by the adversary's computational resources or running time.[13] Also, note that $\mathcal{F}_{\mathsf{a\text{-}smt}}$ guarantees eventual message delivery assuming that the environment gives sufficient resources to the protocol, i.e., activates P_r sufficiently many times. We additionally allow the adversary to permute the messages in the buffer; see [CGHZ16] for a discussion.

Since $\mathcal{F}_{\mathsf{a\text{-}smt}}$ (and our other asynchronous functionalities) may not provide output immediately, we must clarify what we mean when instructing a party to fetch the output from multiple instances of a hybrid (e.g., one per party), as repeatedly attempting to fetch from the first instance before continuing to the second instance might not make any progress. Accordingly, we require that parties fetch the output in a "round-robin fashion" across activations (i.e., after attempting to fetch from each instance, the party moves on to try the remaining steps in the protocol). It is implicit that the party stops making fetch requests to any given instance after it receives an output from that instance (of course, this behavior can be easily implemented using flags).

Finally, we note that while the UC framework has no notion of time, and we are in the asynchronous setting (with eventual message delivery) where parties may proceed at different rates, one can still formally define a notion of asynchronous rounds along the lines of [CGHZ16], which will be referred to in our statements. Informally, rounds are delimited by alternations between

[12] Recall that while (concurrent) A-BA is not a private task, secure channels are needed to construct an OCC.

[13] We refer to [Can20] for a formal definition of running time in the UC framework.

sending messages to and reading messages from the network. More precisely, each party maintains a local round counter, which is incremented every time the party sends an input to a hybrid after fetching a (causally related) output from a hybrid. For example—and this accounts for virtually all of the times that a round advances during an execution of our protocols—a successful fetch request may directly trigger a send in the next activation, in which case the party increments its round counter. Note that we allow for multiple sends to be "batched" within the same round (i.e., only the first send will trigger a round advancement). We refrain from putting forth a formal treatment of causality; notwithstanding, causal relationships between messages will be immediately apparent from our protocol descriptions.

Denote by $R_{\mathcal{A},\mathcal{Z},z}$ the random variable (defined over the random choices of all relevant machines) representing the maximum number of rounds that an honest party takes in an execution with a *fixed* adversary $\mathcal{A}$, environment $\mathcal{Z}$, and input z to $\mathcal{Z}$. We now define the (expected) round complexity of the protocol by considering the maximum expected value of this random variable, over all admissible adversaries, environments, and inputs: $\max_{\mathcal{A},\mathcal{Z},z}\{\mathbb{E}[R_{\mathcal{A},\mathcal{Z},z}]\}$.

3 Ideal Functionalities for a Few Standard Primitives

In this section, we present ideal functionalities for asynchronous primitives used in our constructions. This seemingly simple task requires careful consideration of certain aspects, as the adversary's ability to delay messages in the network has an upstream impact on the achievable security guarantees of distributed tasks. In particular, the adversary can obstruct the output release procedure and impede honest parties' participation. To model *delayed output release*, we extend the mechanism discussed in Section 2.2 (using per-party delay counters). However, to model *delayed participation*, we introduce a novel and more natural approach that addresses limitations of prior work and more closely captures the effects of asynchrony. We start by discussing in Section 3.1 our design choices for the latter, and proceed to describe a few ideal functionalities in Section 3.2.

3.1 Modeling Delayed Participation

In the asynchronous setting, honest parties cannot distinguish whether uncooperative parties are corrupted and intentionally withholding messages, or if they are honest parties whose messages have been delayed. Consequently, when the number of corruptions is upper-bounded by t, waiting for the participation of the last t parties can result in an indefinite wait. In ideal functionalities, this translates to expecting participation and/or input only from a subset of adversarially chosen parties, known as the "core set," with a size of $n - t$. Observe that the value of t is closely related to the behavior of the functionality. In this work, we specifically consider optimal resiliency, where $t = \lceil \frac{n}{3} \rceil - 1$. However, our functionalities can be easily adapted to accommodate other resiliency bounds.

In our modeling approach, the adversary implicitly determines the core set by strategically delaying participation (or input submission) to the functionality. If we instruct the ideal functionality to proceed once $n - t$ parties have participated, the adversary can precisely determine the core set by manipulating the order of participation (or input submissions). To accommodate arbitrary but finite delays for input submissions, we employ a technique similar to the one we use for the output-release mechanism. That is, in addition to a per-party *output delay counter*, there is an *input delay counter* (updatable by the adversary) which is decremented every time the party pings the functionality; once the counter reaches zero, the party is allowed to participate.

This approach contrasts with the work of Cohen [Coh16] and Coretti *et al.* [CGHZ16], wherein the adversary (simulator) **explicitly** sends a core set to the functionality. Obtaining the core set

from the adversary all at once does not accurately mimic real-world executions and requires careful consideration to ensure the implementation works in all scenarios. For instance, a challenging case to model is when the simulator sets the core set, but the environment never activates some parties in that core set. If not handled properly, this situation can lead to either the ideal functionality stalling indefinitely while the real-world execution proceeds, or allowing for core sets of smaller sizes, neither of which is acceptable.

There are other aspects of modeling the core set that can potentially cause issues. If the simulator is required to set the core set early on, it may encounter issues during the simulation because in some real-world protocols, such as the asynchronous MPC protocol of Ben-Or *et al.* [BKR94], the core set is not fixed in the early stages of the execution. Similarly, if the functionality allows late submission of the core set, then when using the functionality as a hybrid, the adversary can potentially stall the functionality unless appropriate preventive mechanisms are in place. In contrast, implicitly defining the core set by delaying parties' participation aligns more closely with real-world executions, reducing the probability of errors. Additionally, it can potentially simplify the simulation process, as the simulator can gradually define the core set as the protocol progresses.

3.2 Ideal Functionalities for a Few Standard Primitives

We now cast a few standard asynchronous primitives as UC functionalities, following our novel modeling approach. We also present security statements showing how classical protocols can be used to realize these primitives.

Asynchronous Broadcast (A-Cast). The first essential primitive used in both our OCC and concurrent A-BA protocols, which also finds numerous applications in other asynchronous protocols, is *Bracha's Asynchronous Broadcast* (A-Cast) [Bra87]. A-Cast enables a distinguished sender to distribute its input, such that if an honest party outputs a value, then all honest parties must (eventually) output the same value. Moreover, if the sender is honest, then all honest parties must (eventually) output the sender's input. While these are essentially the agreement and validity properties required from regular (synchronous) broadcast, we stress that honest parties may not terminate when the sender is corrupted. We formulate A-Cast as the ideal functionality $\mathcal{F}_{\mathsf{a\text{-}cast}}$, shown in Figure 2.

Note that although A-Cast is a single-sender primitive, assuming it can proceed to the output generation phase without sufficient participation from other parties is too idealized. A realizable functionality should only proceed to the output generation phase when $n - t$ parties (which may include the sender) have participated. Non-sender parties demonstrate their participation by issuing fetch requests to the functionality. An important technicality here is that the participation of parties before the sender initiates the session should not contribute to the count. Therefore, $\mathcal{F}_{\mathsf{a\text{-}cast}}$ starts a session only once the input from the sender is received. Consequently, any efforts for participation before that point will not be taken into account. An implication of this design choice when using $\mathcal{F}_{\mathsf{a\text{-}cast}}$ as a hybrid is that parties other than the sender will not know when the session has started.

Bracha's asynchronous broadcast protocol [Bra87] can be used to UC-realize $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with perfect security. We formally state this result in the following proposition, whose proof is straightforward.

Proposition 1. *$\mathcal{F}_{\mathsf{a\text{-}cast}}$ can be UC-realized with perfect security in the $\mathcal{F}_{\mathsf{a\text{-}smt}}$-hybrid model, in constant rounds and against an adaptive and malicious t-adversary, provided $t < \frac{n}{3}$.*

Asynchronous Verifiable Secret Sharing (A-VSS). Another crucial primitive we require, mainly for our OCC protocol, is *Asynchronous Verifiable Secret Sharing* (A-VSS). A-VSS allows a

Functionality $\mathcal{F}_{\text{a-cast}}$

The functionality proceeds as follows. Start a session once a message of the form $(\texttt{send}, \mathsf{sid}, \cdot)$ with $\mathsf{sid} = (\mathcal{P}, P_s, \mathsf{sid}')$ is received from (the sender) $P_s \in \mathcal{P}$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize $\mathsf{participated}_i := 0$ and delay values $D_i^{\text{input}} = D_i^{\text{output}} := 1$. Also initialize y to a default value $\perp$ and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\texttt{delay}, \mathsf{sid}, P_i, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{type} \in \{\text{input}, \text{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{type}} := \max(1, D_i^{\mathsf{type}} + D)$ and send $(\texttt{delay-set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\texttt{send}, \mathsf{sid}, v)$ from $P_i = P_s$ (or the adversary on behalf of corrupted P_s), run the Input Submission Procedure, record $y := v$, and send $(\texttt{leakage}, \mathsf{sid}, v)$ to the adversary.
- Upon receiving $(\texttt{replace}, \mathsf{sid}, v)$ from the adversary (v can be $\perp$), if P_s is corrupted and no party has received their output yet, then update $y := v$.
- Upon receiving $(\texttt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\texttt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $\mathsf{participated}_i = 0$ then do:

1. Update $D_i^{\text{input}} := D_i^{\text{input}} - 1$.
2. If $D_i^{\text{input}} = 0$ then set $\mathsf{participated}_i := 1$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathsf{participated}_j \geq n - t$ and $y \neq \perp$, then do:

1. Update $D_i^{\text{output}} := D_i^{\text{output}} - 1$.
2. If $D_i^{\text{output}} = 0$ then set $(\texttt{output}, \mathsf{sid}, y)$ to be sent to P_i.

Figure 2: The A-Cast functionality.

dealer to secret-share a value among all parties, ensuring that no unauthorized subset of colluding parties can learn any information about the secret. However, any authorized subset of parties should be able to efficiently reconstruct the secret using their shares. The term "verifiable" reflects that the dealer cannot cheat, for example by causing the reconstruction to fail or by inducing inconsistent output values from honest parties. In particular, whenever the sharing phase succeeds, any authorized subset of parties should be able to efficiently complete the reconstruction phase, and all honest parties doing so must recover the same secret. Moreover, if the dealer is honest, the sharing phase must always succeed, and everyone should recover the value originally shared by the dealer. In our context, we consider only the threshold access structure, where a subset of parties can recover the secret if and only if it contains at least $n - t$ of the parties. However, we leak the secret to the adversary as soon as $t + 1$ parties have participated (at least one of whom must be honest). The formulation of A-VSS as an ideal functionality, $\mathcal{F}_{\text{a-vss}}$, is shown in Figure 3. We denote this as the "plain" A-VSS functionality, as it does not allow homomorphic computation over shares; nonetheless, this basic functionality suffices for obtaining a multi-valued asynchronous OCC.

A-VSS, being a single-sender primitive, also requires the participation of at least $n - t$ parties to be realizable. We adopt a similar approach to $\mathcal{F}_{\text{a-cast}}$ in modeling this requirement. Parties other than the dealer demonstrate their participation in the sharing phase by issuing fetch requests, while participation in the reconstruction phase additionally requires sending a (dummy) input message. This ensures that the minimum participation threshold is met for the A-VSS protocol to proceed with each phase. Also, it is important to note that $\mathcal{F}_{\text{a-vss}}$ only initiates a session once it receives the first input (share) from the dealer. Any participation efforts made before that point are not taken into account.

The A-VSS protocol given by Canetti and Rabin in [CR93] can be used to UC-realize $\mathcal{F}_{\text{a-vss}}$ with statistical security for $t < n/3$. This result is formally stated in the following proposition. It

Functionality $\mathcal{F}_{\mathsf{a\text{-}vss}}^{V}$

The functionality is parameterized by a set V of possible secrets and proceeds as follows. Start a session once a message of the form $(\mathtt{share}, \mathsf{sid}, v)$ with $\mathsf{sid} = (\mathcal{P}, P_d, \mathsf{sid}')$ and $v \in V$ is received from (the dealer) $P_d \in \mathcal{P}$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize $\mathsf{phase}_i := \text{share}$, $\mathsf{participated}_{i,\text{share}} = \mathsf{participated}_{i,\text{rec}} := 0$, and current delay values $D_i^{\text{share,input}} = D_i^{\text{share,output}} = D_i^{\text{rec,input}} = D_i^{\text{rec,output}} := 1$. Also initialize secret to a default value $\perp$ and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\mathtt{delay}, \mathsf{sid}, P_i, \mathsf{phase}, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{phase} \in \{\text{share}, \text{rec}\}$, $\mathsf{type} \in \{\text{input}, \text{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{phase,type}} := \max(1, D_i^{\mathsf{phase,type}} + D)$ and send $(\mathtt{delay\text{-}set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\mathtt{share}, \mathsf{sid}, v)$ from $P_i = P_d$ (or the adversary on behalf of corrupted P_d) for $v \in V$, run the Input Submission Procedure, record $\mathsf{secret} := v$, and send $(\mathtt{leakage}, \mathsf{sid}, |\mathsf{secret}|)$ to the adversary.
- Upon receiving $(\mathtt{replace}, \mathsf{sid}, v)$ from the adversary, if P_d is corrupted, $v \in V \cup \{\perp\}$, and no honest party has received $(\mathtt{shared}, \mathsf{sid})$ yet, then update $\mathsf{secret} := v$.
- Upon receiving $(\mathtt{rec}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission Procedure, and if $\sum_{j=1}^{n} \mathsf{participated}_{j,\text{rec}} \geq t + 1$ then send $(\mathtt{leakage}, \mathsf{sid}, P_i, \mathsf{secret})$ to the adversary; else send $(\mathtt{fetched}, \mathsf{sid}, P_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\mathtt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $\mathsf{participated}_{i,\mathsf{phase}_i} = 0$, and either $\mathsf{phase}_i = \text{share}$ or $\mathsf{phase}_i = \text{rec}$ and $(\mathtt{rec}, \mathsf{sid})$ was previously received from P_i, then do:

1. Update $D_i^{\mathsf{phase}_i,\text{input}} := D_i^{\mathsf{phase}_i,\text{input}} - 1$.
2. If $D_i^{\mathsf{phase}_i,\text{input}} = 0$ then set $\mathsf{participated}_{i,\mathsf{phase}_i} := 1$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathsf{participated}_{j,\mathsf{phase}_i} \geq n - t \wedge (\mathsf{phase}_i = \text{share} \vee \mathsf{secret} \neq \perp)$ then do:

1. Update $D_i^{\mathsf{phase}_i,\text{output}} := D_i^{\mathsf{phase}_i,\text{output}} - 1$.
2. If $D_i^{\mathsf{phase}_i,\text{output}} = 0$, then do the following. If $\mathsf{phase}_i = \text{share}$, set $(\mathtt{shared}, \mathsf{sid})$ to be sent to P_i and update $\mathsf{phase}_i := \text{rec}$; otherwise, set $(\mathtt{opened}, \mathsf{sid}, \mathsf{secret})$ to be sent to P_i.

Figure 3: The plain A-VSS functionality.

is worth noting that perfectly secure A-VSS is impossible for $t \geq n/4$ [BKR94, ADS22].

Proposition 2. *For any finite field $\mathbb{F}$ (with $|\mathbb{F}| > n$), $\mathcal{F}_{\mathsf{a\text{-}vss}}^{\mathbb{F}}$ can be UC-realized with statistical security in the $\mathcal{F}_{\mathsf{a\text{-}smt}}$-hybrid model, in constant rounds and against an adaptive and malicious t-adversary, provided $t < \frac{n}{3}$.*

Asynchronous Byzantine Agreement (A-BA). We use A-BA for both binary and multi-valued domains V in our revised concurrent A-BA protocol. In this primitive, each party P_i has an input $v_i \in V$. The goal is for all honest parties to output the same value, such that if $n - 2t$ input values are the same, that value (which for well-definedness we require is unique) is chosen as the output; otherwise, the adversary determines the output. We initially consider *corruption-unfair* A-BA, where the adversary learns the input of each party the moment it is provided. Corruption fairness, as introduced in [HZ10] and later coined in [CGZ23], essentially ensures that the (adaptive) adversary cannot corrupt a party and subsequently influence the input value of that party based on its original input. It is worth noting that corruption-fair A-BA can easily be defined by avoiding leaking honest parties' inputs before the output is generated. We formulate A-BA as the ideal functionality $\mathcal{F}_{\mathsf{a\text{-}ba}}$, shown in Figure 4.

The functionality $\mathcal{F}_{\mathsf{a\text{-}ba}}$ encompasses an additional property known as "non-intrusion" validity

Functionality $\mathcal{F}_{\mathsf{a\text{-}ba}}^{V}$

The functionality is parameterized by a set V of values and proceeds as follows. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize v_i to a default value $\perp$, $\mathsf{participated}_i := 0$, and delay values $D_i^{\text{input}} = D_i^{\text{output}} := 1$. Also initialize $y := \perp'$, $a := \perp$, and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\texttt{delay}, \mathsf{sid}, P_i, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{type} \in \{\text{input}, \text{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{type}} := \max(1, D_i^{\mathsf{type}} + D)$ and send $(\texttt{delay-set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\texttt{input}, \mathsf{sid}, v)$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission Procedure and send $(\texttt{leakage}, \mathsf{sid}, P_i, v)$ to the adversary.
- Upon receiving $(\texttt{replace}, \mathsf{sid}, v)$ from the adversary, record $a := v$.
- Upon receiving $(\texttt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\texttt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $\mathsf{participated}_i = 0$ and $(\texttt{input}, \mathsf{sid}, v)$ was previously received from P_i, then do:

1. Update $D_i^{\text{input}} := D_i^{\text{input}} - 1$.
2. If $D_i^{\text{input}} = 0$, then set $\mathsf{participated}_i := 1$ and record $v_i := v$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathsf{participated}_j \geq n - t$ then do:

1. Update $D_i^{\text{output}} := D_i^{\text{output}} - 1$.
2. If $D_i^{\text{output}} = 0$, then do the following. If $y = \perp'$, then set $y := x$ if there exists a unique value $x \in V$ such that $x = v_k$ for at least $n - 2t$ inputs v_k; else if $a = \perp$ or $a \in V$ was provided by at least one honest party, then set $y := a$; else set $y := \perp$. Additionally, set $(\texttt{output}, \mathsf{sid}, y)$ to be sent to P_i.

Figure 4: The (intrusion-tolerant) A-BA functionality.

(see Section 1.2). In a nutshell, this property guarantees that no malicious value can be present in the output. In other words, the output must be either an honest party's input or a default value $\perp$. This stronger notion of A-BA is vital for the security of our concurrent A-BA protocol.

Note that when the input domain V already contains a "null" value, $\mathcal{F}_{\mathsf{a\text{-}ba}}$ can simply output it as the default value. Slightly abusing notation, we can therefore run multi-valued A-BA on some domain $V' \cup \{\perp\}$, with the assurance that a non-default (i.e., *meaningful*) output value must have originated with an honest party. (We stress that, on the other hand, an output of $\perp$ would guarantee nothing in that scenario, even though $\perp$ is a possible input value.) Looking ahead, our concurrent A-BA protocol will make use of this behavior, by setting V' to be the set of all output vectors and allowing an honest party who did not receive a vector to input $\perp$.

The expected-constant-round binary A-BA protocol of Canetti and Rabin [CR93] can be used to UC-realize $\mathcal{F}_{\mathsf{a\text{-}ba}}^{V}$ with statistical security for binary domains ($|V| = 2$) and $t < n/3$. However, our concurrent A-BA protocol (and the one in [BE03]) require **multi-valued** A-BA, where $|V|$ is not constant (in fact, exponential), in expected-constant rounds. Ben-Or and El-Yaniv [BE03] claim that the constant-round reduction of multi-valued to binary BA proposed by Turpin and Coan [TC84], which works in the synchronous setting for $t < n/3$, can be extended to work in the asynchronous setting by using A-Cast for message distribution. However, we demonstrate that the asynchronous version of this reduction works if and only if $t < n/5$, even when using A-Cast and considering a static adversary. (See Appendix B for details.) This already reflects a modification that we introduce; the direct translation of the reduction works if and only if $t < n/6$. In either case, achieving optimal resiliency is not straightforward.

More recently, Mostéfaoui and Raynal [MR17] presented a constant-round transformation from binary to multi-valued A-BA that works for $t < n/3$. Furthermore, the resulting protocol satisfies

the non-intrusion validity property mentioned above. By applying this transformation to the binary A-BA protocol of Canetti and Rabin [CR93], we can UC-realize $\mathcal{F}_{\mathsf{a\text{-}ba}}^{V}$ for arbitrary V with statistical security:

Proposition 3. *For any domain V, $\mathcal{F}_{\mathsf{a\text{-}ba}}^{V}$ can be UC-realized with statistical security in the $\mathcal{F}_{\mathsf{a\text{-}smt}}$-hybrid model, in expected-constant rounds and against an adaptive and malicious t-adversary, provided $t < \frac{n}{3}$.*

4 Asynchronous Oblivious Common Coin

The oblivious common coin (OCC) is a crucial primitive, particularly in the asynchronous setting. As discussed in the Introduction, achieving even binary A-BA against a single fail-stop corruption becomes impossible without the use of randomization [FLP85], emphasizing the importance of randomization in the asynchronous setting. The OCC primitive allows parties to collectively agree on a random value with a constant probability. This property opens up possibilities for various higher-level asynchronous protocols, as an OCC can potentially serve as the building block that provides the necessary randomization. To illustrate its significance, in the case of binary values, OCC enables binary A-BA in expected-constant rounds [Rab83], and when extended to domains with size equal to the number of participants, it can be used for oblivious leader election (OLE) and sets the stage for expected-constant-round concurrent BA in both synchronous and asynchronous networks [BE03]. Refer to Section 5.2 for a formal definition of asynchronous OLE (A-OLE) and its application toward expected-constant-round concurrent A-BA.

As highlighted in the Introduction, none of the existing OCC proposals is simultaneously information-theoretic, asynchronous, multi-valued, and optimally resilient. Furthermore, no straightforward adaptation of the existing schemes yields an OCC with all of these properties. In Section 4.1, we first formulate (multi-valued) OCC as an ideal functionality in the asynchronous setting with eventual delivery. Subsequently, in Section 4.2, we propose our own OCC protocol that aims to satisfy all of these properties. In Section 4.3, we prove that our protocol runs in constant time and UC-realizes the (multi-valued) OCC functionality with perfect security in the asynchronous setting, even against an adaptive and malicious adversary who corrupts fewer than 1/3 of the parties.

At a high level, our protocol is based on the binary OCC of Feldman [Fel89] and Canetti and Rabin [CR93], and incorporates a novel combinatorial technique derived from our observation stated in Lemma 4 below. By leveraging this lemma, we unveil interesting and powerful properties of the local views formed during the protocol's execution, leading to enhanced extraction capabilities. In fact, instead of extracting a single bit, by choosing appropriate parameters we can extract random values from any arbitrary domain still with a constant probability.

4.1 A-OCC Ideal Functionality

An OCC is parameterized by a set V and some constant probability $p > 0$. Each party starts with an empty input λ and outputs a value from V, where with probability at least p all parties output the same uniformly random value $x \in V$ and with probability $1 - p$ the adversary chooses each party's output.[14]

The above goal can be translated to a UC functionality as follows: Initially, the ideal functionality samples a "fairness bit" $b \leftarrow \text{Bernoulli}(p)$ and a random value $y \xleftarrow{R} V$. Then, if $b = 1$ or no

[14] It is important to note that the term "oblivious" in this context refers to the fact that parties do not learn whether an agreement on a random coin value has been achieved or not, while the adversary does.

meaningful input is received from the adversary, it outputs y to every party. However, if $b = 0$ and meaningful input is received from the adversary, it assigns each party the value provided by the adversary; note that in this case, we allow the adversary to assign outputs gradually. The functionality also informs the adversary about the fairness bit and the random value once at least $t+1$ parties have participated (at least one of whom must be honest). Asynchronous aspects, including delayed output release and participation, are handled as in Section 3. The resulting functionality is shown in Figure 5.

Functionality $\mathcal{F}_{\mathsf{a\text{-}occ}}^{V,p}$

The functionality is parameterized by a set V of possible outcomes and a fairness probability p, and it proceeds as follows. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize y_i to a default value $\perp$, $\mathsf{participated}_i := 0$, and delay values $D_i^{\text{input}} = D_i^{\text{output}} := 1$. Also initialize a, y, and b to $\perp$, and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\texttt{delay}, \mathsf{sid}, P_i, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{type} \in \{\text{input}, \text{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{type}} := \max(1, D_i^{\mathsf{type}} + D)$ and send $(\texttt{delay-set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\texttt{input}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission Procedure, and send $(\texttt{leakage}, \mathsf{sid}, P_i)$ to the adversary. Moreover, if $\sum_{j=1}^{n} \mathsf{participated}_j \geq t+1$, then additionally send $(\texttt{reveal}, \mathsf{sid}, b, y)$ to the adversary.
- Upon receiving $(\texttt{replace}, \mathsf{sid}, v)$ from the adversary, record $a := v$.
- Upon receiving $(\texttt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\texttt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $\mathsf{participated}_i = 0$ and $(\texttt{input}, \mathsf{sid})$ was previously received from P_i, then do:

1. Update $D_i^{\text{input}} := D_i^{\text{input}} - 1$.
2. If $D_i^{\text{input}} = 0$ then set $\mathsf{participated}_i := 1$.
3. If $b = \perp$ then sample a "fairness bit" $b \leftarrow \text{Bernoulli}(p)$ and a random value $y \xleftarrow{\text{R}} V$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathsf{participated}_j \geq n - t$ then do:

1. Update $D_i^{\text{output}} := D_i^{\text{output}} - 1$.
2. If $D_i^{\text{output}} = 0$, then do the following. If $y_i = \perp$, then if $b = 1$ or a cannot be parsed as $(a_1, \ldots, a_n) \in V^n$, set $y_i := y$; otherwise, set $y_i := a_i$. Additionally, set $(\texttt{output}, \mathsf{sid}, y_i)$ to be sent to P_i.

Figure 5: The asynchronous OCC functionality.

4.2 The A-OCC Protocol

We proceed to present our asynchronous and multi-valued OCC protocol. We begin by discussing all the essential building blocks employed in our protocol. Subsequently, we provide a high-level overview of the protocol, highlighting its key ideas. This is followed by a detailed description.

Building blocks. The basic building blocks of our A-OCC protocol are A-VSS and A-Cast. A-VSS enables parties to contribute by privately providing their local randomness and only revealing this randomness when the contributions to the output are determined. Thus, A-VSS ensures the secrecy and verifiability of the shared secrets in an asynchronous setting. The A-VSS primitive is formally modeled as the ideal functionality $\mathcal{F}_{\mathsf{a\text{-}vss}}$, described in Section 3.2. On the other hand, A-Cast facilitates communication among parties by providing stronger guarantees than simple message distribution. This is especially crucial in asynchronous settings where challenges such

as low message dispersion can occur. A-Cast helps in overcoming these challenges and ensures reliable message dissemination among the parties. We use the ideal functionality $\mathcal{F}_{\mathsf{a\text{-}cast}}$, described in Section 3.2, to model this primitive.

The protocol. As previously mentioned, our multi-valued protocol is built upon existing binary A-OCC constructions [Fel89, CR93] and introduces a novel combinatorial technique for extracting values from arbitrary domains. In both the binary protocol and our proposed protocol, each party secret-shares n random elements from a field. It can be observed that at some point during the protocol execution, a vector of length n consisting of random elements from the same field is established (with up to t missing values due to asynchrony). For each coordinate of this vector, random elements shared by $t+1$ parties are utilized to prevent the adversary, controlling up to t parties, from biasing any specific coordinate. Subsequently, each party starts reconstructing secrets shared by other parties to form the same vector locally. In the asynchronous setting, due to the low dispersion of messages, not all coordinates can be reconstructed by honest parties. This can result in different parties reconstructing different subsets of coordinates. However, by using mechanisms to improve message dispersion, as originally demonstrated by Feldman [Fel89], it has been proven that when $t < n/3$, while the local vectors of honest parties may have up to t missing coordinates, they have an overlap of size at least $n - 2t > n/3$.[15] This is significant because without such mechanisms, and allowing for t missing components when $n = 3t + 1$, the overlap in the local vectors of just four parties could be empty.

In fact, by incorporating the "Gather" protocol in [AJM+23], we can achieve an overlap of size at least $n - t > 2n/3$, at the cost of an extra round of message exchange. While this stronger guarantee is not crucial to our feasibility result, we use it to improve the agreement probability of our OCC, and consequently the concrete round efficiency of our concurrent A-BA protocol.

Traditionally, existing protocols extract a single bit from the local views of the random vector by instructing parties to take all existing coordinates modulo n, outputting 0 if any coordinate is 0 and outputting 1 otherwise. In contrast, our protocol represents a significant improvement by going beyond the extraction of a single bit from the local views of the random vector. This enhanced randomness extraction is through a combinatorial observation regarding vectors of random values, as formulated in Lemma 4. This observation allows the parties to agree non-interactively on certain coordinates of the random vector with a constant probability, while also ensuring that these agreed-upon coordinates lie within their overlap section. The minimum such coordinate is then used to select a common output value from the vector.

Another important observation regarding existing binary A-OCC protocols is the lack of a proper termination mechanism. This poses significant challenges as network delays can cause parties to operate out of sync. In such cases, some parties may receive the output before completing their role in the execution, and if they stop, others may not be able to generate the output at all. This directly affects the simulator's ability to accurately simulate the protocol, especially in managing input and output delays in the ideal functionality. This is mainly because, unlike the protocol, the ideal functionality ensures that once a party receives the output, sufficient participation has occurred, and any other party, regardless of others' participation, can fetch the output if activated sufficiently many times. One potential solution could be invoking A-BA on the output at the end; however, this would create a circular dependency since A-OCC itself is used to achieve A-BA. Instead, we choose to adopt a simpler approach inspired by Bracha's termination mechanism. This

[15] Feldman calculated the size of the overlap, denoted as x, based on the number of participants n and the maximum number of corruptions t. The general relation is $x \geq n - t - \frac{t^2}{n-2t}$, which yields $x > n/3$ and $x > 5n/8$ when $t < n/3$ and $t < n/4$, respectively. This argument was later used in [CR93] to achieve optimal resiliency.

approach resolves the participation issue without causing deadlocks and ensures agreement if all parties initiate the procedure with the same value. The formal description of our multi-valued asynchronous OCC protocol $\Pi_{\mathsf{a\text{-}occ}}$ appears in Figure 6.

4.3 Security Proof

Having described our A-OCC protocol, we proceed to present and prove the formal security statement that demonstrates how the protocol UC-realizes $\mathcal{F}_{\mathsf{a\text{-}occ}}$. However, we first prove a combinatorial observation regarding vectors of random values that facilitates the security proof. We formulate this observation separately in the following lemma, as it may be of independent interest.

Lemma 4. *Let V be a vector of n values chosen independently and uniformly at random from a set S of size $N \in \Theta(n^2)$, and let α be a constant satisfying $0 < \alpha \leq 1$. Then for any subset of indices $I \subseteq [n]$ such that $|I| \geq \alpha n$, with constant probability p there is at least one repeated value in V; moreover, all of the repeated values are constrained to the indices in I.*

Proof. It suffices to consider the case $|I| = \lceil \alpha n \rceil$. We first define the following events. Denote by V_J the restriction of V to the indices in $J \subseteq [n]$.

E_V :	At least two components of V have the same value
E_{V_I} :	At least two components of V_I have the same value
$E_{V_{I^c}}$:	At least two components of V_{I^c} have the same value
$E_{(V_I, V_{I^c})}$:	At least one component of V_I has the same value as a component of V_{I^c}

We now compute

Protocol $\Pi^{V}_{\mathsf{a\text{-}occ}}$

The protocol is parameterized by a set V of possible outcomes. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ for a player set $\mathcal{P}$ of size n. Let $t := \lceil \frac{n}{3} \rceil - 1$ and $m := \mathrm{lcm}(n^2, |V|)$, and let $\mathbb{F}$ be the smallest prime field with size at least m. Party $P_i \in \mathcal{P}$ proceeds as follows. Initialize sets C_i, G_i, R_i, and S_i to $\emptyset$, flag $\mathsf{finished}$ to 0, and variables Z_i and y_i to $\perp$. Also initialize $C'_j := \perp$, $b_{j,k} := 0$, $r_{k,j} := \perp$, $v_j := \perp$, and $z'_j := \perp$ for all $j, k \in [n]$.

- Upon receiving input $(\mathtt{input}, \mathsf{sid})$ from the environment, do the following. For each $j \in [n]$, choose $x_{i,j} \xleftarrow{\mathrm{R}} [m]$ and send $(\mathtt{share}, \mathsf{sid}_i^{\mathrm{share},j}, x_{i,j})$ to an instance of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ with SID $\mathsf{sid}_i^{\mathrm{share},j} := (\mathcal{P}, P_i, \mathsf{sid}', \mathtt{share}, j)$.
- Upon receiving input $(\mathtt{fetch}, \mathsf{sid})$ from the environment, if $\mathsf{finished} = 1$ then output $(\mathtt{output}, \mathsf{sid}, y_i)$ to the environment; else do:

1. For each $j, k \in [n]$, fetch the share from the instance of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ with SID $\mathsf{sid}_j^{\mathrm{share},k}$. Upon receiving back $(\mathtt{shared}, \mathsf{sid}_j^{\mathrm{share},k})$ set $b_{j,k} := 1$, and if $b_{j,l} = 1$ for all $l \in [n]$ then update $C_i := C_i \cup \{P_j\}$.
2. Wait until $|C_i| \geq t{+}1$. Then, send $(\mathtt{send}, \mathsf{sid}_i^{\mathrm{attach}}, C_i^{(t+1)})$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_i^{\mathrm{attach}} := (\mathcal{P}, P_i, \mathsf{sid}', \mathtt{attach})$.
3. For each $j \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_j^{\mathrm{attach}}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}_j^{\mathrm{attach}}, Y)$ where $|Y| = t + 1$, record $C'_j := Y$ and wait until $C'_j \subseteq C_i$; then update $G_i := G_i \cup \{P_j\}$.
4. Wait until $|G_i| \geq n - t$. Then, send $(\mathtt{send}, \mathsf{sid}_i^{\mathrm{set}}, G_i^{(n-t)})$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_i^{\mathrm{set}} := (\mathcal{P}, P_i, \mathsf{sid}', \mathtt{set})$.
5. For each $j \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_j^{\mathrm{set}}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}_j^{\mathrm{set}}, G'_j)$ where $|G'_j| = n - t$, wait until $G'_j \subseteq G_i$; then update $R_i := R_i \cup \{P_j\}$.
6. Wait until $|R_i| \geq n{-}t$. Then, send $(\mathtt{send}, \mathsf{sid}_i^{\mathrm{ready}}, R_i^{(n-t)})$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_i^{\mathrm{ready}} := (\mathcal{P}, P_i, \mathsf{sid}', \mathtt{ready})$.
7. For each $j \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_j^{\mathrm{ready}}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}_j^{\mathrm{ready}}, R'_j)$ where $|R'_j| = n - t$, wait until $R'_j \subseteq R_i$; then update $S_i := S_i \cup \{P_j\}$.
8. Wait until $|S_i| \geq n - t$. Then, record $Z_i := G_i$.
9. If $|S_i| \geq n - t$, then for each $P_j \in G_i$ and each $P_k \in C'_j$, send $(\mathtt{rec}, \mathsf{sid}_k^{\mathrm{share},j})$ to the instance of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ with SID $\mathsf{sid}_k^{\mathrm{share},j}$.[a]
10. For each $P_j \in G_i$ and each $P_k \in C'_j$, fetch the secret from the instance of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ with SID $\mathsf{sid}_k^{\mathrm{share},j}$. Upon receiving back $(\mathtt{opened}, \mathsf{sid}_k^{\mathrm{share},j}, s)$ record $r_{k,j} := s$, and if $r_{l,j} \neq \perp$ for all $P_l \in C'_j$ then set $v_j := \sum_{P_l \in C'_j} r_{l,j} \bmod m$.
11. Wait until $Z_i \neq \perp$ and $v_j \neq \perp$ for all $P_j \in Z_i$. Then, let $J := \{j \in [n] \mid v_j \neq \perp\}$, let $S := \{j \in J \mid v_j \equiv v_k \mod n^2 \text{ for some } k \in J \setminus \{j\}\}$, and do the following. Let $z_i := v_{\min\{S\}} \bmod |V|$ (if $S = \emptyset$ then let z_i be an arbitrary value from V) and send $(\mathtt{send}, \mathsf{sid}_i^{\mathrm{term}}, z_i)$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_i^{\mathrm{term}} := (\mathcal{P}, P_i, \mathsf{sid}', \mathtt{term})$.
12. For each $j \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}_j^{\mathrm{term}}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}_j^{\mathrm{term}}, x)$, record $z'_j := x$.
13. Wait until $z'_j \neq \perp$ for at least $n - t$ indices $j \in [n]$. Then, set $\mathsf{finished} := 1$ and y_i to the mode (most repeated value) among all $z'_j \neq \perp$, breaking ties by a fixed order on V.

[a] Note that reconstruction inputs can be sent outside of the initial batch, if a party gets added to G_i later, but only one reconstruction input is ever sent per instance of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$.

Figure 6: The multi-valued asynchronous OCC protocol.

$$
\begin{aligned}
p &= \Pr[E_{V_I} \cap E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}] \\
&= \Pr[E_{V_I} \mid E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}] \Pr[E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}] \\
&\geq \Pr[E_{V_I} \mid E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}] \Pr[E^{\mathrm{c}}_{V}] && \left(E^{\mathrm{c}}_{V} \subseteq E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}\right) \\
&= \left(1 - \Pr[E^{\mathrm{c}}_{V_I} \mid E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}]\right) \Pr[E^{\mathrm{c}}_{V}] \\
&= \left(1 - \frac{\Pr[E^{\mathrm{c}}_{V_I} \cap E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}} \cap E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})}]}{\Pr[E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})} \mid E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}}] \Pr[E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}}]}\right) \Pr[E^{\mathrm{c}}_{V}] \\
&= \left(1 - \frac{\Pr[E^{\mathrm{c}}_{V}]}{\Pr[E^{\mathrm{c}}_{(V_I,V_{I^{\mathrm{c}}})} \mid E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}}] \Pr[E^{\mathrm{c}}_{V_{I^{\mathrm{c}}}}]}\right) \Pr[E^{\mathrm{c}}_{V}] \\
&= \left(1 - \frac{\prod_{i=0}^{n-1}\left(1 - \frac{i}{N}\right)}{\left(1 - \frac{n-\lceil \alpha n \rceil}{N}\right)^{\lceil \alpha n \rceil} \prod_{i=0}^{n-\lceil \alpha n \rceil - 1}\left(1 - \frac{i}{N}\right)}\right) \prod_{i=0}^{n-1}\left(1 - \frac{i}{N}\right) \\
&\approx \left(1 - \frac{\prod_{i=0}^{n-1}\left(e^{-\frac{i}{N}}\right)}{\left(e^{-\frac{n-\lceil \alpha n \rceil}{N}}\right)^{\lceil \alpha n \rceil} \prod_{i=0}^{n-\lceil \alpha n \rceil - 1}\left(e^{-\frac{i}{N}}\right)}\right) \prod_{i=0}^{n-1}\left(e^{-\frac{i}{N}}\right) && \left(\text{since } \frac{n}{N} \ll 1\right) \\
&= \left(1 - \frac{e^{-\frac{1}{N}\sum_{i=0}^{n-1} i}}{e^{-\frac{n\lceil \alpha n \rceil - \lceil \alpha n \rceil^2}{N}}\left(e^{-\frac{1}{N}\sum_{i=0}^{n-\lceil \alpha n \rceil - 1} i}\right)}\right) e^{-\frac{1}{N}\sum_{i=0}^{n-1} i} \\
&= \left(1 - \frac{e^{-\frac{n^2-n}{2N}}}{e^{-\frac{n\lceil \alpha n \rceil - \lceil \alpha n \rceil^2}{N}}\left(e^{-\frac{n^2+\lceil \alpha n \rceil^2 - 2n\lceil \alpha n \rceil - n + \lceil \alpha n \rceil}{2N}}\right)}\right) e^{-\frac{n^2-n}{2N}} \\
&= \left(1 - e^{-\frac{\lceil \alpha n \rceil^2 - \lceil \alpha n \rceil}{2N}}\right) e^{-\frac{n^2-n}{2N}} \\
&\in \Theta(1). && \left(\text{since } N \in \Theta(n^2)\right) \qquad \square
\end{aligned}
$$

With the groundwork laid out, we now prove the security of protocol $\Pi_{\mathsf{a\text{-}occ}}$:

Theorem 1. *There exists a probability $p \in \Theta(1)$ such that for any finite domain V, protocol $\Pi^{V}_{\mathsf{a\text{-}occ}}$ UC-realizes $\mathcal{F}^{V,p}_{\mathsf{a\text{-}occ}}$ with perfect security in the $(\mathcal{F}_{\mathsf{a\text{-}cast}}, \mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}})$-hybrid model where $\mathbb{F}$ is the smallest prime field of size at least $\mathrm{lcm}(|V|, n^2)$, in constant rounds and in the presence of an adaptive and malicious t-adversary, provided $t < \frac{n}{3}$.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with $\Pi^{V}_{\mathsf{a\text{-}occ}}$ and $\mathcal{A}$, or with $\mathcal{F}^{V,p}_{\mathsf{a\text{-}occ}}$ and $\mathcal{S}$, where p is to be determined later.

The simulator internally runs a copy of $\mathcal{A}$, and plays the roles of $\mathcal{F}_{\mathsf{a\text{-}cast}}$, $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$, and the parties in a simulated execution of the protocol. All inputs from $\mathcal{Z}$ are forwarded to $\mathcal{A}$, and all outputs from $\mathcal{A}$ are forwarded to $\mathcal{Z}$. Moreover, whenever $\mathcal{A}$ corrupts a party in the simulation, $\mathcal{S}$ corrupts the same party in the ideal world by interacting with $\mathcal{F}^{V,p}_{\mathsf{a\text{-}occ}}$, and if the corruption was direct (i.e., not via either of the aiding functionalities), then $\mathcal{S}$ sends $\mathcal{A}$ the party's state and thereafter follows $\mathcal{A}$'s instructions for that party. Moreover, $\mathcal{S}$ manages delays in the ideal functionality based on the delays that $\mathcal{A}$ sets on the hybrids.

The simulated execution commences when $\mathcal{S}$ receives the first message $(\texttt{leakage}, \mathsf{sid}, P_i)$ from $\mathcal{F}_{\mathsf{a\text{-}occ}}$; the values b and y arrive later, in the message $(\texttt{reveal}, \mathsf{sid}, b, y)$ that $\mathcal{F}_{\mathsf{a\text{-}occ}}$ sends once $t+1$ parties have participated, which is well before $\mathcal{S}$ needs them. During the internal execution, $\mathcal{S}$ faithfully simulates the behavior of $\mathcal{F}_{\mathsf{a\text{-}cast}}$. As for $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$, $\mathcal{S}$ can simulate the sharing phase without explicitly assigning any random secrets to honest parties. However, the actual value of each secret must be determined by $\mathcal{S}$ before $n-t$ parties participate in the reconstruction phase of the corresponding $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ instance. It is crucial for $\mathcal{S}$ to select these secrets in such a way that the output matches that of $\mathcal{F}_{\mathsf{a\text{-}occ}}$, while ensuring indistinguishability between the real- and ideal-world views.

We first argue that every tally v_j computed by an honest party P_i is uniformly distributed in $[m]$, and moreover that another honest party P_h cannot compute a *different* tally v'_j for party P_j (although up to t tallies may be "missing" from P_h's view). Recall that P_i computes v_j by summing the reconstructed secrets "attached" to P_j (i.e., the votes cast by the parties in C'_j for P_j). An honest party $P_k \in C'_j$ will have cast a uniform vote $x_{k,j} \in [m]$, but even for a corrupted party a well-defined field element will be reconstructed as its vote, since P_i validated that $C'_j \subseteq C_i$ and C_i contains parties whose secrets were successfully shared to P_i. Moreover, the votes of corrupted parties in C'_j are independent of those cast by honest parties in C'_j, because the secrets attached to P_j are fixed by the time the first honest party P_h "accepts" P_j (i.e., adds P_j to G_h)—due to the guarantees of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$—while no honest party starts reconstructing them until it considers P_j as accepted. Since $|C'_j| = t+1$, it must contain at least one honest party, and therefore P_i will compute a uniform tally $v_j \in [m]$. Furthermore, since P_j A-Casts the set C'_j, all honest parties will agree on the parties contributing to P_j's tally, and since any honest party P_h computing v_j has $C'_j \subseteq C_h$, by the guarantees of $\mathcal{F}^{\mathbb{F}}_{\mathsf{a\text{-}vss}}$ they will all reconstruct the secrets attached to P_j to the same values. Therefore, we can indeed speak of a *fixed* vector of uniform tallies, where each honest party eventually holds a local view of at least $n-t$ coordinates.

By employing an enhanced version of the counting argument in [Fel89, CR93] in the same way as [AJM+23], we can demonstrate that once the first honest party P_i reaches the stage where $|S_i| \geq n-t$, a subset A of parties, with size $n-t > 2n/3$, becomes fixed such that each honest party P_h will (eventually) have $A \subseteq Z_h$. That is, over $2/3$ of the tallies will be visible to all honest parties by the time they are able to generate output. In more detail, assume that the honest party P_i reaches the stage where $s := |S_i| \geq n-t$. We can construct an $s \times n$ table T, where $T_{j,k} = 1$ if and only if $P_j \in S_i$ and $P_k \in R'_j$ (where R'_j was received from P_j's A-Cast).[16] Each row of the table contains (exactly) $n-t$ entries equal to one, indicating that there are a total of $s(n-t)$ one-entries in T. We claim that at least one column must contain at least $t+1$ one-entries. Indeed, suppose to the contrary that each column contains at most t one-entries. Since $n-t \leq s$ and $t < n/3$, we have

$$\frac{4}{9}n^2 < (n-t)^2 \leq s(n-t) \leq nt < \frac{1}{3}n^2,$$

which is a contradiction.

Therefore, we can conclude that some column k in (P_i's view of) the table contains at least $t+1$ one-entries. Now consider another honest party P_h who reaches the stage $|S_h| \geq n-t$ and thereby records Z_h. Since $(n-t)+(t+1) > n$, the kth column in P_h's view of the table must necessarily contain a one-entry. From the specification of the protocol, we can deduce that $P_k \in R_h$, and hence $G'_k \subseteq Z_h$ (where G'_k was received from P_k's A-Cast). In other words, there is a set $A := G'_k$ of size $n-t$ such that every honest party P_h will eventually have $A \subseteq Z_h$.

[16] Technically the table has n rows, each corresponding to an A-Casted R-set, and P_i has a local view of s rows that it has validated against R_i.

Once the first honest party P_i reaches the condition $|S_i| \geq n - t$, the simulator $\mathcal{S}$, being aware of message delivery, can easily determine the set A. Basically, $\mathcal{S}$ can form the table T as described above, find a column k with at least $t + 1$ ones, and then set $A := G'_k$ (or rather the indices of the parties in G'_k). Utilizing the set A, and the values b and y received from $\mathcal{F}_{\mathsf{a\text{-}occ}}$, the simulator is able to sample a vector of tallies that follows the same distribution as in the real-world execution and results in the same output as $\mathcal{F}_{\mathsf{a\text{-}occ}}$ when $b = 1$. Let us now outline the process for this inverse sampling. We first define the following events, given the set A and a random vector $(w_1, \ldots, w_n)$ where each element $w_i \xleftarrow{\mathrm{R}} [m]$.

- E_1: There is at least one repeat in the vector $(w'_1, \ldots, w'_n)$, where $w'_i = w_i \bmod n^2$ for all $i \in [n]$, and all the repeats lie inside A.
- $E_{1,y}$ for $y \in V$: The vector $(w_1, \ldots, w_n)$ is in E_1, and the smallest index j of the repeated elements in $(w'_1, \ldots, w'_n)$, where $w'_i = w_i \bmod n^2$ for all $i \in [n]$, contains an element w_j congruent to y modulo $|V|$.
- E_2: There are no repeats at all in the vector $(w'_1, \ldots, w'_n)$, where $w'_i = w_i \bmod n^2$ for all $i \in [n]$.
- E_3: There is at least one repeat in the vector $(w'_1, \ldots, w'_n)$, where $w'_i = w_i \bmod n^2$ for all $i \in [n]$, which is not fully contained inside A.

The simulator $\mathcal{S}$ initially computes the precise probabilities of the events E_1, E_2, and E_3.[17] Given that $m = \mathrm{lcm}(n^2, |V|)$, sampling vectors uniformly from $[m]^n$ and taking each coordinate modulo n^2 is equivalent to uniform sampling from $[n^2]^n$. Therefore, we can compute the aforementioned probabilities by considering vectors exclusively from $[n^2]^n$. Let $l = |A|$. For $1 \leq k \leq l$, denote by N_k the number of ways to choose the elements inside A from any fixed subset of k residues modulo n^2 such that *exactly* k distinct residues are chosen (equivalently, the number of strings of length l over an alphabet of size k that use each character at least once). We have the following recursive expression for N_k.

$$N_k = k^l - \sum_{i=1}^{k-1} \binom{k}{i} N_i$$

Note that $N_1, \ldots, N_l$ are efficiently computable. Moreover, for any $1 \leq k \leq l$, the number of ways to choose the elements in $(w'_1, \ldots, w'_n)$ from $[n^2]$ such that A contains exactly k distinct elements and all repeats (if any) lie inside A is

$$\binom{n^2}{k} N_k \prod_{j=0}^{n-l-1} (n^2 - k - j).$$

Noting that $k = l$ exactly corresponds to the case that there are no repeats at all, we now compute $p_i := \Pr[E_i]$ for $i \in [3]$:

[17] These probabilities only need to be computable by the simulator to give a constructive description of $\mathcal{S}$. Note that an existential argument, which is sufficient to complete the emulation, would not require $\mathcal{S}$ to calculate those probabilities at all. Nonetheless, we opt for a constructive approach to better illustrate the concept.

$$p_1 = \frac{1}{(n^2)^n} \sum_{k=1}^{l-1} \left(\binom{n^2}{k} N_k \prod_{j=0}^{n-l-1} (n^2 - k - j) \right)$$

$$p_2 = \frac{1}{(n^2)^n} \prod_{j=0}^{n-1} (n^2 - j)$$

$$p_3 = 1 - p_1 - p_2$$

Now we present a sampling strategy utilized by $\mathcal{S}$ to ensure that the vector of tallies is sampled in a manner that maintains both the observed distribution in the real-world execution and the consistency of outputs with the ideal functionality. This strategy is effective under the condition that the probability of outputting a random common coin in the ideal functionality, denoted as p, does not exceed the probability defined above as p_1, which is the probability with which the protocol guarantees a random common coin. Looking ahead, $\mathcal{S}$ is required to perform a task with a probability denoted as q_1, which becomes negative if $p > p_1$, thereby rendering the entire process unsound. For any $p \leq p_1$, the simulator does the following:

- If $\mathcal{F}_{\mathsf{a\text{-}occ}}$ outputs $b = 1$, uniformly sample from $E_{1,y}$ where y is the value received from $\mathcal{F}_{\mathsf{a\text{-}occ}}$.
- If $b = 0$ is received from $\mathcal{F}_{\mathsf{a\text{-}occ}}$:
  - With probability $q_1 = \frac{p_1 - p}{1-p}$, uniformly sample from E_1.
  - With probability $q_2 = \frac{p_2}{1-p}$, uniformly sample from E_2.
  - With probability $q_3 = \frac{p_3}{1-p}$, uniformly sample from E_3.

Before delving into the details of the sampling procedures, it is important to understand why this inverse sampling approach effectively eliminates any distinguishability. The simulator samples from $E_{1,y} \subseteq E_1$ when $\mathcal{F}_{\mathsf{a\text{-}occ}}$ outputs $b = 1$, and from E_1 with probability q_1 when $\mathcal{F}_{\mathsf{a\text{-}occ}}$ outputs $b = 0$. Since $\mathcal{F}_{\mathsf{a\text{-}occ}}$ samples $b \leftarrow \text{Bernoulli}(p)$, the simulator samples a vector from E_1 with probability $p + (1-p)q_1 = p_1$, which exactly matches the probability in the real-world execution. Moreover, when $b = 1$, the simulator ensures that the element in the sample at the smallest index of a repeated residue is congruent modulo $|V|$ to the value y received from $\mathcal{F}_{\mathsf{a\text{-}occ}}$, which guarantees that the simulated view is consistent with the output of $\mathcal{F}_{\mathsf{a\text{-}occ}}$. It is important to note that enforcing this constraint does not skew the probability since $\mathcal{F}_{\mathsf{a\text{-}occ}}$ samples y uniformly at random from V and m is a multiple of $|V|$. In cases when $b = 0$, the simulator influences $\mathcal{F}_{\mathsf{a\text{-}occ}}$ to output values that match those in the simulated execution. Additionally, $\mathcal{S}$ samples from E_2 and E_3 with probabilities q_2 and q_3, respectively. Since $\mathcal{F}_{\mathsf{a\text{-}occ}}$ outputs $b = 0$ with probability $1 - p$, the simulator samples from E_2 and E_3 with respective probabilities $(1-p)q_2 = p_2$ and $(1-p)q_3 = p_3$, which exactly match the corresponding real-world probabilities. Therefore, this inverse sampling strategy ensures that there are no distinguishability opportunities for $\mathcal{Z}$.

Now we explain how each sampling procedure works. The general approach is to first determine the "repeat pattern," which is encoded as a vector $(r_1, \ldots, r_n) \in [n^2]^n$ of residues modulo n^2, consistently with the event being sampled from, and then fill in each coordinate of the vector $(w_1, \ldots, w_n) \in [m]^n$ by sampling from the corresponding congruence class (with one caveat, discussed below). This is of course equivalent to sampling the elements in $(w_1, \ldots, w_n)$ such that their residues modulo n^2 have the appropriate repeat pattern, but it greatly simplifies the presentation and also enables us to cast the procedure for sampling from $E_{1,y}$ as a variation on the procedure for sampling from E_1.

- Procedure 1 (sampling from E_1 or $E_{1,y}$):

1. Initialize $(w_1, \ldots, w_n)$ and $(r_1, \ldots, r_n)$ to $(\perp, \ldots, \perp)$.
2. Randomly choose two distinct indices g and h from A.[18] Sample $r_g = r_h \xleftarrow{\mathrm{R}} [n^2]$.
3. For each $i \in A \setminus \{g, h\}$, sample $r_i \xleftarrow{\mathrm{R}} [n^2]$.
4. Fill in the remaining coordinates of $(r_1, \ldots, r_n)$ with random *distinct* residues that have not yet been chosen. That is, for each $j \in [n] \setminus A$, sample $r_j \xleftarrow{\mathrm{R}} [n^2] \setminus \cup_{k=1}^{n} \{r_k\}$.

4.5. (Only if sampling from $E_{1,y}$) Let $S \coloneqq \{j \in [n] \mid r_j = r_k \text{ for some } k \neq j\}$, and sample $x \xleftarrow{\mathrm{R}} \{v \in [m] \mid v \equiv y \mod |V|\}$. Set $w_{\min\{S\}} \coloneqq x$. Then, randomly choose a permutation π over $[n^2]$ such that $\pi(r_{\min\{S\}}) = x \bmod n^2$, and update $r_i \coloneqq \pi(r_i)$ for each $i \in [n]$.

5. For each $i \in [n]$ such that $w_i = \perp$, sample $w_i \xleftarrow{\mathrm{R}} \{v \in [m] \mid v \equiv r_i \mod n^2\}$.

- Procedure 2 (sampling from E_2):
  1. Initialize $(w_1, \ldots, w_n)$ to $(\perp, \ldots, \perp)$.
  2. Randomly choose n distinct residues $r_1, \ldots, r_n$ from $[n^2]$.
  3. For each $i \in [n]$, sample $w_i \xleftarrow{\mathrm{R}} \{v \in [m] \mid v \equiv r_i \mod n^2\}$.
- Procedure 3 (sampling from E_3):
  1. Initialize $(w_1, \ldots, w_n)$ and $(r_1, \ldots, r_n)$ to $(\perp, \ldots, \perp)$.
  2. Randomly choose an index g from $[n] \setminus A$ and an index h from $[n] \setminus \{g\}$. Sample $r_g = r_h \xleftarrow{\mathrm{R}} [n^2]$.
  3. For each $i \in [n] \setminus \{g, h\}$, sample $r_i \xleftarrow{\mathrm{R}} [n^2]$.
  4. For each $i \in [n]$, sample $w_i \xleftarrow{\mathrm{R}} \{v \in [m] \mid v \equiv r_i \mod n^2\}$.

The simulator uses Procedure 1 to sample from either E_1 or $E_{1,y}$. In the former case, there is no need to ensure that the sample is consistent with any particular output value. Thus, $\mathcal{S}$ can randomly "plant" a repeat inside A, sample the rest of the residues appearing within A totally at random, and lastly round out the repeat pattern by selecting unused residues for the indices outside of A. This process guarantees that we sample from E_1 with the uniform distribution.

The latter case, in which $\mathcal{S}$ must sample from E_1 while ensuring that the element at the smallest index of a repeated residue is congruent to y modulo $|V|$, requires a more complex approach. It is not enough to simply follow Procedure 1 and then readjust the repeated residue with minimum index (as well as the elements in the vector $(w_1, \ldots, w_n)$ from the corresponding congruence class) at the end. The reason is that adjusting values after sampling can potentially skew the distribution of the repeat pattern. If we adjust only the values of existing repeats, there is a chance that the adjusted value may match another repeated value, altering the distribution by decreasing the number of distinct repeated values and increasing the number of times a value is repeated. It is even possible that the adjusted value will collide with a value outside of the designated set A! On the other hand, introducing new repeats would increase the number of distinct repeated values and once again deviate from the desired distribution. Addressing these challenges requires careful consideration and decision-making on the part of $\mathcal{S}$, as determining when to add new repeats and when to readjust existing ones is a delicate task.

[18] Note that this is well-defined since $|A| = n - t > 2$ when $1 \leq t < n/3$.

Fortunately, it is possible for $\mathcal{S}$ to carefully re-sample the repeat pattern, preserving the overall *structure* but ensuring consistency with y. Since taking the repeated value with minimum index is a global criterion, this two-pass strategy is essentially the best we can hope for. In more detail, $\mathcal{S}$ starts by determining the (preliminary) repeat pattern as if it were sampling from E_1; in this way, $\mathcal{S}$ obtains a correct distribution of repeats naturally induced by random values. However, $\mathcal{S}$ now samples a random element $x \in [m]$ that is congruent to y modulo $|V|$, which is used to reassign the congruence class (modulo n^2) associated with the smallest index of a repeated value in the old repeat pattern (note that this new congruence class is not uniquely determined by y). The set of distinct residues appearing in the repeat pattern is then re-sampled around this constraint; in other words, we apply a random permutation over residues with the property that the appropriate residue is mapped to $x \bmod n^2$. The simulator now uses the modified repeat pattern to sample the elements in $(w_1, \ldots, w_n)$, except that the element at the smallest index of a repeated residue should be fixed to x. By following this procedure, $\mathcal{S}$ is guaranteed to sample uniformly from $E_{1,y}$, for any $y \in V$.

Procedure 2 is straightforward: $\mathcal{S}$ forms the repeat pattern by choosing random residues, ensuring that no value is used more than once. This simple strategy guarantees that the sample is drawn from E_2 and maintains a uniform distribution.

Finally, in Procedure 3, the simulator first randomly plants a repeat involving an index outside of A (it does not matter whether the second index is inside A or not). Then, the remaining residues can be chosen randomly without any restrictions. This process guarantees a uniform sample from E_3.

That completes the inverse sampling process. With the knowledge of A, b, and y, the simulator is able to sample a vector $(w_1, \ldots, w_n)$ that is indistinguishable from the observations in the real-world execution. Furthermore, the sampled vector is consistent with the output y of the ideal functionality when the fairness bit $b = 1$.

After sampling the vector of tallies $(w_1, \ldots, w_n)$, the simulator retains the flexibility to determine the secrets shared by honest parties. This is possible because none of the honest parties engage in the reconstruction phase of any $\mathcal{F}_{\mathsf{a\text{-}vss}}$ instance until the first honest party P_i satisfies the condition $|S_i| \geq n - t$. Consequently, $\mathcal{S}$ has sufficient time to make decisions regarding the shared secrets without the need to leak them to the adversary. At this point, $\mathcal{S}$ can initiate the opening of secrets through the reconstruction phase of $\mathcal{F}_{\mathsf{a\text{-}vss}}$ in such a way that if an honest party recovers the jth tally v_j, it will correspond to $v_j = w_j$. This can be accomplished by leveraging the fact that each tally consists of secrets from $t + 1$ parties, which must include at least some honest parties. By appropriately adjusting the secrets of the honest parties, $\mathcal{S}$ can ensure that the tally sums up to the desired value. It is crucial to note that, according to the inverse sampling process described above, the vector $(w_1, \ldots, w_n)$ maintains the same distribution as in the real-world execution. Therefore, sampling random secrets while conditioning them on the desired tallies does not introduce any distinguishability in the views.

Correctness. The above approach allows for generating an output in the simulated execution matched with the output of the ideal functionality and results in a view indistinguishable from the real-world execution if the inverse sampling process works correctly. This is the case because when $\mathcal{F}_{\mathsf{a\text{-}occ}}$ samples $b = 1$ and y, the simulator samples the vector of tallies from $E_{1,y}$ which guarantees the same output y for all the honest parties. On the other hand, when $b = 0$ is sampled by $\mathcal{F}_{\mathsf{a\text{-}occ}}$, $\mathcal{S}$ is allowed to influence the output of $\mathcal{F}_{\mathsf{a\text{-}occ}}$. The simulator, who observes the simulated execution, can seize this opportunity and exert influence to ensure that the output of $\mathcal{F}_{\mathsf{a\text{-}occ}}$ matches the output of the simulated execution. As mentioned above, the inverse sampling process relies on the

fact that $p \leq p_1$. By a direct utilization of Lemma 4, we can show that there is a constant lower bound for p_1 for any set A with size $|A| \geq \frac{2}{3}n$. Therefore, there exists some constant p such that $p \leq p_1$, which makes the simulation go through correctly.

Termination. To ensure the termination of the protocol and maintain the indistinguishability of the real and ideal worlds, we need to demonstrate that the protocol will eventually terminate.

For each honest party P_j, the secret-sharing of all of its votes $x_{j,1}, \ldots, x_{j,n}$ will eventually succeed for any honest party P_i. It is certainly the case then that every honest party P_i will eventually pass the threshold $|C_i| \geq t+1$ and A-Cast $C_i^{(t+1)}$.

Whenever an honest party P_h adds P_j to C_j, any other honest party P_i will eventually add P_j to C_i. This is because $\mathcal{F}_{\mathsf{a\text{-}vss}}$ guarantees that if the sharing of a secret succeeds for one honest party, then it will eventually succeed for all honest parties. Since every honest party P_h eventually A-Casts $C_h^{(t+1)}$, every honest party P_i will eventually receive at least $n-t$ A-Casts C'_j satisfying $|C'_j| = t+1$ and $C'_j \subseteq C_i$. Thus, every honest party P_i will eventually pass the threshold $|G_i| \geq n-t$ and A-Cast $G_i^{(n-t)}$.

Whenever an honest party P_h adds P_j to G_h, any other honest party P_i will eventually add P_j to G_i. This is because P_h received some C'_j from P_j's A-Cast, where $|C'_j| = t+1$ and $C'_j \subseteq C_h$, so eventually P_i will receive the same C'_j, and following the analysis above it will eventually hold that $C'_j \subseteq C_i$ as well. Since every honest party P_h eventually A-Casts $G_h^{(n-t)}$, every honest party P_i will eventually receive at least $n-t$ A-Casts G'_j satisfying $|G'_j| = n-t$ and $G'_j \subseteq G_i$. Thus, every honest party P_i will eventually pass the threshold $|R_i| \geq n-t$ and A-Cast $R_i^{(n-t)}$.

Whenever an honest party P_h adds P_j to R_h, any other honest party P_i will eventually add P_j to R_i. This is because P_h received some G'_j from P_j's A-Cast, where $|G'_j| = n-t$ and $G'_j \subseteq G_h$, so eventually P_i will receive the same G'_j, and following the analysis above it will eventually hold that $G'_j \subseteq G_i$ as well. Since every honest party P_h eventually A-Casts $R_h^{(n-t)}$, every honest party P_i will eventually receive at least $n-t$ A-Casts R'_j satisfying $|R'_j| = n-t$ and $R'_j \subseteq R_i$. Thus, every honest party P_i will eventually pass the threshold $|S_i| \geq n-t$ and record Z_i.

For each honest party P_i, every $P_j \in Z_i$ will eventually be added to G_h by any other honest party P_h. Thus, all honest parties will eventually start the reconstruction of all the secrets attached to every $P_j \in Z_i$, and P_i will eventually be able to compute each tally v_j that it needs to determine its output z_i. In actuality, this reasoning does not take into account the termination of parties; this is resolved via the termination procedure at the end of the protocol. An honest party P_i who is able to compute z_i on its own does not terminate immediately, but rather A-Casts z_i and continues to participate in the protocol (in particular, in the reconstruction of secrets). Eventually P_i will receive at least $n-t$ A-Casts z'_j, at which point it can safely terminate with the knowledge than any other honest party P_h will eventually receive those same A-Casts (even if P_h is unable to compute z_h). By taking the mode of these $n-t$ values, the majority of which must have been suggested by honest parties, it is guaranteed that all honest parties will output the same value in the case of a successful coin toss.

In summary, the protocol $\Pi_{\mathsf{a\text{-}occ}}$ ensures termination by guaranteeing that each honest party eventually satisfies the required thresholds and proceeds to the necessary phases of $\mathcal{F}_{\mathsf{a\text{-}vss}}$. This guarantees eventual output generation without any deadlocks, preserving the perfect indistinguishability of the real and ideal worlds.

We now claim that the overall round complexity of $\Pi_{\mathsf{a\text{-}occ}}$ is constant. The protocol consists of a constant number of steps, each involving constant-round operations. Note that by ensuring that the underlying building blocks $\mathcal{F}_{\mathsf{a\text{-}cast}}$ and $\mathcal{F}_{\mathsf{a\text{-}vss}}$ have constant-round realizations, we are able to

preserve the overall constant round complexity of protocol $\Pi_{\mathsf{a\text{-}occ}}$ even after replacing $\mathcal{F}_{\mathsf{a\text{-}cast}}$ and $\mathcal{F}_{\mathsf{a\text{-}vss}}$ with actual protocols realizing them.

We can conclude that the executions of $\Pi^{V}_{\mathsf{a\text{-}occ}}$ with $\mathcal{A}$ and $\mathcal{F}^{V,p}_{\mathsf{a\text{-}occ}}$ with $\mathcal{S}$ are perfectly indistinguishable to any environment $\mathcal{Z}$. This ensures that $\Pi^{V}_{\mathsf{a\text{-}occ}}$ UC-realizes $\mathcal{F}^{V,p}_{\mathsf{a\text{-}occ}}$ with perfect security. □

5 Concurrent A-BA in Expected-Constant Rounds

Concurrent A-BA plays a significant role in existing asynchronous MPC protocols, including those described in [BCG93, BKR94, HNP05, BH07, HNP08, Coh16, CGHZ16, BZL20, LLM+20]. Its primary purpose is to address the lack of coordination among the parties, which is amplified in the asynchronous setting. Specifically, it enables parties to reach agreement on a common subset of input providers (the "core set" [BCG93]). This capability is essential in various applications within asynchronous networks, where the different delays experienced by each party can lead to discrepancies in their respective views.

In this section, we dive into the important problem of achieving concurrent A-BA in an expected-constant number of rounds. As discussed in the Introduction, Ben-Or and El-Yaniv [BE03] highlighted the potential issue of running multiple executions of a probabilistic-termination protocol in parallel, which could lead to an increase in the expected number of rounds required for *all* executions to terminate. The concurrent A-BA protocol proposed in [BE03] relies on A-OLE and multi-valued A-BA, which can be instantiated using our A-OCC protocol from Section 4 and the extended A-BA protocol from [CR93, MR17], respectively. However, during our analysis, we discovered certain issues in their analysis that cast doubt on the expected-constant round complexity of one of their main building blocks and, consequently, their concurrent A-BA protocol. For a more comprehensive presentation of these issues, see Appendix A. It is unclear how to address these issues without modifying the protocol itself.

To rectify these concerns, we modify the underlying message distribution mechanism and incorporate an additional layer of message validation. These changes not only resolve the identified issues, but also significantly simplify the protocol. It is worth emphasizing that our revised concurrent A-BA protocol achieves a level of simplicity that is comparable to the synchronous version proposed in [BE03]. This accomplishment is significant because when designing an asynchronous counterpart to a synchronous protocol, achieving a level of simplicity on par with the synchronous version is often considered the ideal outcome. In the following, we present an ideal functionality for concurrent A-BA, describe our protocol and its required building blocks, and provide a security proof.

5.1 Concurrent A-BA Ideal Functionality

Concurrent A-BA, as the name suggests, refers to a primitive that enables parties to solve N instances of A-BA concurrently. We are primarily interested in the case $N = n$, corresponding to the emulation of n ideal A-BA primitives, commonly used in asynchronous MPC protocols to form the core set and overcome low message dispersion. However, in our study, we consider a more general version that allows for a broader range of values for N. In this setting, each party P_i initiates the concurrent A-BA by providing N values, namely $v_{i,1}, \ldots, v_{i,N}$. Subsequently, all parties receive the same set of N output values, denoted as $y_1, \ldots, y_N$. Each individual output value y_j is computed based on the input values $v_{1,j}, \ldots, v_{n,j}$, following the prescribed procedure outlined in the standard A-BA primitive. Specifically, if $n - 2t$ input values are identical, that common value (which we require to be unique) is selected as the output; otherwise, the output is determined

by the adversary. We capture the task of concurrent A-BA using the ideal functionality $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}$, shown in Figure 7.[19] Since we are able to achieve non-intrusion validity (i.e., for each instance, the corresponding output must be either $\perp$ or the corresponding input of an honest party), we present an *intrusion-tolerant* concurrent A-BA functionality.

Functionality $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$

The functionality is parameterized by a set V of values and a positive integer N representing the number of concurrent instances, and proceeds as follows. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$ and each $j \in [N]$, initialize $v_{i,j}$ to a default value $\perp$, $\mathsf{participated}_{i,j} := 0$, and delay values $D_i^{\mathrm{input}_j} = D_i^{\mathrm{output}} := 1$. Also initialize $y_1, \ldots, y_N$ to $\perp'$ and $a_1, \ldots, a_N$ to $\perp$, and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\texttt{delay}, \mathsf{sid}, P_i, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{type} \in \{\mathrm{input}_1, \ldots, \mathrm{input}_N, \mathrm{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{type}} := \max(1, D_i^{\mathsf{type}} + D)$ and send $(\texttt{delay-set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\texttt{input}, \mathsf{sid}, (v_1, \ldots, v_N))$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission Procedure and send $(\texttt{leakage}, \mathsf{sid}, P_i, (v_1, \ldots, v_N))$ to the adversary.
- Upon receiving $(\texttt{replace}, \mathsf{sid}, (v_1, \ldots, v_N))$ from the adversary, record $(a_1, \ldots, a_N) := (v_1, \ldots, v_N)$.
- Upon receiving $(\texttt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\texttt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $(\texttt{input}, \mathsf{sid}, (v_1, \ldots, v_N))$ was previously received from P_i, then for each $j \in [N]$ such that $\mathsf{participated}_{i,j} = 0$ do:

1. Update $D_i^{\mathrm{input}_j} := D_i^{\mathrm{input}_j} - 1$.
2. If $D_i^{\mathrm{input}_j} = 0$, then set $\mathsf{participated}_{i,j} := 1$ and record $v_{i,j} := v_j$.

Output Release Procedure: If $\sum_{k=1}^{n} \mathsf{participated}_{k,j} \geq n - t$ for every $j \in [N]$ then do:

1. Update $D_i^{\mathrm{output}} := D_i^{\mathrm{output}} - 1$.
2. If $D_i^{\mathrm{output}} = 0$, then do the following. For each $j \in [N]$ do: if $y_j = \perp'$, then set $y_j := x$ if there exists a unique value $x \in V$ such that $x = v_{k,j}$ for at least $n - 2t$ inputs $v_{k,j}$; else if $a_j = \perp$ or $a_j \in V$ was provided by at least one honest party, then set $y_j := a_j$; else set $y_j := \perp$. Additionally, set $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_N))$ to be sent to P_i.

Figure 7: The (intrusion-tolerant) concurrent A-BA functionality.

5.2 The Simplified Concurrent A-BA Protocol

Building blocks. Our concurrent A-BA protocol relies on A-Cast as the fundamental communication primitive due to its enhanced guarantees compared to basic message distribution mechanisms. The ideal functionality $\mathcal{F}_{\mathsf{a\text{-}cast}}$, which models the A-Cast primitive, was described in Section 3.2.

As another crucial building block, our protocol incorporates asynchronous oblivious leader election (A-OLE) as a coordination mechanism among the parties. A-OLE enables parties to randomly elect a leader from among themselves. The term "oblivious" indicates that parties are unaware of whether or not agreement on a random leader has been achieved. In our concurrent A-BA protocol, similar to the approach described in [BE03], there comes a point where all parties suggest outputs, and A-OLE assists them in reaching agreement on the output by adopting the suggestion of the elected leader. To capture the task of A-OLE, we parameterize the A-OCC functionality $\mathcal{F}_{\mathsf{a\text{-}occ}}$,

[19] In the preliminary version of this work [CFG+23], the functionality tracked participation jointly across all N instances rather than separately for each instance. Refining this formulation based on our current model resolves this discrepancy, which corresponds to the distinction between *strong* and *weak* parallel composition [CFG+25].

given in Section 4.1, by a domain with size equal to the number of parties. This yields an ideal functionality for A-OLE, denoted as $\mathcal{F}_{\mathsf{a\text{-}ole}}$, which is defined as $\mathcal{F}_{\mathsf{a\text{-}occ}}^{[n],p}$ for appropriate $p \in \Theta(1)$. Recall from Theorem 1 that $\mathcal{F}_{\mathsf{a\text{-}ole}}$ can be realized using protocol $\Pi_{\mathsf{a\text{-}occ}}^{[n]}$.

Our concurrent A-BA protocol leverages both binary and multi-valued A-BA. Binary A-BA is employed to achieve agreement on a critical decision within the protocol, namely whether to continue a particular iteration. On the other hand, the use of multi-valued A-BA addresses the inherent obliviousness of the A-OLE primitive by providing a means for parties to reach an agreement on the output, and it doubly serves as an agreement on termination. The ideal functionality for A-BA was described in Section 3.2.

As in [BE03], another essential component in our concurrent A-BA protocol is *truncated* executions of an A-BA protocol limited to a predefined number of iterations, consequently implying a fixed number of rounds. In the spirit of [CCGZ19], we model those executions with the ideal functionality $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$, defined in Figure 8. $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ is parameterized with V, p, and itr, where V denotes the domain, p represents the termination probability in each iteration, and itr indicates the maximum number of iterations in the execution.

This ideal functionality also encapsulates a "1-shift" property for termination, meaning that all honest parties produce the output within two consecutive iterations. We note that the adversary has the discretion to determine which parties discover the output first. Unlike a traditional A-BA functionality that outputs a single value, $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ produces a vector of values that includes the output after each iteration of the execution. Modeling the 1-shift property and providing outputs for all iterations is crucial since our concurrent A-BA protocol relies on those properties of truncated executions of A-BA.

It is worth mentioning that $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}^{V,p,\mathsf{itr}}$ can be implemented by executing any intrusion-tolerant A-BA protocol with the 1-shift property and a termination probability of p in each iteration, precisely for itr iterations, and concatenating the output of all iterations to get the final output (with λ representing iterations without output). Canetti and Rabin's binary A-BA protocol [CR93] possesses the desired properties of intrusion tolerance and terminating with a constant probability in each iteration. However, using Bracha's technique for termination [Bra87] in Canetti and Rabin's binary A-BA protocol does not admit the 1-shift property. The main reason is that parties may take the shortcut and use Bracha termination messages to generate the output in their very early iterations. Fortunately, Bracha's termination procedure is unnecessary in the truncated execution of Canetti and Rabin's binary A-BA protocol. This is primarily due to the fact that all parties will naturally terminate after a fixed number of iterations. With this adjustment, Canetti and Rabin's binary A-BA protocol also successfully attains the 1-shift property. Moreover, Mostéfaoui and Raynal's multi-valued A-BA protocol [MR17] offers intrusion tolerance, the 1-shift property, and terminating with a constant probability in each iteration if the underlying binary A-BA used in their construction also exhibits these characteristics. Thus, we can formulate the following proposition about realizing $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}^{V,p,\mathsf{itr}}$.

Proposition 5. *For some constant probability p, any domain V, and any integer itr, $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}^{V,p,\mathsf{itr}}$ can be UC-realized with statistical security in the $\mathcal{F}_{\mathsf{a\text{-}smt}}$-hybrid model, in constant rounds and in the presence of an adaptive and malicious t-adversary, provided $t < n/3$.*

The above statement guarantees statistical security since the A-BA protocols we consider rely on the A-VSS primitive, which can only be realized with statistical security when $t < n/3$. In fact, by working in the $\mathcal{F}_{\mathsf{a\text{-}vss}}$-hybrid model, we can achieve perfectly secure $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$. We remark that the intrusion tolerance of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ is not a requirement in our concurrent A-BA protocol; however, employing a non-intrusion-tolerant version of truncated A-BA will naturally lead to a non-intrusion-tolerant concurrent A-BA protocol.

Functionality $\mathcal{F}^{V,p,\mathsf{itr}}_{\text{trunc-a-ba}}$

The functionality is parameterized by a set V of values, a probability p, and an integer itr representing the number of iterations to run for, and proceeds as follows. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$, where $\mathcal{P}$ is a player set of size n. For each $P_i \in \mathcal{P}$, initialize value v_i and vector y_i to a default value $\perp$, $\mathsf{participated}_i := 0$, and delay values $D_i^{\text{input}} = D_i^{\text{output}} := 1$. Also initialize values y, a, and r to $\perp$, the set $\mathcal{E}$ of parties receiving output earlier to $\mathcal{P}$, and $t := \lceil \frac{n}{3} \rceil - 1$.

- Upon receiving $(\texttt{delay}, \mathsf{sid}, P_i, \mathsf{type}, D)$ from the adversary for $P_i \in \mathcal{P}$, $\mathsf{type} \in \{\text{input}, \text{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, update $D_i^{\mathsf{type}} := \max(1, D_i^{\mathsf{type}} + D)$ and send $(\texttt{delay-set}, \mathsf{sid})$ to the adversary.
- Upon receiving $(\texttt{early}, \mathsf{sid}, \mathcal{R})$ from the adversary, update $\mathcal{E} := \mathcal{R}$.
- Upon receiving $(\texttt{input}, \mathsf{sid}, v)$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission Procedure and send $(\texttt{leakage}, \mathsf{sid}, P_i, v)$ to the adversary.
- Upon receiving $(\texttt{replace}, \mathsf{sid}, v)$ from the adversary, record $a := v$.
- Upon receiving $(\texttt{fetch}, \mathsf{sid})$ from $P_i \in \mathcal{P}$ (or the adversary on behalf of corrupted P_i), run the Input Submission and Output Release Procedures, and send $(\texttt{fetched}, \mathsf{sid}, P_i)$ to the adversary and any messages set by the Output Release Procedure to P_i.

Input Submission Procedure: If $\mathsf{participated}_i = 0$ and $(\texttt{input}, \mathsf{sid}, v)$ was previously received from P_i, then do:

1. Update $D_i^{\text{input}} := D_i^{\text{input}} - 1$.
2. If $D_i^{\text{input}} = 0$, then set $\mathsf{participated}_i := 1$ and record $v_i := v$.

Output Release Procedure: If $\sum_{j=1}^{n} \mathsf{participated}_j \geq n - t$ then do:

1. Update $D_i^{\text{output}} := D_i^{\text{output}} - 1$.
2. If $D_i^{\text{output}} = 0$, then do the following. If $r = \perp$, then sample $r \leftarrow \text{Geometric}(p)$[a] and set $y := x$ if there exists a unique value $x \in V$ such that $x = v_k$ for at least $n - 2t$ inputs v_k; else if $a = \perp$ or $a \in V$ was provided by at least one honest party, then set $y := a$; else set $y := \perp$. If $y_i = \perp$ then do:
   - If $P_i \in \mathcal{E}$, then for $1 \leq k < r$ let $y_{i,k} := \lambda$ (the empty string) and for $r \leq k \leq \mathsf{itr}$ let $y_{i,k} := y$.
   - If $P_i \notin \mathcal{E}$, then for $1 \leq k \leq r$ let $y_{i,k} := \lambda$ and for $r < k \leq \mathsf{itr}$ let $y_{i,k} := y$.
   - Set $y_i := (y_{i,1}, \ldots, y_{i,\mathsf{itr}})$.

   Additionally, set $(\texttt{output}, \mathsf{sid}, y_i)$ to be sent to P_i.

[a] Recall that the geometric distribution corresponds to the termination probability in expected-constant-round A-BA protocols. Moreover, by adjusting this distribution, the functionality can be generalized to accommodate other A-BA protocols.

Figure 8: The (intrusion-tolerant) truncated A-BA functionality with the 1-shift property.

The new concurrent A-BA protocol. Our protocol builds on the core ideas presented in [BE03]. In addition to instantiating the missing OLE building block using our OCC protocol from Section 4, we address the issue in the analysis by redesigning the message distribution phase. Our revised message distribution mechanism not only resolves the issue in the proof but also provides stronger guarantees, which in turn simplifies the final protocol design. In fact, apart from the message distribution phase, the overall structure of our protocol closely resembles the synchronous version of the protocol described in [BE03], as illustrated in Figure 9.

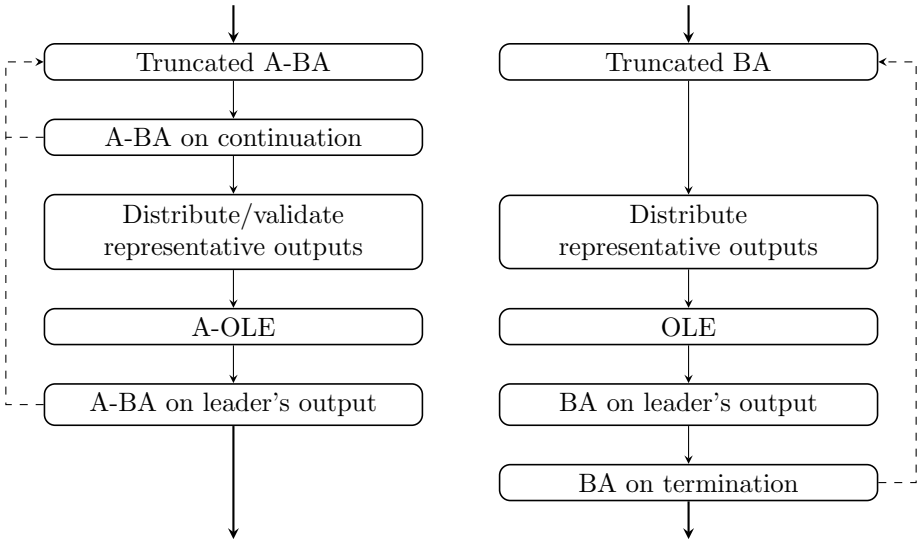

Figure 9: Side-by-side comparison of our simplified concurrent A-BA protocol (left) and Ben-Or and El-Yaniv's concurrent (synchronous) BA protocol (right).

Before diving into the high-level description of our protocol, we highlight the choices we made in the message distribution mechanism and discuss some alternative approaches that fail to meet our requirements. In the eventual-delivery model, at least $n - t$ parties will receive each other's messages if they wait for a sufficient duration, as messages from honest parties will eventually be delivered to one another, but determining the exact waiting time required is not straightforward. One possible approach is to instruct parties to A-Cast the identities of the parties from which they have received messages. After constructing a graph with parties as vertices and adding edges between parties that have reported message receipts from each other, we can look for a clique of size $n - t$; however, finding a maximum clique is known to be NP-complete and also difficult to approximate [FGL+91].

To overcome this challenge, one possible approach is to investigate alternative structures that offer weaker guarantees regarding message dispersion but can be efficiently identified [BCG93, BKR94, Can96, Pat11]. However, it is important to note that this approach does not guarantee the precise message dispersion required for our specific application. In our analysis, it is crucial that *all* honest parties receive messages from a linear fraction of other parties. Even finding a clique of size $n - t$ does not guarantee this level of message dispersion, rendering this approach unsuitable for our protocol. Thus, we adopt another approach that has been used in prior works [Fel89, CR93], and extend it by incorporating a precondition check before the process and introducing a validation layer during the execution. These additions enhance the guarantees, making them more suitable and effective for our specific purpose. We proceed to explain our protocol.

Similar to the (synchronous) protocol described in [BE03], in our concurrent A-BA protocol,

each party initiates for every A-BA instance a batch of m executions of the A-BA protocol (over the same inputs) for a fixed number of iterations, denoted as itr. If the A-BA protocol has a termination probability of at least a constant value p in each iteration, which is the case for most existing A-BA protocols, suitable values of m and itr can be determined so that each party obtains at least one output value for each batch. Each party then selects an output for each instance and forms a suggestion for the final output. The next paragraph, which explains the mechanism for distributing suggestions among parties, is our main modification to the protocol. Then, for the remaining part, we can use a similar structure as the synchronous version of the protocol.

Firstly, parties initiate a binary A-BA protocol to determine a specific condition that allows for choosing and validating suggested outputs later in the execution. Based on the outcome of this binary A-BA, parties decide whether to continue or start over. In the case of continuation, parties perform A-Cast operations to distribute their suggestions and wait to receive suggestions from other parties. Each party only accepts an A-Cast message containing a suggested output if the value is consistent with the outputs obtained from its own truncated A-BA executions. This validation step is crucial as it ensures that even corrupted parties provide acceptable (correct) suggestions. The validation is based on the 1-shift property of A-BA that ensures all honest parties terminate within two consecutive iterations. Parties wait to accept A-Casts of suggested outputs from at least $n - t$ parties and then A-Cast the set of all these $n - t$ suggestions along with the identities of the corresponding senders.[20] They continue accepting A-Casts of suggestions and sets until they receive at least $n - t$ sets that are fully contained within their accepted suggestions. At this point, a sufficient number of messages have been exchanged, and using a counting argument similar to the one in [Fel89, CR93], it can be deduced that at least $n - 2t > n/3$ parties have their suggested outputs received by all honest parties.

After the message distribution phase described above, our protocol proceeds similarly to the synchronous protocol presented in [BE03]. Specifically, parties execute OLE to elect a random leader to adopt its suggested output. Subsequently, a multi-valued A-BA is performed on the adopted output to address the oblivious nature of OLE. Depending on whether agreement on a non-default output is reached, this results in the termination or restarting of the protocol. The intrusion-tolerance property of multi-valued A-BA is crucial to make sure that the common output is not provided by a corrupted party alone. It is worth noting that the favorable scenario occurs when the leader is among the $n - 2t$ parties whose suggested outputs have been accepted by all honest parties. If the leader is elected randomly, this event happens with a probability of $1/3$. In this case, all parties adopt the same output, resulting in termination.

It is worth noting that the set of $n - 2t$ parties whose suggested outputs have been accepted by all honest parties may only contain a single honest party. This single honest party can only be elected with a probability of $O(1/n)$ by the OLE. However, due to the validation step in the message-distribution phase, there is no longer a need to ensure that an honest leader is elected, as the suggested values from corrupted parties are considered valid outputs. Refer to Figure 10 for a formal description of protocol $\Pi_{\mathsf{conc\text{-}a\text{-}ba}}$.

5.3 Security Proof

We now consider the security of our concurrent A-BA protocol. Before stating the theorem, it is worth noting that the specific parameters of the hybrid model, which combine the different ideal functionalities, are not explicitly specified in the theorem statement. However, they can be

[20] In fact, it suffices to A-Cast the identities alone. We primarily use this observation to simplify the description of the protocol, though it also reduces the communication complexity.

Protocol $\Pi^{V,N,m,p,\mathsf{itr}}_{\mathsf{conc\text{-}a\text{-}ba}}$

The protocol is parameterized by a set V of possible inputs (for each instance), positive integers N and m representing respectively the number of instances and the number of batched executions to run per instance, a termination probability p, and the number itr of iterations after which to truncate the executions. At the first activation, verify that $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ for a player set $\mathcal{P}$ of size n. Let $t := \lceil \frac{n}{3} \rceil - 1$. Party $P_i \in \mathcal{P}$ proceeds as follows. Initialize sets $Q_{i,1}$, $Q_{i,2}$, and W_i to $\emptyset$, flag $\mathsf{finished}$ to 0, and counter φ to 1. Also initialize $S^r_{i,j} := \emptyset$, $b_{j,k} := 0$, $\vec{C}'_h := \bot$, and $y_{i,j} := \bot$ for all $h \in [n]$, $j \in [N]$, $k \in [m]$, and $\mathsf{itr} \leq r \leq \mathsf{itr} + 3$.

- Upon receiving input $(\mathtt{input}, \mathsf{sid}, (v_{i,1}, \ldots, v_{i,N}))$ from the environment where $v_{i,1}, \ldots, v_{i,N} \in V$, do the following. For each $j \in [N]$ and $k \in [m]$, send $(\mathtt{input}, \mathsf{sid}^{j,k}_\varphi, v_{i,j})$ to an instance of $\mathcal{F}^{V,p,\mathsf{itr}+3}_{\mathsf{trunc\text{-}a\text{-}ba}}$ with SID $\mathsf{sid}^{j,k}_\varphi := (\mathsf{sid}, \varphi, j, k)$.
- Upon receiving input $(\mathtt{fetch}, \mathsf{sid})$ from the environment, if $\mathsf{finished} = 1$ then output $(\mathtt{output}, \mathsf{sid}, (y_{i,1}, \ldots, y_{i,N}))$ to the environment; else do:
  1. For each $j \in [N]$ and $k \in [m]$, fetch the output from the instance of $\mathcal{F}^{V,p,\mathsf{itr}+3}_{\mathsf{trunc\text{-}a\text{-}ba}}$ with SID $\mathsf{sid}^{j,k}_\varphi$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{j,k}_\varphi, (z_{i,1}, \ldots, z_{i,\mathsf{itr}+3}))$, update $S^r_{i,j} := S^r_{i,j} \cup \{z_{i,r}\} \setminus \{\lambda\}$ for $\mathsf{itr} \leq r \leq \mathsf{itr}+3$, and also set $b_{j,k} := 1$.
  2. Wait until $b_{j,k} = 1$ for all $j \in [N]$ and $k \in [m]$. Then, let $\mathsf{continue}_i := 1$ if for all $j \in [N]$ it holds that $S^{\mathsf{itr}}_{i,j} \neq \emptyset$, and $\mathsf{continue}_i := 0$ otherwise; send $(\mathtt{input}, \mathsf{sid}^{\mathrm{cont}}_\varphi, \mathsf{continue}_i)$ to an instance of $\mathcal{F}^{\{0,1\}}_{\mathsf{a\text{-}ba}}$ with SID $\mathsf{sid}^{\mathrm{cont}}_\varphi := (\mathsf{sid}, \varphi, \mathtt{cont})$.
  3. Fetch the output from the instance of $\mathcal{F}^{\{0,1\}}_{\mathsf{a\text{-}ba}}$ with SID $\mathsf{sid}^{\mathrm{cont}}_\varphi$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{\mathrm{cont}}_\varphi, \mathsf{continue})$, do the following. If $\mathsf{continue} = 1$, wait until for all $j \in [N]$ it holds that $S^{\mathsf{itr}+1}_{i,j} \neq \emptyset$; then for each $j \in [N]$ choose $c_{i,j} \in S^{\mathsf{itr}+1}_{i,j}$, and send $(\mathtt{send}, \mathsf{sid}^{\mathrm{vector}}_{i,\varphi}, (c_{i,1}, \ldots, c_{i,N}))$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}^{\mathrm{vector}}_{i,\varphi} := (\mathcal{P}, P_i, \mathsf{sid}', \varphi, \mathtt{vector})$. Otherwise, start over (i.e., increment φ, re-initialize all other local variables, and run truncated A-BA mN times again).
  4. For each $h \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}^{\mathrm{vector}}_{h,\varphi}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{\mathrm{vector}}_{h,\varphi}, (c'_{h,1}, \ldots, c'_{h,N}))$, record $\vec{C}'_h := (c'_{h,1}, \ldots, c'_{h,N})$ and for each $b \in \{1,2\}$ wait until it holds that $c'_{h,j} \in S^{\mathsf{itr}+1+b}_{i,j}$ for all $j \in [N]$; then update $Q_{i,b} := Q_{i,b} \cup \{P_h\}$.
  5. Wait until $|Q_{i,1}| \geq n - t$. Then, send $(\mathtt{send}, \mathsf{sid}^{\mathrm{set}}_{i,\varphi}, Q^{(n-t)}_{i,1})$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}^{\mathrm{set}}_{i,\varphi} := (\mathcal{P}, P_i, \mathsf{sid}', \varphi, \mathtt{set})$.
  6. For each $h \in [n]$, fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}cast}}$ with SID $\mathsf{sid}^{\mathrm{set}}_{h,\varphi}$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{\mathrm{set}}_{h,\varphi}, Q'_{h,1})$ where $|Q'_{h,1}| = n - t$, wait until $Q'_{h,1} \subseteq Q_{i,2}$; then update $W_i := W_i \cup \{h\}$.
  7. Wait until $|W_i| \geq n - t$. Then, send $(\mathtt{input}, \mathsf{sid}^{\mathrm{elect}}_\varphi)$ to an instance of $\mathcal{F}_{\mathsf{a\text{-}ole}}$ with SID $\mathsf{sid}^{\mathrm{elect}}_\varphi := (\mathsf{sid}, \varphi, \mathtt{elect})$.
  8. Fetch the output from the instance of $\mathcal{F}_{\mathsf{a\text{-}ole}}$ with SID $\mathsf{sid}^{\mathrm{elect}}_\varphi$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{\mathrm{elect}}_\varphi, l_i)$, wait until $|W_i| \geq n-t$; then send $(\mathtt{input}, \mathsf{sid}^{\mathrm{vector}}_\varphi, \vec{C}'_{l_i})$ to an instance of $\mathcal{F}^{V^N \cup \{\bot\}}_{\mathsf{a\text{-}ba}}$ with SID $\mathsf{sid}^{\mathrm{vector}}_\varphi := (\mathsf{sid}, \varphi, \mathtt{vector})$ if it holds that $P_{l_i} \in Q_{i,2}$, and send $(\mathtt{input}, \mathsf{sid}^{\mathrm{vector}}_\varphi, \bot)$ otherwise.
  9. Fetch the output from the instance of $\mathcal{F}^{V^N \cup \{\bot\}}_{\mathsf{a\text{-}ba}}$ with SID $\mathsf{sid}^{\mathrm{vector}}_\varphi$. Upon receiving back $(\mathtt{output}, \mathsf{sid}^{\mathrm{vector}}_\varphi, \vec{Y})$, if $\vec{Y} \neq \bot$ then record $(y_{i,1}, \ldots, y_{i,N}) := \vec{Y}$ and set $\mathsf{finished} := 1$; else start over.

Figure 10: The concurrent A-BA protocol.

determined from the protocol's parameters and are integral to the overall security guarantees of the protocol. Now, let us state the theorem formally:

Theorem 2. *For any domain V, integer $N \geq 2$, constant $0 < p < 1$, and constant integer $\mathsf{itr} > 1$, setting $m := \left\lceil \log_{\frac{1}{1-p}} N \right\rceil$, the protocol $\Pi^{V,N,m,p,\mathsf{itr}}_{\mathsf{conc\text{-}a\text{-}ba}}$ UC-realizes $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$ with statistical security in the $(\mathcal{F}_{\mathsf{a\text{-}cast}}, \mathcal{F}_{\mathsf{a\text{-}ba}}, \mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}, \mathcal{F}_{\mathsf{a\text{-}ole}})$-hybrid model, in expected-constant rounds and in the presence of an adaptive and malicious t-adversary, provided $t < n/3$.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with $\Pi^{V,N,m,p,\mathsf{itr}}_{\mathsf{conc\text{-}a\text{-}ba}}$ and $\mathcal{A}$, or with $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$ and $\mathcal{S}$. Here, V and N are given parameters of the problem, and p is the parameter for the $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ hybrid, which is determined based on the specific protocol used to realize it. The parameters m and itr need to be appropriately adjusted based on the other parameters to ensure an expected-constant round complexity for $\Pi^{V,N,m,p,\mathsf{itr}}_{\mathsf{conc\text{-}a\text{-}ba}}$. As discussed below, setting $m := \left\lceil \log_{\frac{1}{1-p}} N \right\rceil$ for any constant integer $\mathsf{itr} > 1$ suffices.

The simulator internally runs a copy of $\mathcal{A}$, and plays the roles of all the hybrids and the parties in a simulated execution of the protocol. All inputs from $\mathcal{Z}$ are forwarded to $\mathcal{A}$, and all outputs from $\mathcal{A}$ are forwarded to $\mathcal{Z}$. Moreover, whenever $\mathcal{A}$ corrupts a party in the simulation, $\mathcal{S}$ corrupts the same party in the ideal world by interacting with $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$, and if the corruption was direct (i.e., not via either of the aiding functionalities), then $\mathcal{S}$ sends $\mathcal{A}$ the party's state and thereafter follows $\mathcal{A}$'s instructions for that party. Moreover, $\mathcal{S}$ manages delays in the ideal functionality based on the delays that $\mathcal{A}$ sets on the hybrids; in particular, it keeps the output delays high enough that $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$ releases its output only once the simulated execution has terminated, and sets the input delays for each instance as described in the correctness part below.

The simulated execution begins when $\mathcal{S}$ receives messages of the form $(\mathtt{leakage}, \mathsf{sid}, \cdot, \cdot)$ from $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$. These messages contain the values that party P_i receives as input from the environment. $\mathcal{S}$ can derive these input values from the leakage message $(\mathtt{leakage}, \mathsf{sid}, P_i, (v_1, \ldots, v_N))$ by setting $v_{i,j} = v_j$ for all $j \in [N]$. The set $\{v_{1,j}, v_{2,j}, \ldots, v_{n,j}\}$ represents the inputs for the jth instance of the (concurrent) A-BA that the parties are supposed to solve.

Upon receiving these leakage messages, $\mathcal{S}$ can initiate sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ with the actual inputs. For each instance $j \in [N]$, $\mathcal{S}$ invokes m sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ with the inputs $v_{1,j}, v_{2,j}, \ldots, v_{n,j}$. After this point, $\mathcal{S}$ continues the simulation by honestly playing the roles of other ideal functionalities and honest parties involved in the protocol. Since there are no private states that $\mathcal{S}$ needs to guess or fake in order to provide a consistent view, the rest of the simulation proceeds straightforwardly.

Now, this internal execution generates a transcript that is perfectly indistinguishable from a real-world execution. Therefore, it remains to show that the protocol execution indeed results in the same output as $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$. To demonstrate this, we first establish that with probability 1 the execution will eventually terminate. Additionally, we show that if at least $n - 2t$ parties have the same input value for any instance (and there is only one such value), then everyone will receive that value as the output; otherwise, the output will be either one of the inputs provided by some honest parties for that instance or a special value $\perp$.

Termination. After initializing some variables, the parties collectively initiate m sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ for each of the N instances. Since the maximum number of corrupted parties is t and $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ only requires the participation of $n - t$ parties, all sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ will eventually terminate upon receiving a sufficient number of activations from the environment (i.e., every honest party P_i will eventually set $b_{j,k} := 1$ for all $j \in [N]$ and $k \in [m]$).

If, for all N instances, the integer r sampled in any of the m sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ for that instance is at most $\mathsf{itr} - 1$ (so that an honest party P_i must eventually receive $z_{i,\mathsf{itr}} \neq \lambda$ from that session), then every honest party P_i eventually sets $\mathsf{continue}_i := 1$. Here, r is sampled from the geometric distribution Geometric(p) (that corresponds to expected-constant-round A-BA as in [Fel89, CR93]). We define E as the event where for all instances at least one of the m sessions of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ has $r \leq \mathsf{itr} - 1$. The probability of event E occurring can be calculated as follows, using similar logic as presented in [BE03].

We may assume $N \geq 2$, since for $N = 1$ the task is a single A-BA. Fix an instance $j \in [N]$. Since $r \leftarrow \text{Geometric}(p)$, a single session of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ has $\Pr[r \geq \mathsf{itr}] = (1-p)^{\mathsf{itr}-1}$, and since the m sessions of instance j sample independently, the event A_j that none of them samples $r \leq \mathsf{itr} - 1$ has

$$\Pr[A_j] = \left((1-p)^{\mathsf{itr}-1}\right)^m = (1-p)^{m(\mathsf{itr}-1)}.$$

The N instances are independent as well, and $E = \bigcap_{j=1}^{N} A_j^{\mathrm{c}}$, so

$$\Pr[E] = \prod_{j=1}^{N} (1 - \Pr[A_j]) = \left(1 - (1-p)^{m(\mathsf{itr}-1)}\right)^N.$$

Set $m \coloneqq \left\lceil \log_{\frac{1}{1-p}} N \right\rceil$, so that $(1-p)^m \leq \frac{1}{N}$. We then have

$$\Pr[E] \geq \left(1 - \frac{1}{N^{\mathsf{itr}-1}}\right)^N \geq \left(1 - \frac{1}{N}\right)^N \geq \frac{1}{4},$$

where the second inequality holds because $\mathsf{itr} \geq 2$, so $N^{-(\mathsf{itr}-1)} \leq N^{-1}$, and the third because $(1 - \frac{1}{N})^N$ is increasing in N, with value $\frac{1}{4}$ at $N = 2$.

So, for a constant value of $\mathsf{itr} > 1$, the probability $\Pr[E]$ is a non-zero constant. This implies that, with appropriate parameter choices, every honest party P_i sets $\mathsf{continue}_i \coloneqq 1$ with constant probability. In other words, the A-BA on continuation outputs $\mathsf{continue} = 1$ with constant probability.

In any case, since every honest party P_i eventually sets $\mathsf{continue}_i$ and $\mathcal{F}_{\mathsf{a\text{-}ba}}$ only requires participation from $n - t$ parties, agreement on $\mathsf{continue}$ will eventually be reached. When $\mathsf{continue} = 0$, all honest parties will eventually proceed to the next iteration.

On the other hand, when $\mathsf{continue} = 1$, at least one honest party P_g must have provided $\mathsf{continue}_g = 1$, meaning that it observed $S_{g,j}^{\mathsf{itr}} \neq \emptyset$ for all $j \in [N]$. Note that whenever an honest party P_g adds $z_{g,r} \neq \lambda$ to $S_{g,j}^{r}$ for some $j \in [N]$ and $\mathsf{itr} \leq r \leq \mathsf{itr} + 2$, any honest party P_i will eventually add $z_{i,r+1} = z_{g,r}$ to $S_{i,j}^{r+1}$, since $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ models the 1-shift property. Thus, in case of continuation, every honest party P_i will eventually have $S_{i,j}^{\mathsf{itr}+1} \neq \emptyset$ for all $j \in [N]$ and A-Cast $(c_{i,1}, \ldots, c_{i,N})$.

Since every honest party P_g eventually A-Casts a vector $(c_{g,1}, \ldots, c_{g,N})$ satisfying $c_{g,j} \in S_{g,j}^{\mathsf{itr}+1}$ for all $j \in [N]$, following the analysis above every honest party P_i will eventually receive at least $n - t$ A-Casts $\vec{C}'_h = (c'_{h,1}, \ldots, c'_{h,N})$ satisfying $c'_{h,j} \in S_{i,j}^{\mathsf{itr}+2}$ for all $j \in [N]$. Thus, every honest party P_i will eventually pass the threshold $|Q_{i,1}| \geq n - t$ and A-Cast $Q_{i,1}^{(n-t)}$.

Whenever an honest party P_g adds P_h to $Q_{g,1}$, any other honest party P_i will eventually add P_h to $Q_{i,2}$. This is because P_g received some $\vec{C}'_h = (c'_{h,1}, \ldots, c'_{h,N})$ from P_h's A-Cast, where $c'_{h,j} \in S_{g,j}^{\mathsf{itr}+2}$ for all $j \in [N]$, so eventually P_i will receive the same $\vec{C}'_h$, and following the analysis above it will eventually hold that $c'_{h,j} \in S_{i,j}^{\mathsf{itr}+3}$ for all $j \in [N]$. Since every honest party P_g eventually A-Casts $Q_{g,1}^{(n-t)}$, every honest party P_i will eventually receive at least $n - t$ A-Casts $Q'_{h,1}$ satisfying $|Q'_{h,1}| = n - t$ and $Q'_{h,1} \subseteq Q_{i,2}$. Thus, every honest party P_i will eventually pass the threshold $|W_i| \geq n - t$ and participate in $\mathcal{F}_{\mathsf{a\text{-}ole}}$.

Before proceeding, it is important to note that by using a similar counting argument as in [Fel89, CR93], we can guarantee that messages (suggestions) from at least $n - 2t$ parties will eventually be received by all honest parties (by the time of participation in $\mathcal{F}_{\mathsf{a\text{-}ole}}$). In more detail, consider the first honest party P_i who reaches the stage where $s \coloneqq |W_i| \geq n - t$. We can construct an $s \times n$

table T, where $T_{h,l} = 1$ if and only if $P_h \in W_i$ and $P_l \in Q'_{h,1}$ (where $Q'_{h,1}$ was received from P_h's A-Cast). Each row of the table contains (exactly) $n - t$ entries equal to one, indicating that there are a total of $s(n-t)$ one-entries in T. Let q be the minimum number of columns in T that contain at least $t+1$ one-entries. By considering the worst-case distribution of ones, we have the inequality

$$qs + (n - q)t \geq s(n - t).$$

Noting that it suffices to consider the case $s = n - t$ and rearranging terms, we have

$$\begin{aligned} q &\geq \frac{(n-t)^2 - nt}{n - 2t} \\ &= \frac{(n-t)(n-2t) - t^2}{n - 2t} \\ &= n - t - \frac{t^2}{n - 2t} \\ &\geq n - 2t. && (n - 2t > t) \end{aligned}$$

Therefore, we can conclude that at least $q \geq n - 2t$ columns in (P_i's local view of) the table contain at least $t + 1$ one-entries. Fix such a column l, and consider another honest party P_g who reaches the stage $|W_g| \geq n - t$. Since $(n - t) + (t + 1) > n$, the lth column in P_g's view of the table must necessarily contain a one-entry. This means that P_g received the A-Cast of $Q'_{h,1}$ from some party P_h, where $P_l \in Q'_{h,1} \subseteq Q_{g,2}$. In other words, a set of at least $n - 2t$ parties will eventually be contained in the $Q_{g,2}$ set of every honest party P_g (by the time that $|W_g| \geq n - t$).

Since all honest parties eventually participate in $\mathcal{F}_{\mathsf{a\text{-}ole}}$, which only requires participation from $n - t$ parties, every honest party P_i will eventually receive the index of a party P_{l_i}. With constant probability, there is agreement on a random leader P_l. Conditioned on agreeing on a random leader, with probability at least $\frac{n-2t}{n} > \frac{1}{3}$ the leader P_l is among the parties whose vectors will eventually be delivered to (and validated by) all honest parties. We stress that it is crucial here that a linear number of such parties are fixed *before* any honest party participates in $\mathcal{F}_{\mathsf{a\text{-}ole}}$, and consequently before the identity of the leader is leaked to the adversary. Since every honest party P_i waits until $|W_i| \geq n - t$, the condition $P_l \in Q_{i,2}$ will hold, and hence all honest parties will input (the same) $\vec{C}'_l \neq \perp$ to the multi-valued A-BA on the leader's vector. In this case, every honest party P_i will eventually receive $\vec{Y} = \vec{C}'_l$ from the A-BA and terminate.

In any case, every honest party P_i eventually has $|W_i| \geq n - t$ and participates in the multi-valued A-BA, so agreement on some $\vec{Y}$ will eventually be reached. If $\vec{Y} = \perp$, all honest parties will eventually proceed to the next iteration; otherwise, they will all output $\vec{Y}$.

Therefore, the execution of the protocol $\Pi_{\mathsf{conc\text{-}a\text{-}ba}}$ does not encounter any deadlocks. Furthermore, by choosing appropriate parameters, specifically $m := \left\lceil \log_{\frac{1}{1-p}} N \right\rceil$, the protocol can achieve termination with a constant probability in each iteration. This implies that $\Pi_{\mathsf{conc\text{-}a\text{-}ba}}$ will eventually terminate with probability 1, and the expected number of iterations is constant. Since the number of rounds in each iteration is also constant, the overall round complexity of the protocol is constant in expectation.

Correctness. Now we demonstrate that once the protocol terminates, all honest parties output the same value as the ideal functionality $\mathcal{F}^{V,N}_{\mathsf{conc\text{-}a\text{-}ba}}$. This part of the proof requires the intrusion-tolerance property of A-BA, which guarantees that the output is either $\perp$ or a value provided by an honest party.

Since we assumed termination, there was agreement on $\vec{Y} = (y_1, \ldots, y_N)$. Since $\mathcal{F}_{\mathsf{a\text{-}ba}}$ models the intrusion-tolerance property, this implies that at least one honest party P_i provided $\vec{C}'_{l_i} = \vec{Y}$, where l_i was received from $\mathcal{F}_{\mathsf{a\text{-}ole}}$ and $P_{l_i} \in Q_{i,2}$, meaning that party P_{l_i} A-Casted a vector $\vec{C}'_{l_i} = (y_1, \ldots, y_N)$ satisfying $y_j \in S_{i,j}^{\mathsf{itr}+3}$ for all $j \in [N]$. This implies that y_j is a valid and non-empty (not λ) output from some session of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ for the jth instance, for each $j \in [N]$. According to the description of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$, if there is a unique input value that was provided by at least $n - 2t$ of the recorded parties of that session, everyone will receive that value as the output for the instance; otherwise, the output will be either one of the inputs provided by some honest parties for that instance or a special value $\perp$. Note that different sessions may have different sets of recorded parties, as each of them requires the participation of merely $n - t$ parties. Since $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$ tracks participation separately for each instance, and since $\mathcal{S}$ delays its output until the above sessions have been determined, $\mathcal{S}$ can set the input delays so that for every $j \in [N]$ the parties recorded for the jth instance of $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$, together with their recorded values, are exactly those of the session of $\mathcal{F}_{\mathsf{trunc\text{-}a\text{-}ba}}$ from which y_j originates. The two output rules then coincide instance by instance. Whenever the unique value provided by at least $n - 2t$ recorded parties exists, both that session and $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$ output it, and otherwise $\mathcal{S}$ sends $(\mathtt{replace}, \mathsf{sid}, (y_1, \ldots, y_N))$ to $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$, which is a legitimate choice since every y_j is either $\perp$ or an input of an honest party.

Therefore, with probability 1, no environment can distinguish between the execution of $\Pi_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N,m,p,\mathsf{itr}}$ with $\mathcal{A}$ and $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$ with $\mathcal{S}$, assuming expected-PPT machines. However, in order to apply UC composition guarantees, we need to introduce a stopping point (proportional to the security parameter) for our expected-PPT machines and make them strict PPT. In doing so, the distinguishing probability of the environment would become a non-zero value which can be arbitrarily small, depending on the duration for which we allow the machines to run. This ensures that the *strict* PPT protocol $\Pi_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N,m,p,\mathsf{itr}}$ UC-realizes $\mathcal{F}_{\mathsf{conc\text{-}a\text{-}ba}}^{V,N}$ with statistical security. □

Acknowledgements

Our original motivation for this project was to provide a simulation-based treatment of concurrent A-BA protocols, such as Ben-Or and El-Yaniv's [BE03], but the search for building blocks, in particular of an optimally resilient asynchronous OCC protocol became a bit of a "detective story," as many references pointed to an unpublished manuscript by Feldman [Fel89], which was nowhere to be found. We thank Michael Ben-Or for providing it to us, which corroborated its in-existence.

Ran Cohen's research is supported in part by NSF grant no. 2055568. Juan Garay's research is supported in part by NSF grants no. 2001082 and 2055694. Vassilis Zikas's research is supported in part by NSF grant no. 2055599 and by Sunday Group. The authors were also supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

Bibliography

[AAPS24] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Asynchronous agreement on a core set in constant expected time and more efficient asynchronous VSS and MPC. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 451–482. Springer, Cham, 2024.

[AC10] Hagit Attiya and Keren Censor-Hillel. Lower bounds for randomized consensus under a weak adversary. *SIAM Journal on Computing*, 39(8):3885–3904, 2010.

[ADH08] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *27th ACM PODC*, pages 405–414. ACM, 2008.

[ADS22] Ittai Abraham, Danny Dolev, and Gilad Stern. Revisiting asynchronous fault tolerant computation with optimal resilience. *Distributed Computing*, 35(4):333–355, 2022.

[AJM+23] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. *Distributed Computing*, 36(3):219–252, 2023.

[AMS19] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *38th ACM PODC*, pages 337–346. ACM, 2019.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, 1993.

[BE88] Michael Ben-Or and Ran El-Yaniv. Interactive consistency in constant expected time. Technical report, Inst. of Math. and Comp. Sci., Hebrew University, Jerusalem, 1988.

[BE03] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[Ben83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *2nd ACM PODC*, pages 27–30. ACM, 1983.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *30th FOCS*, pages 410–415. IEEE Computer Society Press, 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, 1988.

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 376–392. Springer, Berlin, Heidelberg, 2007.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *13th ACM PODC*, pages 183–192. ACM, 1994.

[Bor96] Malte Borcherding. Levels of authentication in distributed agreement. In *10th International Workshop on Distributed Algorithms WDAG*, pages 40–55. Springer, Berlin, Heidelberg, 1996.

[Bra87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[BS93] Donald Beaver and Nicol So. Global, unpredictable bit generation without broadcast. In *EUROCRYPT'93*, volume 765 of *LNCS*, pages 424–434. Springer, Berlin, Heidelberg, 1993.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

[BZL20] Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731. Springer, Cham, 2020.

[Can96] Ran Canetti. *Studies in secure multiparty computation and applications*. PhD thesis, Weizmann Institute of Science, 1996.

[Can20] Ran Canetti. Universally composable security. *Journal of the ACM*, 67(5):1–94, 2020.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, 1988.

[CCGZ19] Ran Cohen, Sandro Coretti, Juan Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. *Journal of Cryptology*, 32(3):690–741, 2019.

[CCGZ21] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. *Journal of Cryptology*, 34(2):12, 2021.

[CFG+23] Ran Cohen, Pouyan Forghani, Juan A. Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 422–451. Springer, Cham, 2023.

[CFG+25] Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Is it even possible? on the parallel composition of asynchronous MPC protocols. Cryptology ePrint Archive, Paper 2025/1707, 2025.

[CGHZ16] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 998–1021. Springer, Berlin, Heidelberg, 2016.

[CGZ23] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 3–38. Springer, Cham, 2023.

[CHM+22] Ran Cohen, Iftach Haitner, Nikolaos Makriyannis, Matan Orland, and Alex Samorodnitsky. On the round complexity of randomized byzantine agreement. *Journal of Cryptology*, 35(2):10, 2022.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, Berlin, Heidelberg, 2001.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[CMS89] Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *Journal of the ACM*, 36(3):591–614, 1989.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Berlin, Heidelberg, 2016.

[CPS19] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Round complexity of Byzantine agreement, revisited. IACR Cryptology ePrint Archive, Report 2019/886, 2019. https://eprint.iacr.org/2019/886.

[CPS20] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Sublinear-round byzantine agreement under corrupt majority. In *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 246–265. Springer, Cham, 2020.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51. ACM Press, 1993.

[CR98] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. Full version of [CR93], 1998. https://www.cs.tau.ac.il/~canetti/materials/cr93.ps.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[dSKT22] Luciano Freitas de Souza, Petr Kuznetsov, and Andrei Tonkikh. Distributed randomness from approximate agreement. In *36th DISC*, volume 246 of *LIPIcs*, pages 24:1–24:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[DWZ23] Sisi Duan, Xin Wang, and Haibin Zhang. FIN: practical signature-free asynchronous common subset in constant time. In *ACM CCS 2023*, pages 815–829. ACM, 2023.

[Fel88] Paul Neil Feldman. *Optimal Algorithms for Byzantine Agreement*. PhD thesis, Massachusetts Institute of Technology, 1988.

[Fel89] Paul Feldman. Asynchronous byzantine agreement in constant expected time. Unpublished manuscript, 1989.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *22nd ACM PODC*, pages 211–220. ACM, 2003.

[FGL+91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost NP-complete (preliminary version). In *32nd FOCS*, pages 2–12. IEEE Computer Society Press, 1991.

[Fit03] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2003.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, 1988.

[FM89] Paul Feldman and Silvio Micali. An optimal probabilistic algorithm for synchronous byzantine agreement. In *ICALP 89*, volume 372 of *LNCS*, pages 341–378. Springer, Berlin, Heidelberg, 1989.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[FN09] Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds sufficient for authenticated byzantine agreement. In *23rd DISC*, volume 5805 of *LNCS*, pages 449–463. Springer, 2009.

[GGL22] Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 96–119. Springer, Cham, 2022.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, 2007.

[GLL+22] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Efficient asynchronous byzantine agreement without private setups. In *42nd ICDCS*, pages 246–257. IEEE, 2022.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, 1987.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Berlin, Heidelberg, 2005.

[HNP08] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 473–485. Springer, Berlin, Heidelberg, 2008.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, Berlin, Heidelberg, 2010.

[KK09] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Berlin, Heidelberg, 2013.

[KY86] Anna R. Karlin and Andrew Chi-Chih Yao. Probabilistic lower bounds for Byzantine agreement and clock synchronization. Unpublished manuscript, 1986.

[LLM+20] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119. Springer, Cham, 2020.

[LLR02] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In *21st ACM PODC*, pages 203–212. ACM, 2002.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *Journal of the ACM*, 53(6):881–917, 2006.

[LLTW20] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *39th ACM PODC*, pages 129–138. ACM, 2020.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Mic17] Silvio Micali. Very simple and efficient byzantine agreement. In *ITCS 2017*, volume 4266 of *LIPIcs*, pages 6:1–6:1. Schloss Dagstuhl, 2017.

[MR90] Silvio Micali and Tal Rabin. Collective coin tossing without assumptions nor broadcasting. In *CRYPTO'90*, volume 537 of *LNCS*, pages 253–266. Springer, Berlin, Heidelberg, 1990.

[MR17] Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.

[Nie02] Jesper Buus Nielsen. A threshold pseudorandom function construction and its applications. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 401–416. Springer, Berlin, Heidelberg, 2002.

[Pat11] Arpita Patra. Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In *15th OPODIS*, volume 7109 of *LNCS*, pages 34–49. Springer, 2011.

[PCR14] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Asynchronous byzantine agreement with optimal resilience. *Distributed Computing*, 27(2):111–146, 2014.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PW92] Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In *9th STACS*, volume 577 of *LNCS*, pages 339–350. Springer, 1992.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409. IEEE Computer Society Press, 1983.

[SLM+23] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri Aravinda Krishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 554–584. Springer, Cham, 2023.

[TC84] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[WXDS20] Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 412–456. Springer, Cham, 2020.

[WXSD20] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. In *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 381–411. Springer, Cham, 2020.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, 1982.

[ZD22] Haibin Zhang and Sisi Duan. PACE: fully parallelizable BFT from reproposable byzantine agreement. In *ACM CCS 2022*, pages 3151–3164. ACM-CCS, 2022.

A Attack on Ben-Or and El-Yaniv's Select Protocol

In this section, we begin by presenting a static adversary that disproves an important claim regarding the expected-constant round complexity for one of the main subroutines in Ben-Or and El-Yaniv's concurrent A-BA protocol [BE03]. Consequently, the assured expected-constant round complexity of their concurrent A-BA protocol is no longer valid. Furthermore, we demonstrate that when considering adaptive adversaries, attempting to argue even for weaker claims becomes futile. The attacks we present here consider a simple case with only four parties, one of which can be corrupted (respecting the bound $t < n/3$). To provide essential context for our attacks, we present some of the protocols nearly verbatim from [BE03] in Figures 11, 12, and 13.

Next, we highlight that certain aspects of the attacks do not necessitate any corruption and can be executed solely by imposing appropriate delays. These aspects are common to both the static and adaptive attacks; thus, they form the initial focus of our analysis before we dive into the separate subsections for the static and adaptive attacks. Claims 6 and 7 formulate capabilities of adversaries only imposing delays maliciously.

Protocol $\Pi_{\text{a-cast}+}$

Common inputs: n, $t \leq (n-1)/3$; k (the identity of the transmitter)
Local input for processor p_i: none
Additional local input for processor p_k: V
Local output for processor p_i: $Value_i$, $Relay_i$

Epoch 1: (p_k only)

- A-Cast V.

Epoch 2: (Every processor p_i)

- Upon accepting a value v from a transmitter p, A-Cast ["$commit$", p, v].

Epoch 3: (Every processor p_i)

- Upon accepting $2t+1$ ["$commit$", p', v'] A-Casts from a set S of processors (transmitters), $Value_i \leftarrow v'$, $Relay_i \leftarrow S$, and return $Value_i, Relay_i$.

Figure 11: The A-Cast$^+$ protocol from [BE03, Fig. 3].

Protocol Π_{spread}

Common inputs: n, $t \leq (n-1)/3$
Local input for processor p_i: V_i
Local output for processor p_i: $\langle X_1, X_2, \ldots, X_n \rangle$

Epoch 1:

- A-Cast$^+$ V_i.

Epoch 2:

- Wait until $2t+1$ values $V_{i_1}, \ldots, V_{i_{2t+1}}$ accepted (from $p_{i_1}, \ldots, p_{i_{2t+1}}$).
- A-Cast $\langle (p_{i_1}, V_{i_1}), \ldots, (p_{i_{2t+1}}, V_{i_{2t+1}}) \rangle$.
  Participate in the A-Cast of a vector $u = \langle (p_{j_1}, U_{j_1}), \ldots, (p_{j_{2t+1}}, U_{j_{2t+1}}) \rangle$ only if for every $1 \leq k \leq 2t+1$, U_{j_k} accepted from p_{j_k}.

Epoch 3:

- Wait until $t+1$ vectors $u_{l_1}, \ldots, u_{l_{2t+1}}$ [sic] accepted (from $p_{l_1}, \ldots, p_{l_{t+1}}$) such that for every $1 \leq k \leq t+1$ if $u_{l_k} = \langle (p_{k_1}, U_{k_1}), \ldots, (p_{k_{2t+1}}, U_{k_{2t+1}}) \rangle$, and for every $1 \leq r \leq 2t+1$, U_{k_r} accepted from p_{k_r}'s A-Cast$^+$.
- For every $1 \leq l \leq n$:
  - If a value V was accepted from p_l via A-Cast$^+$ then $X_l \leftarrow (V, Relay_{i,l})$ (where $(V, Relay_{i,l})$ is the output of A-Cast$^+$).
  - Else $X_l := \emptyset$.
- Return $\langle X_1, X_2, \ldots, X_n \rangle$.

Figure 12: The Spread protocol from [BE03, Fig. 4].

Protocol Π_{select}

Common inputs: n, $t \leq (n-1)/3$
Local input for processor p_i: V_i - an arbitrary value; $Pred_i(x)$ - a predicate
Local output for processor p_i: D_i

Epoch 1:

- $U_i \leftarrow V_i$ and run Spread on input U_i and let $View_i$ denote the local output.

Epoch 2:

- Run A-OLE. Let k be the output denoting the identity of the leader, p_k.

Epoch 3:

- $U \leftarrow View_i[k]$ (i.e., the value "related" to p_k upon termination of Spread).
- If U was accepted during the run of Spread and $\models Pred_i(U)$, then $R_i \leftarrow U$ and $U_i \leftarrow$ ["$deflected$", $U, Relay_i$], where $Relay_i$ is obtained by Spread (via A-Cast$^+$) for the accepted broadcast of the leader p_k.
- Else $R_i \leftarrow \emptyset$.
- Run A-BA on input R_i. Let D_i denote the local output.
- If $D_i \neq \emptyset$ then return D_i.

Epoch 4:

- A-Cast $View_i$ ($View_i$ defined in Epoch 1).
  Participate in admissible A-Casts only.

Epoch 5:

- Wait until $2t+1$ views, $View_{i_1}, \ldots, View_{i_{2t+1}}$ accepted.
- A-Cast U_i.

Epoch 6:

- Wait until $2t+1$ values accepted.
- If a ["$deflected$", U', $Relay$]-message was accepted and $p_k \in Relay$ (p_k is the leader from Epoch 2) and there are $t+1$ processors in $Relay$ which have a non-null value related to p_k in their accepted Views with the same value U' (if several such messages were accepted with different U' values take an arbitrary one), then $R_i \leftarrow U'$.
- Else $R_i := \emptyset$.

Epoch 7:

- Run A-BA on input R_i and let D_i denote the local output.
- If $D_i \neq \emptyset$ then return D_i.
- Else $U_i \leftarrow V_i$ and go to Epoch 1.

Figure 13: The Select protocol from [BE03, Fig. 5].

Claim 6. *In any invocation of the $\Pi_{\text{a-cast}^+}$ protocol, the adversary can arbitrarily assign different sets of participants of size $n-t$ to the Relay sets of honest parties' outputs by imposing appropriate delays. No corruption is necessary for this manipulation.*

Proof. To assign a set S of participants to the *Relay* set of an honest party p_i, the adversary can allow all the honest parties to accept the A-Cast initiated in Epoch 1 of $\Pi_{\text{a-cast}^+}$. However, the adversary selectively delivers ["*commit*", $\cdot, \cdot$] messages from parties in set S to p_i before delivering any other ["*commit*", $\cdot, \cdot$] messages. □

Claim 7. *Let $P = \{p_1, p_2, p_3, p_4\}$ be the set of participants, v_i for $i \in [4]$ be the input of p_i, and $Relay_{i,j}$ for $i, j \in [4]$ be arbitrary subsets of P with size 3. The adversary can manipulate the output of the Π_{spread} protocol as follows solely through message scheduling. No corruption is necessary.*

$$
\begin{aligned}
p_1 &: \langle (v_1, Relay_{1,1}), (v_2, Relay_{1,2}), (v_3, Relay_{1,3}), (v_4, Relay_{1,4}) \rangle \\
p_2 &: \langle (v_1, Relay_{2,1}), (v_2, Relay_{2,2}), (v_3, Relay_{2,3}), (v_4, Relay_{2,4}) \rangle \\
p_3 &: \langle (v_1, Relay_{3,1}), (v_3, Relay_{3,3}), (v_4, Relay_{3,4}) \rangle \\
p_4 &: \langle (v_2, Relay_{4,2}), (v_3, Relay_{4,3}), (v_4, Relay_{4,4}) \rangle
\end{aligned}
$$

Proof. According to the description of Π_{spread}, for any $i, j \in [4]$, the jth component of the output of p_i is either $\perp$ or is the output of $\Pi_{\text{a-cast}^+}$ accepted from p_j (from Epoch 1). Since in each party's output all the available values corresponding to other parties are correct, the adversary does not need to make any changes to them. Moreover, based on Claim 6, the adversary can set the *Relay* set in the output of $\Pi_{\text{a-cast}^+}$ to any arbitrary subset of participants as long as it has the correct size, which is 3. Since all the $Relay_{i,j}$ sets are of size 3, the adversary will not face any challenges setting them. Now we only need to demonstrate how the adversary can include and exclude inputs from different parties in each party's output to achieve the desired event. The adversary can accomplish this by carefully designing the order in which the parties accept the $\Pi_{\text{a-cast}^+}$ messages from Epoch 1. The following steps outline the message delivery order:

- The adversary delivers the $\Pi_{\text{a-cast}^+}$ messages to p_1 in the order: p_1, p_3, p_4, p_2.
- The adversary delivers the $\Pi_{\text{a-cast}^+}$ messages to p_2 in the order: p_2, p_3, p_4, p_1.
- The adversary delivers the $\Pi_{\text{a-cast}^+}$ messages to p_3 in the order: p_1, p_3, p_4, and delays the $\Pi_{\text{a-cast}^+}$ message from p_2 until p_3 terminates.
- The adversary delivers the $\Pi_{\text{a-cast}^+}$ messages to p_4 in the order: p_2, p_3, p_4, and delays the $\Pi_{\text{a-cast}^+}$ message from p_1 until p_4 terminates.

By using this scheduling and delivering all the A-Cast$^+$ messages initiated in Epoch 2, the following outcomes occur:

- Parties p_1 and p_3 accept the vector $\langle (p_1, v_1), (p_3, v_3), (p_4, v_4) \rangle$ twice (once from p_1 and once from p_3 by the participation of p_1, p_2, p_3).
- Parties p_2 and p_4 accept the vector $\langle (p_2, v_2), (p_3, v_3), (p_4, v_4) \rangle$ twice (once from p_2 and once from p_4 by the participation of p_1, p_2, p_4).

Therefore, all parties terminate with the given output as stated in the claim. □

A.1 Static Attack

Π_{select} may involve multiple iterations, where parties either agree on a value and terminate, or restart the protocol. In the proof of the Π_{select} protocol, an event E is defined, which is claimed to occur with a constant probability. It is argued that when E happens in an iteration, all parties terminate. However, we demonstrate that this claim is not valid, resulting in uncertainty regarding the expected-constant round complexity guarantee of the Π_{select} protocol.

Assuming the preconditions for the Π_{select} protocol hold (i.e., for every non-faulty parties p_i and p_j, $\models Pred_j(V_i)$ and $V_i \neq \emptyset$), we will demonstrate that the argument regarding the round complexity is not valid. Let the static adversary $\mathcal{A}$ corrupt p_1 before the protocol begins. According to Claim 7, $\mathcal{A}$ can make parties terminate Π_{spread} in Epoch 1 of Π_{select} with the following outputs where $Relay_{1,2} = Relay_{4,2} = \{p_1, p_2, p_3\}$ (no corruption is required):

$$
\begin{aligned}
p_1 &: \langle (v_1, Relay_{1,1}), (v_2, Relay_{1,2}), (v_3, Relay_{1,3}), (v_4, Relay_{1,4}) \rangle \\
p_2 &: \langle (v_1, Relay_{2,1}), (v_2, Relay_{2,2}), (v_3, Relay_{2,3}), (v_4, Relay_{2,4}) \rangle \\
p_3 &: \langle (v_1, Relay_{3,1}), (v_3, Relay_{3,3}), (v_4, Relay_{3,4}) \rangle \\
p_4 &: \langle (v_2, Relay_{4,2}), (v_3, Relay_{4,3}), (v_4, Relay_{4,4}) \rangle
\end{aligned}
$$

Remark 8. Based on the properties of the Π_{spread} protocol, after the termination of the first honest party, there exist two subsets of participants, P_1 and P_2, satisfying the following conditions: P_1 only contains honest parties, $|P_1| = t + 1$, $|P_2| = 2t + 1$, and all the values from parties in P_2 are included in the output vector of all the parties in P_1.

Now let us assume that $\mathcal{A}$ allows p_4 to be the first to terminate Π_{spread} in Epoch 1 of the Π_{select} protocol. In this case, P_1 and P_2 could be $\{p_2, p_4\}$ and $\{p_2, p_3, p_4\}$, respectively.

In Epoch 2 of the Π_{select} protocol, a leader p_k is randomly chosen. The event E is defined as $p_k \in P_2 \cap (P - T)$ and it is claimed that if E occurs, the protocol terminates with the correct output. Since p_k is chosen randomly and $|P_2 \cap (P - T)| = t + 1$, the probability of the event E is at least $\frac{1}{3}$. Therefore, this claim suggests an expected-constant number of iterations (or asynchronous rounds).

However, we claim that when $p_k = p_2$, which satisfies E (since p_2 is honest and is in P_2), the protocol will start over instead of terminating. In the following, we explain the adversary's behavior in Epoch 3 to Epoch 7 of the Π_{select} protocol and provide a proof for our claim.

In Epoch 3 of Π_{select}, $\mathcal{A}$ can delay p_2 and assign a malicious value $v_2' \neq v_2$ to R_1. However, $\mathcal{A}$ allows p_1 to follow the protocol for the rest of its actions. As a result, the following state is observed:

- p_1: $R_1 = v_2'$ and $U_1 = (v_2, \{p_1, p_2, p_3\})$
- p_2: Delayed
- p_3: $R_3 = \perp$ and $U_3 = v_3$
- p_4: $R_4 = v_2$ and $U_4 = (v_2, \{p_1, p_2, p_3\})$

Consequently, the parties initiate the A-BA protocol with the inputs $(v_2', \text{delayed}, \perp, v_2)$. However, since there is no majority in the input values, we have no guarantee on the output of this A-BA protocol. In fact, as formulated in the A-BA ideal functionality, the adversary $\mathcal{A}$ can manipulate the protocol execution to make it output $\perp$ and prevent the termination of Π_{select}. Therefore, it is necessary to proceed to the next epoch, as the protocol cannot terminate at this stage.

In Epoch 4 of Π_{select}, $\mathcal{A}$ allows p_1 to follow the protocol. It should be noted that all the views in this epoch are admissible to everyone, as the components of all the views are either $\perp$ or the correct values. Therefore, all the views can be received by the honest parties. The only control that $\mathcal{A}$ has is over the order in which the views are received by the parties in the next epoch.

In Epoch 5 of Π_{select}, $\mathcal{A}$ allows p_1 to follow the protocol and strategically manages delays to achieve the following:

- Delivers $view_1, view_3, view_4$ to p_1 as its first $2t+1$ views.
- Delays the acceptance of any view for p_2 until the A-BA in Epoch 7 of Π_{select} is terminated.
- Delivers $view_1, view_3, view_4$ to p_3 as its first $2t+1$ views.
- Delivers $view_1, view_3, view_4$ to p_4 as its first $2t+1$ views.

Then, parties A-Cast values as follows based on the output they received from the Π_{spread} protocol in Epoch 1 of Π_{select}:

- p_1: $U_1 = (v_2, \{p_1, p_2, p_3\})$
- p_2: Still waiting from the previous epoch
- p_3: $U_3 = v_3$
- p_4: $U_4 = (v_2, \{p_1, p_2, p_3\})$

In Epoch 6 of Π_{select}, $\mathcal{A}$ lets p_1 follow the protocol, keeps p_2 waiting, and allows the remaining parties to receive each other's A-Cast messages. As a result, the following situation arises:

- p_1, p_3, p_4: Receive $(v_2, \{p_1, p_2, p_3\})$ from p_1, p_4, and v_3 from p_3.
- p_2: Still waiting from previous epochs

However, the "if" condition in Epoch 6 is not satisfied because only p_1 among $\{p_1, p_2, p_3\}$ has an accepted view that includes a value associated with p_2. Therefore, the following assignments are made:

- p_1, p_3, p_4: Assign $R_1 = R_3 = R_4 = \perp$.
- p_2: Still waiting from previous epochs

In Epoch 7 of Π_{select}, $\mathcal{A}$ allows p_1 to follow the protocol. Consequently, parties initiate the A-BA protocol with the following inputs:

- p_1, p_3, p_4: $\perp$
- p_2: Still waiting from previous epochs

As a result, parties complete the A-BA protocol and obtain the output $\perp$. Since the output is $\perp$, they have to start over, completing our attack.

A.2 Adaptive Attack

In the static attack, we demonstrated that the claim "for all possibilities for the set P_2, if the leader is an honest party in P_2, then the protocol terminates with the correct value" is not valid. We provided specific choices of the leader within certain possibilities for P_2 where the claim does not hold. This shows that the claim is incorrect in those cases.

In the adaptive attack, we further illustrate the difficulty in finding an argument for the weaker claim "there exists a possibility for P_2 such that if the leader is an honest party in P_2, then the protocol terminates with the correct values." We use the same setup with four parties, namely p_1, p_2, p_3, and p_4, and demonstrate that if either p_1 or p_2 is chosen as the leader, the adversary $\mathcal{A}$ can manipulate the protocol to start over instead of terminating. Since any legitimate set P_2 consists of three parties, it must include at least one of p_1 and p_2. Therefore, in any possible set P_2, there is an honest party whose selection as the leader does not guarantee termination. This shows the hopelessness of finding a valid argument for the weaker claim.

Assuming the preconditions for the Π_{select} protocol hold, we now consider adaptive corruption in this subsection. The adversary $\mathcal{A}$ does not corrupt any party before the leader is selected. If the chosen leader is p_2, then $\mathcal{A}$ follows the description of the static attack by corrupting p_1 and executing the attack as previously described.

However, if party p_1 is chosen as the leader, we can leverage symmetry and modify the adversary's description accordingly. We rename the parties as follows: $p'_1 := p_2$, $p'_2 := p_1$, $p'_3 := p_4$, and $p'_4 := p_3$. The adversary $\mathcal{A}$ corrupts p'_1 and runs the same static adversary code on the participant set $\{p'_1, p'_2, p'_3, p'_4\}$. Furthermore, it is important to note that in the static attack, we previously specified that $Relay_{1,2} = Relay_{4,2} := \{p_1, p_2, p_3\}$, while allowing $Relay_{2,1}$ and $Relay_{3,1}$ to be arbitrary since they were not essential to the attack and could take any value. However, in order to achieve full symmetry in the static attack and apply the aforementioned approach, $\mathcal{A}$ should now set $Relay_{2,1} = Relay_{3,1} := \{p_1, p_2, p_4\}$ in the description of the static attack.

With these adjustments, our adaptive adversary is complete.

B The Asynchronous Turpin-Coan Extension

Here, we investigate when the generic extension of binary to multi-valued synchronous BA (for $t < n/3$) given by Turpin and Coan [TC84] works in the asynchronous setting with eventual delivery. It turns out that an asynchronous version of the extension—with appropriate modifications—is secure when $t < n/5$, but is provably insecure for any $t \geq n/5$ regardless of which binary A-BA protocol is used. We remark that while our negative result is not exactly a lower bound, as the attack that we present is on a specific extension protocol, the fact that we lose two additive factors of t in resiliency (and even three, with a more naïve approach) gives some evidence that more sophisticated techniques are needed to maintain optimal resiliency, as in [MR17]. Since our primary goal is to show that the Turpin-Coan extension cannot be used to get optimally resilient, multi-valued A-BA in expected-constant rounds, we choose to work with simpler, property-based definitions in this section.

We start by reviewing the original extension in [TC84], which requires just two additional rounds. Parties first distribute their inputs amongst one another, over P2P channels. An honest party, based on how many of the received values disagree with his own, considers himself either "perplexed" or "content," and announces this information. If enough parties claim to be perplexed, the honest party becomes "alert." The parties now run a binary BA protocol to agree on this last state (effectively, to determine if they all started with the same input), and depending on the

outcome they either output a default value, or are able to recover a common value from their local transcripts of the protocol.

Protocol $\Pi^V_{\mathsf{a\text{-}tc}}$

Local input of party p_i: $v_i \in V$, $|V| \geq 2$
Local output of party p_i: $v \in V$
Code for party p_i:

0. Initialize arrays $v[1..n]$ and $x[1..n]$ by filling them with a default value $\perp$.
1. **A-Cast** v_i.
2. Whenever v'_j is accepted from party p_j, record $v[j] := v'_j$.
3. Wait until $n - t$ values are accepted. Then, let $x_i := 1$ ("perplexed") if $v[j] \neq \perp$ and $v[j] \neq v_i$ for at least $t + 1$ different indices j, and otherwise let $x_i := 0$ ("content"). **A-Cast** (v_i, x_i).
4. Whenever (v'_j, x'_j) is accepted from party p_j, record $x[j] := x'_j$. If $v[j] = \perp$, then additionally record $v[j] := v'_j$.
5. Wait until $n - t$ pairs are accepted. Then, let $a_i := 1$ ("alert") if $x[j] = 1$ for at least $t + 1$ different indices j, and otherwise let $a_i := 0$. Run **binary A-BA** with input a_i.
6. If the binary computation agrees 1, then set v to be a predefined default value from V. Otherwise, if content set $v := v_i$, and if perplexed set v to be the majority value from the non-bottom entries in $v[]$ at indices j for which $x[j] = 0$. **Output** v.

Figure 14: The asynchronous Turpin-Coan extension.

Our asynchronous version of this extension, $\Pi_{\mathsf{a\text{-}tc}}$, is shown in Figure 14. We simply replace the basic message distribution mechanism with A-Cast, and account for the fact that adversarial A-Cast's may never terminate (i.e., honest parties should only expect to receive $n - t$ values). To avoid a somewhat artificial attack wherein the adversary can manipulate delays so that t entries are missing in an honest party's $v[]$ array and a *different* set of t entries are missing in the $x[]$ array, we require parties to re-send their input value in the second A-Cast. Note that this additional modification cannot *decrease* security, since honest parties will always use their actual input. In the following proposition, we prove the security of $\Pi_{\mathsf{a\text{-}tc}}$ for $t < n/5$. We emphasize that in the asynchronous setting we must impose relaxed termination requirements. However, since the modified extension is still deterministic and constant-round, any termination guarantees provided by the underlying binary A-BA protocol should be preserved; accordingly, we only prove the agreement and validity properties.

Proposition 9. *For any input domain V, protocol $\Pi^V_{\mathsf{a\text{-}tc}}$ securely implements multi-valued A-BA from A-Cast and binary A-BA, in constant rounds and in the presence of an adaptive and malicious t-adversary, provided $t < n/5$.*

Proof. The round complexity is easily established by inspection (considering A-Cast and binary A-BA to be ideal primitives). We now prove correctness:

Validity. If all honest parties begin with the same initial value v, then no honest party is perplexed and all honest parties are not alert. The binary computation agrees 0 and all honest parties (which are content) output v.

Agreement. There are two cases: the binary computation agrees either 1 or 0. In the former case, all honest parties output a common default value. For the latter case, we show that all content

parties have the same initial value, and that this value is correctly deduced by all the perplexed parties.

Note that each content party's input appears in at least $n - t - t = n - 2t$ positions in its $v[]$ array. Since at most t of these positions correspond to corrupted parties, we have that each content party has the same initial value as at least $n - 3t$ honest parties, which represent a majority of the honest parties when $n > 5t$. Of course there cannot be two distinct majorities, so all content parties must have the same initial value.

Furthermore, since the binary computation agrees 0, there must be at least $2t+1$ honest content parties, for otherwise there would be at least $n-3t$ honest perplexed parties and all honest parties would be alert (at least $n - 4t \geq t + 1$ of the corresponding entries in any honest party's $x[]$ array would not be $\perp$) and agreement would be on 1. Each perplexed party has $x[j] = 0$ or $x[j] = \perp$ for all honest content parties and possibly for some corrupted parties. Even in the worst case, when $x[j] = \perp$ for t of the honest content parties, there are still at least $2t + 1 - t = t + 1$ honest content parties left but at most t corrupted parties, so the content parties are a majority of those for which $x[j] = 0$ and $v[j] \neq \perp$. □

It is not difficult to see that when the predefined default value comes from outside of V, the resulting multi-valued A-BA protocol is actually *intrusion-tolerant* (see Section 1.2). We now show that $\Pi_{\mathsf{a\text{-}tc}}$ is insecure when $t \geq n/5$ (i.e., that the analysis above is tight). Our attack does not make any assumptions regarding the underlying binary A-BA protocol, other than that it satisfies the properties of A-BA. Moreover, we require only static corruptions, and somewhat surprisingly the attack considers only binary inputs (i.e., using the extension when $|V| = 2$ actually breaks the security of the binary protocol for $n/5 \leq t < n/3$).

Proposition 10. *For all input domains V, protocol $\Pi^V_{\mathsf{a\text{-}tc}}$ does NOT securely implement multi-valued A-BA from A-Cast and binary A-BA, in the presence of a static and malicious t-adversary, for any $t \geq n/5$.*

Proof. It suffices to consider the case $n = 5t$. We construct a distribution of inputs, a static adversary $\mathcal{A}$, and a message scheduling such that agreement is broken.

Let $\{p_1, \ldots, p_{5t}\}$ be the set of parties. Define four sets $A := \{p_1, \ldots, p_{2t}\}$, $B := \{p_{2t+1}, \ldots, p_{3t}\}$, $T := \{p_{3t+1}, \ldots, p_{4t}\}$, and $C := \{p_{4t+1}, \ldots, p_{5t}\}$. The parties in T are statically corrupted by $\mathcal{A}$ and instructed to start the protocol with input 0. Suppose the parties in A and B start with input 1, and parties in C with input 0. Now consider a message scheduling in which: 1) both A-Casts from parties in C get delayed for parties in A, 2) the A-Casted input values from $p_1, \ldots, p_t \in A$ get delayed for parties in B and C, 3) the A-Casted pairs from parties in C get delayed for parties in B, and 4) the A-Casted pairs from $p_{t+1}, \ldots, p_{2t} \in A$ get delayed for parties in C.

Observe that all parties in A will A-Cast that they are content. $\mathcal{A}$ also instructs parties in T to maliciously A-Cast that they are content. On the other hand, the parties in both B and C will A-Cast that they are perplexed. Then, no party in A or B will be alert, whereas all parties in C will be alert. We summarize all this information in Figure 15.

Consequently, if $\mathcal{A}$ instructs parties in T to run the binary A-BA protocol with input 0 (not alert), and delays messages from parties in C during the computation, then agreement will be on 0. Therefore, parties in A will output their initial value 1, while the parties in B and C will try to deduce the correct output from their arrays. All parties in B can do this successfully ($2t$ of the $3t$ relevant entries in the $v[]$ array will be 1). However, for parties in C, t of the $2t$ relevant entries will be 1 while the other t will be 0. □

It is possible to show that without the small modification in which parties A-Cast their input twice, the protocol is secure if and only if $t < n/6$. We have a method to extend binary to

	A_1	A_2	B	T	C	
A	1 $(1, \mathrm{c})$	1 $(1, \mathrm{c})$	1 $(1, \mathrm{p})$	0 $(0, \mathrm{c})$		not alert
B		1 $(1, \mathrm{c})$	1 $(1, \mathrm{p})$	0 $(0, \mathrm{c})$	0 	not alert
C		1 	1 $(1, \mathrm{p})$	0 $(0, \mathrm{c})$	0 $(0, \mathrm{p})$	alert

Figure 15: Summary of the A-Casts received by parties in A, B, and C, and their resulting alert status. The content and perplexed values are denoted by "c" and "p," respectively. Note that each column corresponds to a set of t parties (A_1 and A_2 denote $\{p_1, \ldots, p_t\}$ and $\{p_{t+1}, \ldots, p_{2t}\}$, respectively).

multi-valued A-BA in constant rounds for $t < n/4$, which is not much more complicated than the synchronous extension in [TC84] (although it requires *two* invocations of the binary protocol). However, as discussed in Section 3.2, to re-gain optimal resiliency $t < n/3$ we use the asynchronous extension of Mostéfaoui and Raynal [MR17]. We leave open the possibility of obtaining expected-constant-round multi-valued A-BA via other approaches, such as substituting our multi-valued OCC in the binary A-BA protocol from [CR93] (and making appropriate changes). In any case, we prefer the solution in [MR17] as it is a *black-box* reduction.