A Theoretical Take on a Practical Consensus Protocol

Victor Shoup[1]

Offchain Labs
victor@shoup.net

May 17, 2024

Abstract. The Asynchronous Common Subset (ACS) problem is a fundamental problem in distributed computing. Very recently, Das et al. (2024) developed a new ACS protocol with several desirable properties: (i) it provides optimal resilience, tolerating up to $t < n/3$ corrupt parties out of n parties in total, (ii) it does not rely on a trusted set up, (iii) it utilizes only "lighweight" cryptography, which can be instantiated using just a hash function, and (iv) it has expected round complexity $O(1)$ and expected communication complexity $O(\kappa n^3)$, where κ is the output-length of the hash function. The purpose of this paper is to give a detailed, self-contained exposition and analysis of this protocol from the point of view of modern theoretcal cryptography, fleshing out a number of details of the definitions and proofs, providing a complete security analysis based on concrete security assumptions on the hash function (i.e., without relying on random oracles), and developing all of the underlying theory in the universal composability framework.

1 Introduction

The **Asynchronous Common Subset (ACS)** problem is a fundamental problem in distributed computing. Roughly speaking, an ACS protocol allows n parties — up to t of which may be corrupt (Byzantine), and communicating asynchronously (with no timing assumptions) over secure point-to-point channels — to agree on a set X of $n - t$ parties, where each party in X satisfies some validity predicate. For example, each party in X may be a sender that has successfully sent a message via a reliable broadcast protocol, or a dealer that has successfully shared a secret via a verifiable secret sharing protocol.

Very recently, [DDL$^+$24] developed a new ACS protocol with several desirable properties:

(i) it provides optimal resilience, tolerating up to $t < n/3$ corrupt parties,
(ii) it does not rely on a trusted set up (apart from secure point-to-point channels),
(iii) it utilizes only "lightweight" cryptography, which can be instantiated using just a hash function, and
(iv) it has expected round complexity $O(1)$ and expected communication complexity $O(\kappa n^3)$, where κ is the output-length of the hash function.

This is, in fact, the *first* ACS protocol in the literature with *all* of these properties. For example, [AJM$^+$21,GLL$^+$21] satisfy all but (iii), as their protocols rely on pairing-based elliptic-curve cryptography. Relying only on lightweight cryptography means that the protocol in [DDL$^+$24] is very efficient computationally and post-quantum secure.

The purpose of this paper is to give a detailed, self-contained exposition and analysis of this new protocol from the point of view of modern theoretical cryptography, fleshing out a number of details of the definitions and proofs, and developing all of the underlying theory in the universal composability framework. Also, while the analysis in [DDL+24] was mainly done in the random oracle model, we provide a complete analysis strictly in the standard (uniform) computational model, where we make specific, concrete assumptions on hash functions; specifically, we need *collision resistance* and *linear hiding*, the latter being an assumption introduced in [SS23], which essentially says that a keyed version of the hash function acts a pseudorandom function under a specific type of related-key attack. We also need a standard *pseudorandom function* (which can, if desired, also be instantiated using a hash function).

We stress that the purpose of this paper is purely exposition and analysis — although our description of the protocol is slightly different than in [DDL+24], all of the novel protocol techniques are already in [DDL+24], and we do not make any improvements to that protocol here. We do, however, provide a probabilistic analysis of the number of iterations of the main loop of the protocol, estimating expectation and tail bounds, based on reductions to the above-mentioned cryptographic assumptions. This analysis is new and may be of independent interest as it may be easily adapted to other protocols with a similar probabilistic, iterative structure.

We refer the reader to [DDL+24] for more motivation, a comparison to other work in the literature, as well as a report on the results of an experimental evaluation of the protocol. We also stress that while our point of view in this paper is somewhat theoretical, the protocol itself is quite practical.

The rest of the paper. Section 2 reviews basic definitions as well as building blocks from the literature (or minor variations thereof). This section also reviews the related problem of VABA (validated asynchronous Byzantine agreement) and the well-known reduction from ACS to VABA, which is also the approach to ACS taken here. So the main innovation of the new ACS protocol is actually a new VABA protocol. Section 3 introduces the new subprotocols that are the key ingredients in the new VABA protocol: a Gather protocol with a new binding property and a secret sharing protocol with security properties tailored specifically to our needs. Both of these subprotocols may be of independent interest and likely have other applications. Section 4 describes and analyzes the new VABA protocol. Section 5 discusses how such an ACS protocol would likely be used in practice — namely, as a way of bootstrapping an asynchronous distributed system by generating a number of “random beacon” instances, thus providing the infrastructure needed to run other protocols (including even more efficient ACS or VABA protocols, as well as distributed key generation protocols).

2 Preliminaries

2.1 System model

We have n parties $P_1, \ldots, P_n$, of which at most $t < n/3$ may be corrupt. We assume *static* corruptions. Let $\mathcal{H}$ denote the indices of the honest parties, and let $\mathcal{C}$ denote the indices of the corrupt parties. The restriction to static corruptions is mainly for simplicity of presentation — all of the protocols presented here should be secure in the adaptive corruption model, but we do not explore that here.

We assume the parties are connected by secure point-to-point channels, which provide both privacy and authentication. As we are working exclusively in the asynchronous communication model, there is no bound on the time required to deliver messages between honest parties.

We use the following fairly standard notation for intervals of integers: for integer k, we write $[k]$ for the set $\{1, \ldots, k\}$.

2.2 Uniformly bounded communication and computational complexity

As usual, the **communication complexity** of a protocol is the sum over all honest parties P of the number of bits that P sends to any other party, and the **message complexity** is the sum over all honest parties P of the number of messages that P sends to any other party.

All of the protocols we discuss in this paper must satisfy a **uniformly bounded communication complexity** property. This property says that the communication complexity is bounded by a *fixed* polynomial (that is, not depending on the adversary) in the security parameter and the number of parties, at least with all but negligible probability for any computationally bounded adversary.

We can define an analogous notion of **uniformly bounded message complexity**, where we simply replace communication complexity by message complexity. Of course, if the communication complexity is uniformly bounded, then so it the message complexity.

All of the protocols we discuss must also satisfy a **uniformly bounded computational complexity** property. This property says that the **computational complexity** (that is, the total amount of computation performed by all honest parties) is bounded by a *fixed* polynomial (that is, not depending on the adversary) in the total number of bits received by all honest parties, at least with all but negligible probability for any computationally bounded adversary.

Such notions were introduced in [CKPS01] as a way to properly analyze protocols in a cryptographic setting where we need to consider computationally bounded adversaries. In this paper, we shall combine these two properties and say that a protocol has **uniformly bounded communication and computational complexity** if it satisfies both properties. One nice aspect of this notion is that it is closed under protocol composition. Another is that it guarantees that with overwhelming probability, the amount of computation performed in any attack by all parties, including honest parties and the adversary, is polynomially bounded (with all but negligible probability). This ensures that we can make use of cryptographic assumptions in the analysis of such protocols.

Note that in the above definitions are defined in terms of a game played between a computationally bounded adversary and the protocol. In the UC framework [Can00], this game would take place in the secure-channels hybrid model, and this adversary would actually comprise

- the environment, which supplies inputs to and receives outputs from protocol machines belong to honest parties, and
- the so-called “dummy adversary”, which acts as a simple relay between protocol machines and the environment (so the environment itself ultimately controls the protocol messages sent between parties).

Our definitions here are generally compatible with the definitions of polynomial runtime in [HUMQ09,HS11], and as such, we can be sure that composability properties are maintained in a way that is largely independent of which particular variant of the UC framework is used.

2.3 Reliable broadcast

A **reliable broadcast (RBC)** protocol allows a sender S to broadcast a single message m to $P_1, \ldots, P_n$ in such a way that all parties are guaranteed to receive the same message. More precisely, in addition to having uniformly bounded communication and computational complexity (see Section 2.2), an RBC protocol should satisfy a *security* property and a *completeness* property. The security property is captured by the ideal functionality $\mathfrak{F}_{\mathrm{RBC}}$ in Fig. 1.

$\mathfrak{F}_{\mathrm{RBC}}$

$\mathsf{Input}(m)$: this operation is invoked once by the sender S, who inputs a message m. In response, $\mathfrak{F}_{\mathrm{RBC}}$ sends the message $\mathsf{NotifyInput}(m)$ to the ideal-world adversary.

$\mathsf{RequestOutput}(i)$: after the input has been received, this operation may be invoked by the ideal-world adversary, who specifies $i \in [n]$. In response, $\mathfrak{F}_{\mathrm{RBC}}$ sends to P_i the message $\mathsf{Output}(m)$.

Fig. 1. The Reliable Broadcast ideal functionality (parameterized by S)

The completeness property says that: if one honest party generates an output, or if S is honest and supplies an input, then every honest party *eventually* generates an output. Here, *eventually* means if and when all messages sent from one honest party to another have been delivered.

Note that without uniformly bounded message complexity, the completeness property would be trivially satisfied by any protocol in which there are always more messages that need to be delivered.

Note that this notion of completeness is a specific property of a concrete protocol. We do not attempt to capture this via an ideal functionality in the UC framework. This approach to modeling completeness, and the related notion of liveness, adheres to the approach introduced in [CKPS01] and used more recently, for example, in [SS23,GS23]. In particular, we do not attempt to use the formal notion of time in the UC framework introduced in [KMTZ11] to model completeness or liveness. Our view is that this extra (and, in our opinion, somewhat complicated) machinery is unnecessary and that these notions are more simply and quite adequately modeled as properties of concrete protocols (as we have done so here).

A well-known Reliable Broadcast protocol is due to Bracha [Bra87]. While Bracha's protocol is simple and only requires a small constant number of rounds of communication, its communication complexity is $O(|m|n^2)$, which for large messages is far from optimal. There are many other RBC protocols in the literature that have better communication complexity, while still needing only a constant number of rounds of communication. One such protocol is that from [CT05], which has a communication complexity of $O(|m|n + \kappa n^2 \log n)$, where κ is the output length of a collision-resistant hash function. Another such protocol is from [DXR21], which has a communication complexity of $O(|m|n + \kappa n^2)$. The protocol from [CT05] is simpler and a bit more computationally efficient, and so may be preferable to that in [DXR21] when $|m| \gg \kappa n \log n$. However, in all of our applications in this paper, we will have $|m| = O(\kappa n)$, and so the construction in [DXR21] is preferable.

Note: as a technical aside, in order for these protocols to satisfy our definition of uniformly bounded communication complexity in Section 2.2, we will require that messages m are of some length bounded by a fixed polynomial (determined by the protocol, not by the adversary) in the security parameter. This will be the case for all of the applications of RBC in this paper.

2.3.1 One-sided voting. A degenerate version of Bracha broadcast can be used as a simple *one-sided voting* protocol, and is given in Fig. 2. This is a well-known technique — see, for example, [SS23], which is the basis for our presentation here. In this protocol, each party may *provide support* by supplying a $\mathsf{support}$ input to the protocol, and may *accept* by generating a accept output. The key security property of this protocol is as follows:

If any honest party accepts, then at least $n-t-t'$ honest parties must have provided support, where $t' \leq t$ is the number of corrupt parties.

This protocol also satisfies the following *completeness* property: if all honest parties provide support or some honest party accepts, then *eventually*, all honest parties accept. As usual, *eventually* means if and when all messages sent between honest parties have been delivered.

This protocol obviously has communication complexity $O(n^2)$.

```
$\Pi_{\text{OneSidedVote}}$

// Party $P_i$
acquired ← false, voted ← false, done ← false

while not done do
    wait until either:
        not acquired and received input support ⇒
            acquired ← true
            send (echo) to $P_1, \ldots, P_n$

        not voted and received (echo) from $n - t$ distinct parties ⇒
            send (vote) to $P_1, \ldots, P_n$
            voted ← true

        not voted and received (vote) from $t + 1$ distinct parties ⇒
            send (vote) to $P_1, \ldots, P_n$
            voted ← true

        voted and received (vote) from $n - t$ distinct parties ⇒
            // output stage
            output accept
            done ← true
```

Fig. 2. Degenerate version of Bracha's protocol for one-sided voting

2.4 Asynchronous Common Subset

In an **Asynchronous Common Subset (ACS)** protocol, the parties $P_1, \ldots, P_n$ agree in a set $X \subseteq [n]$ of party indices of size (at least) $n - t$. The indices in X must each satisfy a *validity predicate*. This validity predicate is essentially an "asynchronous validity predicate" as in [AAPS23]. However, to make our definitions compatible with the UC framework [Can00], we need to adjust the mechanics slightly.

As the protocol proceeds, each party P_i may invoke the operation $\mathsf{Input}(\mathsf{validate}, j)$ for $j \in [n]$ — party P_i may do this several times, for various indices j. In this case, we say P_i **validates** P_j, and intuitively, it means that party P_i sees that the index j satisfies the validity predicate. We say that a party P_j has been **locally validated** if it has been validated by *some* honest party, and we say that P_j has been **globally validated** if it has been validated by *all* honest parties.

The connection to the notion of an "asynchronous validity predicate" validated_i in [AAPS23] is that when "P_i validates P_j" in our framework is equivalent to "$\text{validated}_i(j)$ returns 1" in

[AAPS23]. The main difference is that in our framework, we explicitly outsource the validation to the environment, and instead of having some kind of "asynchronous callback mechanism", as in [AAPS23], we simply have the environment supply an input signal when the validation occurs. Also, in our applications, we essentially only require that $\mathrm{validated}_i(j)$ returns 1 or returns nothing at all.

In addition to having uniformly bounded communication and computational complexity (see Section 2.2), an ACS protocol should satisfy a *security* property and a *liveness* property. The security property is captured by the ideal functionality $\mathfrak{F}_{\mathrm{ACS}}$ in Fig. 3.

$\mathfrak{F}_{\mathrm{ACS}}$

$\mathsf{Input}(\mathsf{validate}, j)$: this operation may be invoked by each party P_i, where $j \in [n]$. In response, $\mathfrak{F}_{\mathrm{acs}}$ sends the message $\mathsf{NotifyInput}(\mathsf{validate}, i, j)$ to the ideal-world adversary.

$\mathsf{RequestOutput}(i, X)$: this operation may be invoked by the ideal-world adversary, one per $i \in [n]$. In response, $\mathfrak{F}_{\mathrm{acs}}$ sends to P_i the message $\mathsf{Output}(X)$. The value X must be a subset of $[n]$ of size (at least) $n - t$ where P_i has already validated P_j for all $j \in X$. Moreover, all invocations of this operation must specify the same set X.

Fig. 3. The Asynchronous Common Subset ideal functionality

The liveness property says that if at some point in time the set of locally validated parties has size at least $n - t$, then *eventually* all honest parties produce an output. Here, *eventually* means if and when all messages sent from one honest party to another have been delivered and all parties that have been locally validated have also been globally validated.

Again, we note that without the uniformly bounded message complexity property, the liveness property would be trivially satisfied by any protocol in which there are always more messages that need to be delivered.

Our definition of liveness works well together with other notions of completeness and liveness. For example, an ACS protocol may be "driven" by a reliable broadcast (RBC) protocol (see Section 2.3), where we run n instances of RBC, and each party plays the role of sender in exactly one of them. We can then run ACS where P_i *validates* P_j means that P_i has reliably received a message m_j from P_j — that is, has obtained an output message m_j from the RBC instance in which P_j is the sender. The completeness property for RBC guarantees that in the combined protocol, if all honest parties broadcast a value, then if and when all messages sent from one honest party to another have been delivered, then all parties will have output a value in the ACS protocol. The same holds true if we replace RBC by another protocol with a similar completeness property, such as (complete) verifiable secret sharing.

2.5 Gather

A **Gather** protocol has the same API as an ACS protocol, but with weaker guarantees. Each party P_i outputs a set X_i of size at least $n - t$, but these sets need not be the same. However, these sets must all contain a common "core set" X of size (at least) $n - t$. Moreover, this core set must already be determined by the time the first honest party generates an output — this is sometimes called *Gather with binding core.*

In addition to having uniformly bounded communication and computational complexity (see Section 2.2), a Gather protocol should satisfy a *security* property and a *liveness* property. The security property is captured by the ideal functionality $\mathfrak{F}_{\text{Gather}}$ in Fig. 4. The liveness property is identical to that for ACS.

$\mathfrak{F}_{\text{Gather}}$

$\mathsf{Input}(\mathsf{validate}, j)$: this operation may be invoked by each party P_i, where $j \in [n]$. In response, $\mathfrak{F}_{\text{acs}}$ sends the message $\mathsf{NotifyInput}(\mathsf{validate}, i, j)$ to the ideal-world adversary.

$\mathsf{RequestOutput}(i, X_i, X)$: this operation may be invoked by the ideal-world adversary, one per $i \in [n]$. In response, $\mathfrak{F}_{\text{acs}}$ sends to P_i the message $\mathsf{Output}(X_i)$. The value X_i must be a subset of $[n]$ such that P_i has already validated P_j for all $j \in X_i$. The "core set" X must be a subset of X_i of size at least $n - t$. Moreover, all invocations of this operation must specify the same "core set" X.

Fig. 4. The Gather (with binding core) functionality

2.5.1 A Gather protocol. Here is a basic (core-binding) Gather protocol, included here mainly for completeness (and which is adapted from a related protocol in [DWZ23]).

We let Valid_i denote the set of indices of parties that party P_i has validated (this set grows over time).

1. Each party P_i waits until $|\mathsf{Valid}_i| \geq n - t$, and then broadcasts (first, S_i), where $S_i \coloneqq \mathsf{Valid}_i$.
2. Each party P_i waits until it receives an "ack" message from $n - t$ distinct parties, and then broadcasts (second, T_i), where $T_i \coloneqq \mathsf{Valid}_i$. (See below for how "ack" messages are generated.)
3. Each party P_i waits to receive messages (second, T_j) with $T_j \subseteq \mathsf{Valid}_i$ from a collection $\{P_j\}_{j \in J_i}$ of $n - t$ distinct parties, and then outputs $X_i \coloneqq \bigcup_{j \in J_i} T_j$.

Concurrently to the above, each party P_i sends an "ack" message to party P_j whenever the following condition holds:

- P_i has not already sent an "ack" message to P_j,
- P_i has received a unique message of the form (first, S_j) from P_j, and
- $S_j \subseteq \mathsf{Valid}_i$.

Let us call this protocol $\mathit{\Pi}_{\text{BasicGather}}$. It is clear that $\mathit{\Pi}_{\text{BasicGather}}$ has message complexity $O(n^2)$, communication complexity $O(n^3)$. As for security and liveness, we have the following:

Theorem 2.1. *$\mathit{\Pi}_{\text{BasicGather}}$ securely emulates $\mathfrak{F}_{\text{Gather}}$ and satisfies the Gather liveness property.*

Proof. The main thing to check is that the sets output by each party contain a core of size $n - t$ that is determined by the time the first party produces an output. Consider the first point in time at which an honest party, say P_i, finishes Step 2. At this time, P_i has received "ack" messages from $n - t$ parties, and at least $n - 2t$ of those are honest, call them $\{P_j\}_{j \in J^*}$. For each $j \in J^*$, party P_j will include S_i in its second set T_j. Moreover, since $t < n/3$, for any honest party P_k in Step 3, we must have $J_k \cap J^* \neq \emptyset$, and so all honest parties will include S_i in their output. So this set S_i is the required core.

To show that $\mathit{\Pi}_{\text{BasicGather}}$ securely emulates $\mathfrak{F}_{\text{Gather}}$, the simulator in the ideal world just runs the actual protocol and computes the core set as described above.

The proof of liveness is straightforward and is omitted. □

2.6 Validated Asynchronous Byzantine Agreement

We can generalize the notion of an ACS protocol (see Section 2.4) so that the output set X is only required to be of size k, where $1 \leq k \leq n - t$ is a parameter. In this generalization, the ideal functionality $\mathfrak{F}_{\mathrm{ACS}}$ in Fig. 3 is modified so that the set X is required to be of size at least k (instead of $n - t$), and the liveness definition is modified so that the precondition is that at least k (instead of $n - t$) parties have been locally validated. If we set $k \coloneqq 1$, this is called a **Validated Asynchronous Byzantine Agreement (VABA)** protocol.

ACS and VABA are equivalent, in the sense that we can build one from the other. One direction is entirely trivial: given an ACS protocol, we can build a VABA protocol by simply running ACS to get X and then output (say) the smallest index in X. In the other direction, following [CKPS01], we can build an ACS protocol from a VABA protocol and an RBC protocol (see Section 2.3) as follows. We assume (without loss of generality) that the VABA protocol always outputs a set of size exactly equal to 1. As a notational convenience, we let Valid_i denote the set of indices of parties that party P_i has validated (this set grows over time).

1. Each party P_i waits until $|\mathsf{Valid}_i| \geq n - t$, and then reliably broadcasts $S_i \coloneqq \mathsf{Valid}_i$.
2. The parties engage in a VABA protocol, in which party P_i validates P_j when
   - P_i reliably receives S_j from P_j, and
   - $S_j \subseteq \mathsf{Valid}_i$.
3. When party P_i obtains the singleton set $\{j^*\}$ as output from the VABA protocol, it outputs S_{j^*}.

Using the reliable broadcast protocol from [DXR21], the communication complexity of this ACS protocol is $O(\kappa n^3)$ plus that of the VABA protocol. Alternatively, for $n \ll \kappa$, Bracha's reliable broadcast protocol [Bra87] gives a better communication complexity of $O(n^4)$. As already mentioned, the new ACS protocol presented in this paper is actually obtained via this reduction from a new VABA protocol.

3 Subprotocols

In this section we present the core subprotocols used in the new VABA protocol.

3.1 Gather with binding cover

In Section 2.5 we discussed the notion of a Gather protocol. The new VABA protocol will require a stronger type of Gather protocol that provides a certain **binding cover** property. Essentially, this property says that any index included in the output of any honest party must be that of a party that was already locally validated at the time the first honest party generated an output.

The security property is captured by the ideal functionality $\mathfrak{F}_{\mathrm{GatherCov}}$ in Fig. 5.

Consider again the example where we "drive" the protocol with an RBC protocol, where P_i *validates* P_j means that P_i has obtained an output message m_j from the RBC instance in which P_j is the sender, and that P_i has found that m_j satisfies a particular asynchronous validity predicate V. The basic Gather security property says that when any honest party P_i generates a Gather output X_i, we know that $X_i \supseteq X$, and that for each $j \in X_i$, party P_i has reliably received and verified a message from P_j. The binding cover property says that, in addition, for each $j \in X_i$, some honest party had already reliably received and verified a message from P_j at the time the first honest party generated a Gather output.

$\mathfrak{F}_{\text{GatherCov}}$

$\mathsf{Input}(\mathsf{validate}, j)$: this operation may be invoked by each party P_i, where $j \in [n]$. In response, $\mathfrak{F}_{\text{acs}}$ sends the message $\mathsf{NotifyInput}(\mathsf{validate}, i, j)$ to the ideal-world adversary.

$\mathsf{RequestOutput}(i, X_i, X)$: this operation may be invoked by the ideal-world adversary, one per $i \in [n]$. In response, $\mathfrak{F}_{\text{acs}}$ sends to P_i the message $\mathsf{Output}(X_i)$. If this is the first time this operation is invoked, define the "cover set" $Y \coloneqq \{j \in [n] : P_j \text{ has already been locally validated}\}$. The value X_i must be a subset of the "cover set" Y such that P_i has already validated P_j for all $j \in X_i$. The "core set" X must be a subset of X_i of size at least $n - t$. Moreover, all invocations of this operation must specify the same "core set" X.

Fig. 5. The Gather with binding cover ideal functionality

3.1.1 Cover-binding Gather from basic Gather. Here is a generic construction of a cover-binding Gather protocol from any basic (core-binding) Gather protocol. We make use of n instances of the one-sided voting protocol $\mathit{\Pi}_{\text{OneSidedVote}}$ from Section 2.3.1, call them $\text{Vote}_1, \ldots, \text{Vote}_n$.

Each party P_i maintains a boolean variable $\mathit{withdraw}_i$, initially set to false (but which will eventually be set to true), and a set Valid_i of party indices, initially empty. Whenever party P_i validates P_j, party P_i supplies a $\mathsf{support}$ input to Vote_j, *unless* $\mathit{withdraw}_i$. Whenever, P_i obtains an accept output from some Vote_j, it adds the index j to the set Valid_i.

All parties participate in an instance of the basic gather protocol, in which each party P_i validates P_j when $j \in \mathsf{Valid}_i$. Concurrently, when the size of Valid_i reaches $n - t$, party P_i sets $\mathit{withdraw}_i \leftarrow \mathsf{true}$ and broadcasts the message "withdrawn".

Each party P_i waits for

- the basic Gather protocol to complete, obtaining a set $X_i \subseteq [n]$, and
- to receive a "withdrawn" message from $n - t$ distinct parties.

Only then does P_i output the result X_i.

Let us call this protocol $\mathit{\Pi}_{\text{GenericGatherCov}}$. It is clear that $\mathit{\Pi}_{\text{GenericGatherCov}}$ has message and communication complexity $O(n^3)$ plus that of the underlying basic Gather protocol. As for security and liveness, we have the following:

Theorem 3.1. *If the underlying basic Gather protocol securely emulates $\mathfrak{F}_{\text{Gather}}$, then $\mathit{\Pi}_{\text{GenericGatherCov}}$ securely emulates $\mathfrak{F}_{\text{GatherCov}}$. If the underlying basic Gather protocol satisfies the Gather liveness property, then so does $\mathit{\Pi}_{\text{GenericGatherCov}}$.*

Proof. The main thing to check the following: any index included in the output of any honest party must be that of a party that was already locally validated at the time the first honest party generated an output. This follows from a standard "quorum intersection" argument. Consider the point in time where the first honest party produces an output. At this point in time, at least $n-t-t'$ honest parties must have already withdrawn their support in the one-sided voting protocols, where $t' \leq t$ is the number of corrupt parties. Thus, if P_j has not been locally validated as yet, it can only obtain support in Vote_j from at most $(n - t') - (n - t - t') = t$ honest parties, which is insufficient to make any honest part output accept in Vote_j. This means that j will never appear in Valid_i for any honest party P_i, and hence will never appear in the output set of any honest party.

To show that $\mathit{\Pi}_{\text{GenericGatherCov}}$ securely emulates $\mathfrak{F}_{\text{GatherCov}}$, the simulator in the ideal world just runs the one-sided voting protocols and the simulator for the basic gather protocol.

The proof of liveness is straightforward and is omitted. □

We note that we can add the binding cover property to the basic Gather protocol presented in Section 2.5.1 protocol in a way that slightly simplifies the generic construction given above. Just as above, we preface the protocol with one-sided voting with logic for "support withdrawal" (so that the sets Valid_i contain those indices j for which P_i has obtained output accept from Vote_j). In addition, a party withdraws support for voting as soon is finishes Step 1 of the protocol. We do not need to add the "withdrawn" messages as in Section 3.1.1 — the "ack" messages can play that role. However, we need to modify the logic for sending "ack" messages, adding to the list of preconditions for party P_i the requirement that P_i has already completed Step 1 (and, in particular, has withdrawn support in the one-sided voting protocol instances).

Let us call the resulting protocol $\Pi_{\text{BasicGatherCov}}$. It is clear that $\Pi_{\text{BasicGatherCov}}$ has message and communication complexity $O(n^3)$. The following is easily proved by combining the proofs of Theorems 2.1 and 3.1.

Theorem 3.2. *$\Pi_{\text{BasicGatherCov}}$ securely emulates $\mathfrak{F}_{\text{GatherCov}}$ and satisfies the Gather liveness property.*

3.2 Asynchronous Secret Key Sharing

We introduce a type of secret sharing protocol with security properties tailored very specifically to our needs. An **asynchronous secret key sharing (ASKS)** protocol works as follows. There is a **designated dealer** D and a **secret space** $\mathcal{S}$. The protocol has two stages, a **dealing phase** and a **reconstruction phase**.

In the dealing phase, the dealer shares a secret $s \in \mathcal{S}$ among all parties. If the dealer is honest, all honest parties will eventually terminate the dealing phase; moreover, s is random and the adversary (who controls all of the corrupt parties) learns nothing about s during this phase. If the dealer is corrupt, the dealing phase may or may not terminate, but if it does terminate for one honest party, then it will eventually terminate for all honest parties; moreover, the secret s may be an arbitrary element of $\mathcal{S}$, and the adversary must commit to s by the time the first honest party finishes the dealing phase.

After an honest party finishes the dealing phase, it may start the reconstruction phase. If all honest parties start the reconstruction phase, then all honest parties will eventually output the secret s from the dealing phase. If D is honest, the adversary only learns s when the first honest party starts the reconstruction phase.

More precisely, in addition to having uniformly bounded communication and computational complexity (see Section 2.2), an ASKS protocol should satisfy a *security* property and a *completeness* property. The security property is captured by the ideal functionality $\mathfrak{F}_{\text{ASKS}}$ in Fig. 6.

The completeness property says two things:

- If one honest party generates a $\mathsf{done\text{-}deal}$ output, or if D is honest and supplies a deal input, then every honest party *eventually* generates a $\mathsf{done\text{-}deal}$ output.
- If all honest parties input recon then every honest party *eventually* generates a $\mathsf{done\text{-}recon}$ output.

As usual, *eventually* means if and when all messages sent from one honest party to another have been delivered.

We shall give a secure ASKS protocol that is simple and efficient; however, its analysis relies on modeling a certain hash function as a random oracle. We can also prove that the same protocol

$\mathfrak{F}_{\text{ASKS}}$

Input(deal, $[s]$): this operation is invoked once by the dealer D. If D is corrupt, the optional argument $s \in \mathcal{S}$ must be supplied; otherwise, the optional argument should not be supplied and $\mathfrak{F}_{\text{ASKS}}$ chooses $s \in \mathcal{S}$ at random. In either case, we say *the secret has been input*. In response, $\mathfrak{F}_{\text{ASKS}}$ sends the message NotifyInput(deal) to the ideal-world adversary.

RequestOutput(done-deal, i): after *the secret has been input*, this operation may be invoked by the ideal-world adversary, who specifies $i \in [n]$. In response, $\mathfrak{F}_{\text{ASKS}}$ sends to P_i the message Output(done-deal).

Input(recon): This operation may be invoked once by each party P_i. If this is the first time this is invoked by any honest party, $\mathfrak{F}_{\text{ASKS}}$ sends NotifyInput(recon, i, s) to the ideal-world adversary, and we say *the secret has been revealed*; otherwise, $\mathfrak{F}_{\text{ASKS}}$ sends NotifyInput(recon, i) to the ideal-world adversary.

RequestOutput(done-recon, i): after *the secret has been revealed*, this operation may be invoked by the ideal-world adversary, who specifies $i \in [n]$. In response, $\mathfrak{F}_{\text{ASKS}}$ sends to P_i the message Output(done-recon, s).

Fig. 6. The ASKS ideal functionality (parameterized by D and $\mathcal{S}$)

satisfies a somewhat weaker notion of security without modeling the hash function as a random oracle, but rather, just under a specific (and reasonable) cryptographic assumption on the hash function (but which itself can be justified in the random oracle model). Moreover, this notion of security is sufficient for our applications in the paper, allowing us to prove all of our main results without relying on the random oracle model.

We call this weaker notion of security **ASKS with leakage**, and it is parameterized by a **secret generating function** G, which takes as input the set $\mathcal{C}$ of indices of corrupt parties and outputs a triple (s, d, r). Here, s is the secret as before, and d represents the information leaked during the dealing phase, and r is additional information leaked during the reconstruction phase.

The corresponding ideal functionality, denoted $\mathfrak{F}_{\text{ASKS}}[G]$, is given in Fig. 7 (changes are highlighted).

$\mathfrak{F}_{\text{ASKS}}[G]$

Input(deal, $[s]$): this operation is invoked once by the dealer D. If D is corrupt, the optional argument $s \in \mathcal{S}$ must be supplied, and $\mathfrak{F}_{\text{ASKS}}[G]$ sets $d \leftarrow r \leftarrow \bot$; otherwise, the optional argument should not be supplied and $\mathfrak{F}_{\text{ASKS}}[G]$ computes $(s, d, r) \xleftarrow{\$} G(\mathcal{C})$. In either case, we say *the secret has been input*. In response, $\mathfrak{F}_{\text{ASKS}}[G]$ sends the message NotifyInput(deal, d) to the ideal-world adversary.

RequestOutput(done-deal, i): after *the secret has been input*, this operation may be invoked by the ideal-world adversary, who specifies $i \in [n]$. In response, $\mathfrak{F}_{\text{ASKS}}[G]$ sends to P_i the message Output(done-deal).

Input(recon): This operation may be invoked once by each party P_i. If this is the first time this is invoked by any honest party, $\mathfrak{F}_{\text{ASKS}}[G]$ sends NotifyInput(recon, i, s, r) to the ideal-world adversary, and we say *the secret has been revealed*; otherwise, $\mathfrak{F}_{\text{ASKS}}[G]$ sends NotifyInput(recon, i) to the ideal-world adversary.

RequestOutput(done-recon, i): after *the secret has been revealed*, this operation may be invoked by the ideal-world adversary, who specifies $i \in [n]$. In response, $\mathfrak{F}_{\text{ASKS}}[G]$ sends to P_i the message Output(done-recon, s).

Fig. 7. The ASKS with leakage ideal functionality (parameterized by D, $\mathcal{S}$, and secret generating function G)

We call G **securely hiding** if no efficient adversary can effectively distinguish between the two distributions

$$\{(s,d) : (s,d,r) \xleftarrow{\$} G(\mathcal{C})\}$$

and

$$\{(s^*,d) : (s,d,r) \xleftarrow{\$} G(\mathcal{C}), s^* \xleftarrow{\$} \mathcal{S}\}.$$

While this property is not, strictly speaking, a part of the ASKS with leakage definition, it is needed in applications. Intuitively, if an ASKS protocol is a secure ASKS protocol with leakage defined by secret generating function G, and G is securely hiding, then the secret s generated by an honest dealer is indistinguishable from random until such time that it is revealed in the reconstruction stage.

It is easy to see that any secure ASKS protocol is also secure with leakage defined by the secret generating function $G(\mathcal{C}) \coloneqq (s, \perp, \perp)$, where $s \xleftarrow{\$} \mathcal{S}$.

3.2.1 A simple construction. We sketch a simple construction, which is closely related to one in [SS23] for a somewhat different purpose, namely, "secure key distribution".

In the dealing phase, the dealer D chooses a random polynomial $f \in \mathbb{Z}_q[x]$ of degree at most t. Here, q is a large prime. For $j \in [n]$, D computes $y_j = f(j) \in \mathbb{Z}_q$ and $h_j \leftarrow H(j, y_j) \in \mathcal{S}$. Here, $H : [0\,..\,n] \times \mathbb{Z}_q \rightarrow \mathcal{S}$ is a cryptographic hash function. Note that the secret to be shared is $s \coloneqq H(0, f(0))$. Next, the dealer uses an RBC protocol (see Section 2.3) to reliably broadcast the vector $(h_1, \ldots, h_n)$, and also sends party P_j the value y_j for each $j \in [n]$ (over a secure point-to-point channel).

After reliably receiving $(h_1, \ldots, h_n)$ via RBC, and after receiving the value $y_i \in \mathbb{Z}_q$ directly from D, party P_i checks if $h_i = H(i, y_i)$. If so, P_i inputs support to a one-sided voting protocol (see Section 2.3.1). In any case, P_i waits for the one-sided voting protocol to output accept (which may occur even if P_i did not provide support to the one-sided voting protocol) — if and when that happens, P_i outputs done-deal to indicate the it has completed the dealing phase.

In the reconstruction phase, each P_i that input support to the one-sided voting protocol in the dealing phase sends the value y_i to all parties. Each party P_i waits to reliably receive $(h_1, \ldots, h_n)$ as well as a collection $\{y_j\}_{j \in J}$ of values from $t+1$ distinct parties which satisfy $h_j = H(j, y_j)$ for all $j \in J$. When this happens, P_i interpolates through the points $\{(j, y_j)\}_{j \in J}$ to obtain a polynomial $f \in \mathbb{Z}_q[x]$ of degree at most t, and checks that $H(j, f(j)) = h_j$ for all $j \in [n]$. If so, P_i outputs $s \coloneqq H(0, f(0))$; otherwise, P_i outputs some default value. (Note that we could have the protocol instead output a special "error" value that effectively proves the dealer was corrupt.)

Let us call this protocol Π_{ASKS}. Assuming that H has output-length κ and that $\log_2 q = O(\kappa)$, the message complexity of Π_{ASKS} is $O(n^2)$ and the communication complexity of Π_{ASKS} is $O(\kappa n^2)$ — these bounds hold in both the dealing and reconstruction phases, but do not include the cost of the underlying RBC protocol. Assuming we implement RBC using the protocol from [DXR21], these same bounds hold for the protocol as a whole (assuming the hash functions used in this RBC protocol also have output-length κ).

As for security and completeness, we have the following:

Theorem 3.3. *If we model H as a random oracle, and if $1/q$ and $1/2^\kappa$ are negligible, and if the underlying RBC protocol securely emulates $\mathfrak{F}_{\text{RBC}}$, then Π_{ASKS} securely emulates $\mathfrak{F}_{\text{ASKS}}$. If the underlying RBC protocol satisfies the RBC completeness property, then Π_{ASKS} satisfies the ASKS completeness property.*

Proof. For security, we have to give a simulator. We work in a hybrid model, where in addition to modeling H as a random oracle, we also model the RBC subprotocol as an ideal functionality. We consider two cases.

The first case is where the dealer D is corrupt. In this case, although we are modeling H as a random oracle, the only property we really require of H is that it is hard for the adversary to find collisions in H, which is ensured by the assumption that $1/2^{\kappa}$. The simulator runs the logic of the honest parties. Consider the point in time where the first honest party is about to output done-deal. The goal of the simulator is to "extract" the dealer's secret $s \in \mathcal{S}$ so that it can input this secret to $\mathfrak{F}_{\text{ASKS}}$ as the deal input on behalf of D. Now, at this point in time, at least $n - 2t \geq t + 1$ honest parties must have input support to the one-sided voting protocol, which means that these $t + 1$ honest parties received shares consistent with the commitment vector $(h_1, \ldots, h_n)$. Therefore, the simulator can process these $t + 1$ shares exactly as in the reconstruction phase of the protocol to obtain $s \in \mathcal{S}$. The collision resistance of H guarantees that this is the secret that each honest party would eventually output in the reconstruction phase of the protocol, which ensures that the simulation is faithful to the real execution.

The second case is where the dealer D is honest. Here, the simulator needs to program the random oracle. The simulator just runs the protocol, choosing the polynomial f at random just as in the protocol. When the random secret s is revealed to the simulator by the ideal functionality, since $1/q$ is negligible, the adversary has queried H at $(0, f(0))$ with only negligible probability, and so the simulator may at that time "program" H so that $H(0, f(0)) \coloneqq s$. We also rely on the collision resistance of H here as well, to ensure that the shares supplied by corrupt parties in the reconstruction phase are correct.

That proves security. The proof of liveness is straightforward and is omitted. □

We can also analyze $\mathit{\Pi}_{\text{ASKS}}$ without relying on a random oracle, and instead show that it satisfies an appropriate notion of ASKS with leakage with respect to the secret generating function G defined as follows: $G(\mathcal{C}) \coloneqq (s, d, r)$, where

$$s \coloneqq H(0, f(0)),$$

$$d \coloneqq \big(\ \{f(j)\}_{j \in \mathcal{C}},\ \{H(j, f(j))\}_{j \in [n] \setminus \mathcal{C}}\ \big),$$

$$r \coloneqq \big(\ \{f(j)\}_{j \in [n] \setminus \mathcal{C}}\ \big),$$

and $f \in \mathbb{Z}_q[x]$ is a random polynomial of degree at most t.

Then we have:

Theorem 3.4. *If H is collision resistant, and if the underlying RBC protocol securely emulates $\mathfrak{F}_{\text{RBC}}$, then $\mathit{\Pi}_{\text{ASKS}}$ securely emulates $\mathfrak{F}_{\text{ASKS}}[G]$, where G is the secret generating function defined above.*

Proof. As in the proof of Theorem 3.3, we design a simulator that works in a hybrid model where the RBC subprotocol is modeled as an ideal functionality.

The case where the dealer is corrupt is handled identically as in that proof — we only relied on collision resistance in that case.

In the case where the dealer is honest, the secret generating function G was designed in such a way as to give the simulator precisely the information leaked during the dealing and reconstruction phases. We also rely on the collision resistance of H here as well, to ensure that the shares supplied by corrupt parties in the reconstruction phase are correct. □

For applications, we also want to show that the secret generating function G is securely hiding. This follows from an assumption that H satisfies a certain kind of indistinguishability assumption under a "related key attack" introduced in [SS23], called the **linear hiding assumption**.

This assumption is defined by a game in which the adversary first chooses a collection of pairs $\{(a_i, b_i)\}_{i\in\mathcal{I}}$, where $\mathcal{I} \subseteq [0\mathinner{.\,.} n]$ and $(a_i, b_i) \in \mathbb{Z}_q^* \times \mathbb{Z}_q$ for each $i \in \mathcal{I}$. The task of the adversary is to distinguish the distribution

$$\big\{H(i, a_i r + b_i)\big\}_{i\in\mathcal{I}},$$

where $r \in \mathbb{Z}_q$ is randomly chosen, from the uniform distribution on $\mathcal{S}^{\mathcal{I}}$. The assumption states that no computationally bounded adversary can effectively distinguish these two distributions.

The following theorem is easily proved following the arguments in [SS23]:

Theorem 3.5. *The secret generating function G defined above is securely hiding assuming H satisfies the linear hiding assumption.*

As discussed in [SS23], the linear hiding assumption can be justified by modeling H as a random oracle. However, it is a reasonable concrete assumption in its own right, and the analysis of the security properties of $\mathit{\Pi}_{\mathrm{ASKS}}$ under this concrete assumption is much more meaningful than modeling H as a (programmable) random oracle as in Theorem 3.3. In particular, the linear hiding assumption is a simple, natural, and *falsifiable* assumption (see [Nao03]).

3.2.2 Relation to other work. What we call ASKS is not to be confused with the notion of "weak VSS" that goes back to [RB89] and has been studied in a number of papers — see [KKK07] for a more modern definition and treatment. One difference between ASKS and "weak VSS" is that ASKS works in the asynchronous model while "weak VSS" works only in the synchronous model. A second difference is that in "weak AVSS", when the dealer is corrupt, some honest parties may output a special error symbol $\perp$, while others may output the common secret $s \in \mathcal{S}$. A third difference is that in ASKS, for an honest dealer, the secret is chosen at random by the protocol, and is not an input to the protocol.

Confusingly, in [DW20], a notion of "asynchronous weak VSS" is defined; however, in their definition, when the dealer is corrupt, all honest parties must output the *same* value from the set $\mathcal{S} \cup \{\perp\}$. This is really not much different from our notion of ASKS, as the protocol can always just replace $\perp$ by some default value in $\mathcal{S}$. We also note that in [DW20], no notion of preserving the secrecy of an honest dealer's secret is ever discussed (in any definitions or proofs). A construction that has similarities to that presented here is given in [DW20].

In a follow-up paper [DGNW20], a very informal definition of such a secrecy property is given but the security proof of the scheme (the same one as in [DW20]) does not mention this secrecy property at all (it does not suggest how or under what assumptions it might be proved). The paper [BBB$^+$23] gives somewhat different scheme (based loosely on a construction in [BKP11]) and claims that it provides some sort of *statistical* hiding property of the secret based on a corresponding statistical hiding assumption on the hash function. While this assumption on the hash function may be reasonable, the reasoning here is fundamentally flawed. The problem is that the adversary sees the output of many hashes with correlated inputs (as in our linear hiding assumption above). There is simply not enough entropy in these inputs to justify any statistical hiding property.

None of the papers [DW20,DGNW20,BBB$^+$23] give a satisfying definition of security, let alone anything like a rigorous proof of security. We mention again that our construction and its analysis is based on ideas from [SS23], although the context there was a bit different.

4 The new VABA protocol

4.1 Description of the protocol

The new protocol Π_{VABA} makes use of:

- an RBC protocol (see Section 2.3);
- a cover-binding Gather protocol (see Section 3.1);
- an ASKS protocol with a secret space $\mathcal{S}$ (see Section 3.2) with leakage corresponding to a securely hiding secret generating function G.

In addition to these protocols, we need a specialized cryptographic function called a **rank derivation function**. This is a function $\mathrm{Rank}(\cdot, \cdot)$ that takes two inputs:

- an index $j \in [n]$,
- a collection of values $\{s_k\}_{k \in K}$, indexed by some set $K \subseteq [n]$, with $s_k \in \mathcal{S}$ for all $k \in K$,

and outputs a value in a finite set $\mathcal{R}$. We write such an invocation of Rank as

$$r \leftarrow \mathrm{Rank}(j, \{s_k\}_{k \in K}).$$

In addition, we require that the output space $\mathcal{R}$ is large (i.e., $1/|\mathcal{R}|$ is negligible), and should be equipped with a linear order (for example, if $\mathcal{R}$ is the set of all bit strings of some fixed length, we can order elements of $\mathcal{R}$ according to the value of the integers they represent in binary). We discuss below in Section 4.3 the precise security property a rank derivation function must satisfy, as well as a simple construction.

Recall that in a VABA protocol, each party P_i validates other parties P_j incrementally over time. We let Valid_i denote the set of indices of parties that P_i has validated (this set grows over time).

Protocol Π_{VABA} proceeds in rounds (or "views") $v = 1, 2, \ldots$. In each round v, each party P_i will "vote" for a one party — we let $vote_i^{(v)}$ denote the index of the party that P_i votes for in round v. A party P_i will only vote for a party that it has already validated, that is, we will have $vote_i^{(v)} \in \mathsf{Valid}_i$ at the time P_i chooses $vote_i^{(v)}$. In round $v = 1$, P_i is free to choose $vote_i^{(v)}$ any way it wishes, subject to this constraint. In some applications, we can guarantee that $i \in \mathsf{Valid}_i$ at the time P_i chooses its initial vote (for example, if the ACS protocol is "driven" by a reliable broadcast or verifiable secret sharing protocol), in which case P_i can choose its own index i as its initial vote. While we do not require this in general, when it does hold, it can provide certain additional benefits (see Section 4.5).

The logic for round v is as follows:

1. Start n instances $\mathrm{ASKS}_1^{(v)}, \ldots, \mathrm{ASKS}_n^{(v)}$ of the ASKS protocol, where in each $\mathrm{ASKS}_i^{(v)}$ party P_i acts as dealer to share a secret $s_i^{(v)} \in \mathcal{S}$.
   We denote by $\mathsf{DD}_i^{(v)}$ the set of "done dealings" for P_i, that is, the set of indices j for which P_i has finished the dealing phase (i.e., has output $\mathsf{done\text{-}deal}$) of $\mathrm{ASKS}_j^{(v)}$ (this set grows over time).
2. Each party P_i waits for $|\mathsf{DD}_i^{(v)}| \geq t{+}1$ and sets $PD_i^{(v)} \leftarrow \mathsf{DD}_i^{(v)}$ (P_i's set of "proposed dealings").
3. Start n instances $\mathrm{vRBC}_1^{(v)}, \ldots, \mathrm{vRBC}_n^{(v)}$ of the RBC protocol, where in each $\mathrm{vRBC}_i^{(v)}$ party P_i reliably broadcasts $(vote_i^{(v)}, PD_i^{(v)})$.

4. Start an instance of the cover-binding Gather protocol, in which each party P_i validates P_j when all of the following hold:
   (a) P_i has reliably received an output $(vote_j^{(v)}, PD_j^{(v)})$ from P_j via $\text{vRBC}_j^{(v)}$;
   (b) $vote_j^{(v)} \in \mathsf{Valid}_i$,
   (c) $|PD_j^{(v)}| \geq t+1$ and $PD_j^{(v)} \subseteq \mathsf{DD}_i^{(v)}$;
   (d) if $v > 1$, then it must also be the case that $vote_j^{(v)}$ is *justified*, meaning that it is a mode of
   $$\{\, prevote_k^{(v-1)} \,\}_{k \in PV}$$
   for some subset $PV \subseteq \mathsf{RPV}_i^{(v-1)}$ of size at least $n-t$. As usual, a mode of a finite collection of values is a value that appears at least as often as any other. Note that the values $\mathsf{RPV}_i^{(v-1)} \subseteq [n]$ and $prevote_k^{(v-1)} \in [n]$ are defined in Steps 7–8 of the previous round (see below). We give an algorithm below in Section 4.2 for determining if a given vote is justified.

   We let $\mathsf{GatherValid}_i^{(v)}$ denote the set of indices of parties that P_i has validated (as above) for this instance of the Gather subprotocol (this set grows over time).
5. Each P_i waits for the Gather protocol to complete, obtaining a set $X_i^{(v)} \subseteq [n]$, and when that happens, P_i initiates the reconstruction phase of all ASKS protocol instances $\text{ASKS}_j^{(v)}$ with $j \in \mathsf{DD}_i^{(v)}$ (and will continue to do so for other ASKS protocol instances $\text{ASKS}_j^{(v)}$ as the set $\mathsf{DD}_i^{(v)}$ continues to grow).
6. Each party P_i waits to obtain the secret $s_k^{(v)}$ for all
   $$k \in \bigcup_{j \in X_i^{(v)}} PD_j^{(v)},$$
   and then for all $j \in X_i^{(v)}$ computes
   $$rank_j^{(v)} \leftarrow \text{Rank}\left(\, j,\ \{s_k^{(v)}\}_{k \in PD_j^{(v)}} \,\right)$$
   and computes $prevote_i^{(v)}$ to be $vote_\ell^{(v)}$ for an index $\ell \in X_i^{(v)}$ that maximizes $rank_\ell^{(v)}$.
7. Start n instances $\text{pvRBC}_1^{(v)}, \ldots, \text{pvRBC}_n^{(v)}$ of the RBC protocol, where in each $\text{pvRBC}_i^{(v)}$ party P_i reliably broadcasts $prevote_i^{(v)}$.

   We denote by $\mathsf{RPV}_i^{(v)}$ the set of "received prevotes" for P_i, which is defined as the set of indices j for which P_i has reliably received a value $prevote_j^{(v)} \in \{\, vote_k^{(v)} : k \in \mathsf{GatherValid}_i^{(v)} \,\}$ from P_j via $\text{pvRBC}_j^{(v)}$ (this set grows over time).
8. Each party P_i waits for $|\mathsf{RPV}_i^{(v)}| \geq n-t$ and sets $PV_i^{(v)} \leftarrow \mathsf{RPV}_i^{(v)}$ and sets $vote_i^{(v+1)}$ to a mode of
   $$\{\, prevote_j^{(v)} \,\}_{j \in PV_i^{(v)}}. \tag{1}$$
   In addition, if all prevotes appearing in (1) are the same value $prevote$, P_i outputs its decision $\{prevote\}$ for the VABA protocol, and participates in just one more round of the protocol.

Adding a termination gadget. That completes our description of the protocol. Note that the protocol as given in [DDL+24] includes a simple device known as a "termination gadget". This is an extra step that can be added to any consensus protocol (like VABA or ACS), and that allows parties that have decided to eventually safely delete all state information associated with the protocol instance (except insofar that they need to maintain enough state information that allows them to broadcast one final message with an "eventual delivery" guarantee).

4.2 Determining which votes are justified

We give here a simple and efficient algorithm that allows party P_i to determine which votes are justified in round $v > 1$, as required in Step 4(d) of Π_{VABA}. A vote is justified if it is a mode of

$$\{\, \mathit{prevote}_k^{(v-1)} \,\}_{k \in PV}$$

for some subset $PV \subseteq \mathsf{RPV}_i^{(v-1)}$ of size at least $n - t$. We use two arrays:

- an array F indexed by $x = 1 \mathinner{..} n$, where $F[x]$ counts the number of indices $j \in \mathsf{RPV}_i^{(v-1)}$ such that $\mathit{prevote}_j^{(v-1)} = x$;
- an array S indexed by $c = 0 \mathinner{..} n$, where
$$S[c] = \sum_{x=1}^{n} \min(c, F[x]).$$

A vote w is justified iff $S[F[w]] \geq n - t$. Party P_i initializes the arrays F and S to zero. Whenever an index j is added to $\mathsf{RPV}_i^{(v-1)}$, if $\mathit{prevote}_j^{(v-1)} = x$, party P_i increments $F[x]$ and then also increments $S[c]$ for $c = F[x] \mathinner{..} n$.

Thus, whenever party P_i adds an index to $\mathsf{RPV}_i^{(v-1)}$, it needs to perform just $O(n)$ work to update the set of justified votes for round v. We note that the mechansim used here for justifying votes is a bit different than the mechanism used in [DDL+24], which requires additional information to be reliably broadcast in Step 3.

4.3 Rank derivation functions: a security property and a construction

Recall the syntax of a rank derivation function as given above. We state here the security property we need for such a function in order to analyze protocol Π_{VABA}. This property is defined by a game in which the adversary first chooses

- a disjoint decomposition $(\mathcal{C}, \mathcal{H})$ of the set $[n]$,
- values $s_k \in \mathcal{S}$ for each $k \in \mathcal{C}$,
- a set $Y \subseteq [n]$,
- a collection of sets $\{K_j\}_{j \in Y}$, where $K_j \subseteq [n]$ and $K_j \cap \mathcal{H} \neq \emptyset$ for each $j \in Y$.

The task of the adversary is to distinguish between the distribution

$$\{\ \mathrm{Rank}(j, \{s_k\}_{k \in K_j})\ \}_{j \in Y}, \tag{2}$$

where $s_k \in \mathcal{S}$ is chosen at random for each $k \in \mathcal{H}$, and the uniform distribution on $\mathcal{R}^Y$. The security property states that no computationally bounded adversary can effectively distinguish these two distributions. If this property holds we say that Rank is a **pseudorandom rank derivation function**.

4.3.1 Combining rank derivation with a secret generating function.

In our application, the secrets s_k for $k \in \mathcal{H}$ in the game defining the pseudorandomness property for Rank are themselves pseudorandom. Specifically, each such s_k is generated as $(s_k, d_k, r_k) \xleftarrow{\$} G(\mathcal{C})$, where G is a secret generating function as in Section 3.2.

To model this, we can modify the game defining the pseudorandomness property for Rank as follows. First, the adversary chooses the disjoint decomposition $(\mathcal{C}, \mathcal{H})$ and gives this to the challenger. In response, the challenger computes $(s_k, d_k, r_k) \xleftarrow{\$} G(\mathcal{C})$ for each $k \in \mathcal{H}$ and sends $\{d_k\}_{k \in \mathcal{H}}$ to the adversary. The rest of the game proceeds as before, except that the challenger uses the values s_k for $k \in \mathcal{H}$ as computed above to compute the ranks (2). It is important to give the adversary the values $\{d_k\}_{k \in \mathcal{H}}$ before it chooses the values $\{s_k\}_{k \in \mathcal{C}}$, Y, and $\{K_j\}_{j \in Y}$, in order to model the situation where the adversary's choice of the latter values may depend on the former.

We call an adversary playing this game a **rank distinguishing adversary**. We say that it is **hard to distinguish real ranks from random ranks** if every efficient rank distinguishing adversary has a negligible advantage in distinguishing real ranks from random ranks in this game. It is straightforward to argue (using a standard hybrid argument) that it is hard to distinguish real ranks from random ranks assuming G is securely hiding and Rank is pseudorandom.

4.3.2 Using the rank derivation function.

Roughly speaking, the way we will use the fact that it is hard to distinguish real ranks from random ranks as follows. Consider the point in time in a given round v of protocol $\mathit{\Pi}_{\text{VABA}}$ that the first honest party finishes the Gather subprotocol. By the *binding core* property of the subprotocol, there is a core set $X^{(v)}$ that is determined at this time such that $|X^{(v)}| \geq n - t$ and every honest party P_i will output $X_i^{(v)} \supseteq X^{(v)}$. Moreover, by the *binding cover* property of the subprotocol, if we define

$$Y^{(v)} := \bigcup_{i \in \mathcal{H}} \mathsf{GatherValid}_i,$$

where we use the values of $\mathsf{GatherValid}_i$ at this point in time, then every honest party P_i will output $X_i^{(v)} \subseteq Y^{(v)}$. Thus, the only ranks computed by the honest parties are $\mathit{rank}_j^{(v)}$ for $j \in Y^{(v)}$. This means that the sets $\mathit{PD}_j^{(v)}$ that will used by the honest parties in computing these ranks have already been determined, as have the corresponding secrets s_k for $k \in \mathit{PD}_j^{(v)}$ contributed by any corrupt parties P_k. Using the fact that it is hard to distinguish real ranks from random ranks, as discussed in Section 4.3.1, all of the ranks $\mathit{rank}_j^{(v)}$ for $j \in Y^{(v)}$ are "effectively as good as random". More precisely, in the distinguishing game described in Section 4.3.1, $\mathcal{C}$ is the set of corrupt parties, $\mathcal{H}$ is the set of honest parties, and the rank distinguishing adversary sets

$$Y := Y^{(v)} \quad \text{and} \quad K_j := \mathit{PD}_j^{(v)} \text{ for } j \in Y.$$

As we will see, the key to proving that $\mathit{\Pi}_{\text{VABA}}$ reaches consensus quickly is based on the simple fact:

Lemma 4.1. *Assume that the ranks $\mathit{rank}_j^{(v)}$ for $j \in Y^{(v)}$ are random elements of $\mathcal{R}$, and that $|\mathcal{R}| \geq \frac{3}{2}n^3$. Let γ be the probability that there is a unique maximum rank and that its index lies in the core set $X^{(v)}$. Then $\gamma \geq 2/3$.*

Proof. Let us write X in place of $X^{(v)}$ and Y in place of $Y^{(v)}$. We may assume that Y is as large as possible, so $Y = [n]$. and that X is as small as possible, so $|X| = n - t$. Let us assume that $\mathcal{R} = \{0, \dots, B-1\}$. So we view the ranks as a collection $\{r_j\}_{j=0}^{n-1}$ of random numbers $r_j \in \mathcal{R}$.

We claim that for any fixed $i \in [n]$, the probability of the event M_i that $r_i > r_j$ for all $j \neq i$ is at most $1/n$. Indeed, we have

$$\begin{aligned}
\Pr[M_i] &= \sum_{r \in \mathcal{R}} \Pr[r_i = r] \Pr[M_i \mid r_i = r] \\
&= \frac{1}{B} \sum_{r \in \mathcal{R}} (r/B)^{n-1} = \frac{1}{B^n} \sum_{r \in \mathcal{R}} r^{n-1} \\
&\leq \frac{1}{B^n} \int_0^B z^{n-1} dz \quad \text{(estimating the sum by an integral)} \\
&= \frac{1}{B^n} \cdot \frac{B^n}{n} = \frac{1}{n}.
\end{aligned}$$

Let C be the event that there is a collision, that is, $r_i = r_j$ for some $i \neq j$. Then by the union bound we have

$$\begin{aligned}
1 - \gamma &\leq \Pr[C] + \sum_{i \in [n] \setminus X} \Pr[M_i] \\
&\leq \frac{n^2}{2B} + \frac{t}{n} \\
&\leq 1/3 \quad (\text{since } n \geq 3t + 1 \text{ and } B \geq \tfrac{3}{2} n^3).
\end{aligned}$$

□

4.3.3 A simple construction based on a PRF. Let $F : \mathcal{S} \times [n] \to \mathcal{R}$ be a pseudo-random function (PRF), where the first argument is the key input and the second argument is the data input. We also assume that $\mathcal{R}$ is a finite abelian group, for which we use additive notation (for example, the set of all bit strings of a given length, so that addition is bit-wise exclusive-or). For a given $j \in [n]$ and collection of keys $\{s_k\}_{k \in K}$, where $K \subseteq [n]$ and $s_k \in \mathcal{S}$ for each $k \in K$, we define

$$\mathrm{Rank}(j, \{s_k\}_{k \in K}) := \sum_{k \in K} F(s_k, j).$$

Theorem 4.1. *If F is a secure PRF, then* Rank *as defined above is a pseudorandom rank derivation function.*

The proof is an easy exercise that we leave to the reader.

Other constructions. One might also consider the construction:

$$\mathrm{Rank}(j, \{s_k\}_{k \in K}) := F\big(\sum_{k \in K} s_k,\ j \ \big).$$

This assumes that $\mathcal{S}$ forms an abelian group. While this construction may well be secure, it cannot be proved under the assumption that F is a PRF. One would need to assume that F is a PRF under a "related key attack". It can also be proven secure if we model F as a random oracle.

4.4 Analysis of the VABA protocol

We now carry out an analysis of the security and liveness properties of protocol $\mathit{\Pi}_{\mathrm{VABA}}$. To prove certain properties, it suffices to consider a *hybrid* version of $\mathit{\Pi}_{\mathrm{VABA}}$, in which all of the RBC, Gather, and ASKS subprotocol instances are replaced by corresponding ideal functionalities.

The key property is the following:

Theorem 4.2. *Consider a finite run of the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$ in which some honest party P completes round $v+1$ but did not decide as of round v. Further suppose that in round v there is a value prevote, such that no more than t protocol instances $\mathrm{pvRBC}_k^{(v)}$ (where each corresponding sender P_k may be corrupt or honest) receive inputs other than prevote. Then P decides the value prevote in round $v+1$.*

Proof. We first argue that any justified vote in round $v+1$ must be a vote for the value *prevote*. This is a simple counting argument. A vote $\mathit{vote}_j^{(v+1)}$ in round $v+1$ must be justified by the existence of a set $PV \subseteq [n]$ of size at least $n-t$ such that for each $k \in PV$, a prevote $\mathit{prevote}_k^{(v)}$ was input to $\mathrm{pvRBC}_k^{(v)}$ and $\mathit{vote}_j^{(v+1)}$ is a mode of the collection of values

$$\{\, \mathit{prevote}_k^{(v)} \,\}_{k \in PV}.$$

Since at most t of the values $\mathit{prevote}_k^{(v)}$ are different from *prevote*, it follows that the mode of this collection must be *prevote*.

Since the only justified vote in round $v+1$ is a vote for *prevote*, and since any valid prevote must correspond to some justified vote, it follows that P must decide the value *prevote* in round $v+1$. □

We can use Theorem 4.2 to establish the security and liveness properties of $\mathit{\Pi}_{\mathrm{VABA}}$, which follow from the following theorem, assuming all subprotocols securely emulate their corresponding ideal functionalities and satisfy their corresponding liveness/completeness properties.

Theorem 4.3. *Consider a finite run of the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$ in which at least one honest party decides, and let v be the lowest-numbered round in which that occurrs. Then all honest parties that decide a value must decide the same value and do so by round $v+1$ at the latest.*

Proof. Suppose some party P_i decides a value in round v. By definition, P_i sees $n-t$ prevotes in that round for this value. Clearly, any other honest party P_j that also decides in the same round must also decide the same value. By Theorem 4.2, any other honest parties will decide the same value by round $v+1$. □

To use this theorem to prove liveness, we need to also prove the following: if all honest parties start a given round, they will eventually all finish that round. This is straightforward and is left to the reader. Note that liveness only says that all honest parties must decide if and when all messages sent from one honest party to another have been delivered and all parties that have been locally validated have also been globally validated. This notion of liveness is meaningless unless we also establish uniformly bounded message complexity. To do that, we want to show that the protocol reaches consensus quickly with overwhelming probability.

We can also use Theorem 4.2 to establish that the protocol reaches consensus quickly, using the reasoning in Section 4.3.2. Consider a run of the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$ and a round v in which some honest party has completed the Gather subprotocol, and let $X^{(v)}$ be the corresponding core set of size at least $n - t$. Define

$$Y^{(v)} := \bigcup_{i \in \mathcal{H}} \mathsf{GatherValid}_i,$$

where we use the values of $\mathsf{GatherValid}_i$ at the point in time at which the first honest party completed the Gather subprotocol in round v. At this same point in time, the rank values $\mathit{rank}_j^{(v)}$ for $j \in Y^{(v)}$ are completely determined, even though their values are not yet known to the adversary and are in fact computationally indistinguishable from random elements of $\mathcal{R}$ — here we use the hiding property of the secret generating function G associated with the ASKS protocol, and the pseudorandomness property of the rank derivation function. We say the adversary "loses the rank ordering game in round v" if these pseudorandom rank values are distinct and the index of the maximum rank value lies in the core set $X^{(v)}$. Otherwise, we say the adversary "wins the rank ordering game in round v". If these rank values were truly random, Lemma 4.1 says that the adversary loses the rank ordering game with probability at least $2/3$. Moreover, when this happens all honest parties will cast a prevote in round v for $\mathit{vote}_\ell^{(v)}$. When that happens, then Theorem 4.2 implies that all honest parties will decide by round $v + 1$.

Although it is rather tedious, one can turn the above intuitive argument into a rigorously provable statement on the rate at which $\mathit{\Pi}_{\mathrm{VABA}}$ reaches consensus. For a given adversary A, consider a complete run of protocol $\mathit{\Pi}_{\mathrm{VABA}}$ as driven by A — we assume that A drives the protocol for at most N rounds, where N is bounded by a polynomial $p_A(\lambda)$, which depends on A, in the security parameter λ. Now, A may halt the protocol at any step, and in particular, it may halt before any honest party decides. We denote by R_A the random variable that represents be the largest number of rounds that any one party finishes during this run without making a decision.

Theorem 4.4. *Assume that $|\mathcal{R}| \geq \frac{3}{2}n^3$, all subprotocols securely emulate their corresponding ideal functionalities, the secret generating function is securely hiding, and the rank derivation function is pseudorandom. Then we have:*

(i) For every adversary A, there exists a negligible function $\epsilon_A(\lambda)$ such that

$$E[R_A] \leq 3/2 + \epsilon_A(\lambda).$$

(ii) For every adversary A and every function $L = L(\lambda)$ that is polynomially bounded in λ and efficiently computable, there exists a negligible function $\epsilon'_A(\lambda)$ such that

$$\Pr[R_A \geq L + 1] \leq 3^{-L} + \epsilon'_A(\lambda).$$

(iii) For every adversary A, there exists a negligible function $\epsilon''_A(\lambda)$ such that and for all $v \geq 1$

$$\Pr[R_A \geq v + 1] \leq 3^{-v} + \epsilon''_A(\lambda).$$

Proof. The assumption that the secret generating function is securely hiding and the rank derivation function is pseudorandom imply that it is hard to distinguish real ranks from random ranks (as in Section 4.3.1) and the assumption that $|\mathcal{R}| \geq \frac{3}{2}n^3$ allows us to use Lemma 4.1.

Note that (iii) implies both (i) and (ii). However, the proofs and (i) and (ii) are much more straightforward with much tighter reductions, yielding functions $\epsilon_A(\lambda)$ and $\epsilon'_A(\lambda)$ that are much more realistic.

So we begin with a proof of (i). To that end, we first consider the same game defining R_A but with respect to the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$. Let us denote the corresponding random variable S_A.

Let us define a related random variable W_A as follows. Suppose that in the execution of the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$, for each round $v = 1, 2, \ldots$, we consider the first point in time where an honest party finishes the Gather subprotocol. We say that the adversary plays the rank ordering game in round v, and we can determine whether or not the adversary wins the rank ordering game in round v by looking at the relevant ranks, which are already determined by this time. We define W_A to be the number of successive rounds in which in the adversary plays and wins the rank ordering game. Then we have $S_A \leq W_A + 1$.

For each round number v, we define $\alpha_A^{(v)}$ to be the probability that the adversary plays and wins the rank ordering game in rounds $1, \ldots, v$, which is the same as the probability that $W_A \geq v$. So by the "tail sum formula" for expectation, we have

$$E[W_A] = \sum_{v \geq 1} \Pr[W_A \geq v] = \sum_{v=1}^{N} \alpha_A^{(v)}.$$

We then define $\beta_A^{(v)}$ to be the probability that the adversary plays and wins the rank ordering game in rounds $1, \ldots, v$, except that in round v, the rank ordering game is played (if it is played at all) with random ranks instead of real ranks. Note that by Lemma 4.1, we have

$$\beta_A^{(1)} \leq 1/3 \quad \text{and} \quad \beta_A^{(v)} \leq \alpha_A^{(v-1)}/3 \quad \text{for } v = 2, \ldots, N. \tag{3}$$

To prove (i), we design a rank distinguishing adversary B as follows. B chooses $v \in [N]$ at random and outputs 1 iff the adversary plays and wins the rank ordering game in rounds $1, \ldots, v$, where for round v, we use ranks supplied by a challenger in the rank distinguishing game. It is easily seen that B has a distinguishing advantage of

$$\eta_A(\lambda) := \Big| \frac{1}{N} \sum_{v=1}^{N} \alpha_A^{(v)} - \frac{1}{N} \sum_{v=1}^{N} \beta_A^{(v)} \Big|,$$

which, by assumption, is negligible. Therefore, by (3), we have

$$\begin{aligned} \sum_{v=1}^{N} \alpha_A^{(v)} &\leq N \cdot \eta_A(\lambda) + \sum_{v=1}^{N} \beta_A^{(v)} \\ &\leq N \cdot \eta_A(\lambda) + \frac{1}{3} + \frac{1}{3} \sum_{v=1}^{N} \alpha_A^{(v)}, \end{aligned}$$

and so

$$\sum_{v=1}^{N} \alpha_A^{(v)} \leq \frac{1}{2} + \frac{3N}{2} \eta_A(\lambda).$$

That shows that the expected value of W_A is at most 1/2 plus negligible, and since $S_A \leq W_A + 1$, the expected value of S_A is at most 3/2 plus negligible.

That gives the result we want for the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$. To get the result for $\mathit{\Pi}_{\mathrm{VABA}}$ itself, we note that the distributions of R_A and S_A are computationally indistinguishable, and the result follows from the general fact that two random variables that take values in $\{0,\dots,N\}$ for polynomially bounded N have expectations that differ by a negligible amount (this can be proven by designing a distinguishing adversary based on the "tail sum formula" for expectation that distinguishes between the real and ideal subprotocols). That proves (i).

Now consider (ii). For $v = 1,\dots,N$, define $\delta_A^{(v)} := \alpha_A^{(v)} - \beta_A^{(v)}$. By (3), we have

$$\alpha_A^{(v)} \le (1/3)^v + \overbrace{(1/3)^{v-1}\delta_A^{(1)} + \cdots + (1/3)\delta_A^{(v-1)} + \delta_A^{(v)}}^{\omega_A^{(v)} :=}. \tag{4}$$

So to prove (ii), it suffices to show that $\omega_A^{(L)}$ is negligible. We stress that here, L is a fixed, polynomially bounded function in λ. To do this, we design a rank distinguishing adversary B'. Just like B, adversary B' embeds a given challenge instance of the rank distinguishing game into a round, but instead of choosing a round number uniformly at random, B' chooses round number $L - r + 1$, where r is chosen from $\{1,\dots,L+1\}$ according to a truncated geometric distribution with success parameter $2/3$ — that is, for $r = 1,\dots,L$, it chooses r with probability $(1/3)^{r-1}(2/3)$, and chooses $r = L + 1$ with probability $(1/3)^L$. If round 0 (that is, $r = L+1$) is chosen, B' just outputs 0. It is easily seen that B' has a distinguishing advantage of $\eta'_A(\lambda) := |\frac{2}{3}\omega_A^{(L)}|$. By assumption, $\eta'_A(\lambda)$ is negligible, which proves (ii). We are assuming here that B' can perfectly sample r from the truncated geometric distribution; however, we can implement B' so that it samples r from a distribution that is statistically close to the truncated geometric distribution.

Now to prove (iii). Consider the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$. We want to show that there exists a negligible function $\zeta''_A(\lambda)$, which depends on A, such that

$$\Pr[W_A \ge v] \le 3^{-v} + \zeta''_A(\lambda).$$

for all $v \ge 1$. We stress that $\zeta''_A(\lambda)$ depends only on A and not on v.

To this end, note that

$$|\omega_A^{(v)}| \le \tfrac{3}{2}\mathit{\Delta}_A,$$

where $\omega_A^{(v)}$ is defined as in (4) and $\mathit{\Delta}_A := \max_{v=1}^{N}|\delta_A^{(v)}|$. It suffices to show that $\mathit{\Delta}_A$ is negligible. To do this, we can proceed by contradiction. Suppose that $\mathit{\Delta}_A$ is not negligible. This means that there exists an adversary A and a polynomial $q_A(\lambda)$ such that for infinitely many settings of the security parameter λ, we have $\mathit{\Delta}_A \ge 1/q_A(\lambda)$. So this means that for infinitely many settings of the security parameter λ, for some $v = 1,\dots,N$, we have $|\delta_A^{(v)}| \ge 1/q_A(\lambda)$. So we can design a rank distinguishing adversary B'' that for each such λ embeds a challenge for the rank distinguishing game at this round v. It is tempting to think we can just "hardwire" v into B''. However, if we restrict ourselves to a uniform computation model (i.e., programs, not circuits), this is not possible, unless N is a constant. Instead, we can — using an enormous (though still polynomially bounded) amount of sampling and a Chernoff bound — determine with high probability a single best round number v to use in B''.

In slightly more detail, we design a distinguishing adversary B'' that plays the rank distinguishing game as follows. First, B'' will estimate $\alpha_A^{(v)}$ and $\beta_A^{(v)}$ for all $v = 1,\dots,N$ by sampling from each corresponding distribution a polynomial number of times (depending on $q_A(\lambda)$). Using a Chernoff bound, this can be done so that B'' will find a v such that $|\delta_A^{(v)}| \geq 1/(2q_A(\lambda))$ with all but negligible probability. B'' will then use this v to embed a challenge for the rank distinguishing game. The running time of B'' is bounded by a polynomial in λ (which depends on $q_A(\lambda)$ and $p_A(\lambda)$). The distinguishing advantage of B'' is negligibly close to $1/(2q_A(\lambda))$ (for all sufficiently large settings of the security parameter λ for which $\Delta_A \geq 1/q_A(\lambda)$).

Note that the reduction here is not entirely "black box", as B'' will have to explicitly make use of the polynomials $q_A(\lambda)$ and $p_A(\lambda)$.

We also note that a similar sampling technique needs to be used to move from the hybrid version of $\mathit{\Pi}_{\mathrm{VABA}}$ to $\mathit{\Pi}_{\mathrm{VABA}}$ itself. □

As for complexity bounds, we see that in each round v of the protocol, the message complexity is $O(n^3)$, the communication complexity is $O(\kappa n^3)$, and the round complexity is $O(1)$. By part (i) of Theorem 4.4, in expectation, these same bounds hold for the protocol as a whole. We may also apply part (ii) of the theorem with, say, $L(\lambda) := \lambda$ to establish that the communication complexity is uniformly bounded (as in Section 2.2).

4.5 A quality property

Our VABA protocol enjoys a certain quality property. As we mentioned, in some applications, P_i can choose its own index i as its initial vote. In this case, it is easy to see with probability at least $\approx 1/3$, the output of $\mathit{\Pi}_{\mathrm{VABA}}$ will be the index of an honest party. Indeed, this will happen if in round $v = 1$, the maximum rank value lies in the set $X^{(v)} \cap \mathcal{H}$, which has size at least $n - 2t$.

4.6 Using a random beacon

We sketch here an alternative version of $\mathit{\Pi}_{\mathrm{VABA}}$ that replaces the ASKS/rank-derivation mechanism (see Section 4.3) for assigning ranks by a "random beacon". A random beacon is a standard tool used in the design of many protocols related to asynchronous consensus [CKS00,CKPS01]. In each round v, we shall require a random beacon that reveals a vector of ranks $(r_1^{(v)}, \ldots, r_n^{(v)}) \in \mathcal{R}^n$, with the following properties:

- this vector will eventually be revealed to all honest parties if all honest parties initiate the reveal protocol for the random beacon,
- this vector is computationally indistinguishable from a random vector of ranks at any point in time before at least one honest party has initiated the reveal protocol.

Such a random beacon may be implemented using a random beacon that outputs a short seed which is then stretched using a pseudorandom generator.

Using such a beacon, we may modify $\mathit{\Pi}_{\mathrm{VABA}}$ as follows:

- Steps 1 and 2 are eliminated, and in Step 3 each party P_i reliably broadcasts just the value $\mathit{vote}_i^{(v)}$.
- In Step 4, we may use a Gather protocol that does not have the cover-binding property (but it must still be core binding), and the values $\mathit{PD}_j^{(v)}$ are not used (and, in particular, the condition in Step 4(c) is dropped).
- In Step 5, each party P_i waits for the Gather protocol to output $X_i^{(v)}$ and then initiates the reveal protocol for the random beacon in round v.

- In Step 6, each party P_i waits for the random beacon to reveal the vector $(r_1^{(v)}, \ldots, r_n^{(v)})$ and sets $\mathit{rank}_j^{(v)} \leftarrow r_j^{(v)}$ for each $j \in X_i^{(v)}$, and then computes $\mathit{prevote}_i^{(v)}$ as before.
- Steps 7 and 8 are unchanged.

With these changes, all of the analysis in Section 4.4 goes through essentially unchanged, except that in Theorem 4.4, we use the pseudorandomness property of the random beacon in place of the hiding property of the secret generating function associated with the ASKS protocol and the pseudorandomness property of the rank derivation function.

We also mention that in the typical case where all parties vote for themselves in the first round, we can optimize the protocol even further by completely eliminating the reliable broadcast of the initial votes. In this case $\mathsf{GatherValid}_i^{(v)}$ for $v = 1$ is just Valid_i.

While this variant is simpler that Π_{VABA}, it suffers from the obvious drawback of requiring a setup assumption that provides the random beacon. Nevertheless, we think this variant (which is also presented in [DDL+24]) is interesting for at least two reasons:

- It gives a remarkably simple and efficient reduction from VABA to a random beacon. For example, in [DWZ23], an efficient reduction from VABA to a random beacon is given, but that reduction also goes through asynchronous binary agreement (ABA). The above variant of Π_{VABA} gives a direct reduction to a random beacon without the need for an ABA subprotocol.
- As discussed below in Section 5, in a setting where we already have "bootstrapped" a random beacon, the above variant may be attractive. We expect this variant is competitive with other VABA protocols, especially in terms of latency.

5 A note on bootstrapping an asynchronous distributed system

There are certainly more efficient ACS protocols than the one presented here, but they generally rely on some kind of setup assumption, such as access to a "random beacon". One such protocol is the FIN protocol [DWZ23], which relies on "lightweight" cryptography as well as a random beacon and a subprotocol for asynchronous binary agreement (ABA, which itself can be efficiently implemented using a random beacon). Another such protocol is the variant of our protocol discussed in Section 4.6.

Therefore, the most likely application of the ACS protocol presented here is as a critical component in a "bootstrapping" protocol that would generate many random beacon instances, which could then be used in a more efficient ACS protocol or other applications. Because our ACS protocol does not need a trusted setup, it could be run once in combination with a batch asynchronous secret sharing protocol, such as the one in [SS23], which also uses only lightweight cryptography and provides optimal resilience with no setup assumptions. For this application, one would want to use the random-oracle version of the protocol in [SS23], so as to avoid the need for a random beacon.

In more detail, this would work as follows. Every party would act as a dealer in an instance of the secret sharing protocol, disseminating a batch of L dealings, and the ACS protocol would then be used to agree a set of $n - t$ such batches. These shared secrets could be combined using "batch randomness extraction" techniques [HN06] so that we end up with a total of $\Omega(Ln)$ shared random secrets.

Suppose we use the secret sharing protocol in [SS23]. To estimate the communication complexity, we assume that the field over which the secret sharing is done has elements whose bit length is $\approx \kappa$, and that $L = \Omega(n \log n)$ and $\kappa = \Omega(n/\log n)$. With these parameters, we obtain an amortized communication complexity of $O(\kappa n)$ per shared random secret. Note that this amortized communication complexity holds on an "optimistic path" in which no dealer *provably* misbehaves. In particular, if t^* dealers provably misbehave, the communication complexity may increase by a factor of t^* but the identities of these t^* misbehaving dealers will become known to all of the honest parties (who may then take action to punish these dealers, or at least remove them from the committee). We note that the expected round complexity of generating all these shared random secrets (including the ACS step) is constant in expectation.

We can use one shared random secret to instantiate one random beacon instance. The communication complexity of opening one such secret is $O(\kappa n^2)$. Using well-known techniques (see [CP17]), one can also achieve an amortized communication complexity for opening secrets of $O(\kappa n)$ per secret if we open secrets in batches of size $\Omega(n)$ — while this is possible in some applications, such as general multi-party computation, it is not always possible in others. In any case, once we have "bootstrapped" many random beacon instances, we can use these in any applications that need a random beacon, including another, more efficient ACS protocol that relies on a random beacon (and we can use this other ACS protocol to generate more random beacon instances as needed, using the same technique as above).

Note that this approach to building a random beacon with no setup and only lightweight cryptography is much more efficient than the one in [BBB+23]. Their protocol has a bootstrapping phase that has a round complexity of $\Omega(\kappa \log n)$ and a communication complexity of $\Omega(\kappa^2 n^3 \log n)$. After that, the amortized communication complexity to both create a beacon and to open a beacon is $\Omega(\kappa n^2 \log n)$, which is worse than that of the approach sketched above, even if the secret sharing scheme falls off the optimistic path, and even more so in the optimistic case.

We can also use an ACS protocol to "bootstrap" other setup assumptions, such as a BLS threshold signature scheme [BLS01,Bol03], a KZG common reference string [KZG10], or the like. Such a setup may then enable the use of other, possibly even more efficient ACS protocols.

Acknowledgments

Thanks to Dennis Hofheinz for interesting discussions relating to Theorem 4.4.

References

AAPS23. I. Abraham, G. Asharov, A. Patra, and G. Stern. Perfectly secure asynchronous agreement on a core set in constant expected time. Cryptology ePrint Archive, Paper 2023/1130, 2023. https://eprint.iacr.org/2023/1130.

AJM+21. I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation, 2021. arXiv:2102.09041, http://arxiv.org/abs/2102.09041.

BBB+23. A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. Reiter. HashRand: Efficient asynchronous random beacon without threshold cryptographic setup. Cryptology ePrint Archive, Paper 2023/1755, 2023. https://eprint.iacr.org/2023/1755.

BKP11. M. Backes, A. Kate, and A. Patra. Computational verifiable secret sharing revisited. Cryptology ePrint Archive, Paper 2011/281, 2011. URL https://eprint.iacr.org/2011/281. https://eprint.iacr.org/2011/281.

BLS01. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

Bol03. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

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

Can00. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://ia.cr/2000/067.

CKPS01. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. Cryptology ePrint Archive, Paper 2001/006, 2001. https://eprint.iacr.org/2001/006.

CKS00. C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. Cryptology ePrint Archive, Paper 2000/034, 2000. https://eprint.iacr.org/2000/034.

CP17. A. Choudhury and A. Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theory*, 63(1):428–468, 2017.

CT05. C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In P. Fraigniaud, editor, *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 503–504. Springer, 2005.

DDL$^+$24. S. Dasi, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup. Asynchronous consensus without trusted setup or public-key cryptography. Cryptology ePrint Archive, Paper 2024/677, 2024. https://eprint.iacr.org/2024/677.

DGNW20. S. Dolev, B. Guo, J. Niu, and Z. Wang. Sodsbc: A post-quantum by design asynchronous blockchain framework. Cryptology ePrint Archive, Paper 2020/205, 2020. URL https://eprint.iacr.org/2020/205. https://eprint.iacr.org/2020/205.

DW20. S. Dolev and Z. Wang. SodsBC: Stream of distributed secrets for quantum-safe blockchain. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 247–256, Los Alamitos, CA, USA, 2020. IEEE Computer Society.

DWZ23. S. Duan, X. Wang, and H. Zhang. FIN: Practical signature-free asynchronous common subset in constant time. Cryptology ePrint Archive, Paper 2023/154, 2023. https://eprint.iacr.org/2023/154.

DXR21. S. Das, Z. Xiang, and L. Ren. Asynchronous data dissemination and its applications. In Y. Kim, J. Kim, G. Vigna, and E. Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2705–2721. ACM, 2021. Also available at https://eprint.iacr.org/2021/777.

GLL$^+$21. Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Efficient asynchronous byzantine agreement without private setups, 2021. arXiv:2106.07831, http://arxiv.org/abs/2106.07831.

GS23. J. Groth and V. Shoup. Fast batched asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2023/1175, 2023. https://eprint.iacr.org/2023/1175.

HN06. M. Hirt and J. B. Nielsen. Robust multiparty computation with linear communication complexity. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer, 2006.

HS11. D. Hofheinz and V. Shoup. GNUC: A new universal composability framework. Cryptology ePrint Archive, Paper 2011/303, 2011. https://eprint.iacr.org/2011/303.

HUMQ09. D. Hofheinz, D. Unruh, and J. Müller-Quade. Polynomial runtime and composability. Cryptology ePrint Archive, Paper 2009/023, 2009. https://eprint.iacr.org/2009/023.

KKK07. J. Katz, C.-Y. Koo, and R. Kumaresan. Improving the round complexity of vss in point-to-point networks. Cryptology ePrint Archive, Paper 2007/358, 2007. https://eprint.iacr.org/2007/358.

KMTZ11. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. Cryptology ePrint Archive, Paper 2011/310, 2011. https://eprint.iacr.org/2011/310.

KZG10. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

Nao03. M. Naor. On cryptographic assumptions and challenges. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

RB89. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In D. S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85. ACM, 1989.

SS23. V. Shoup and N. P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. Cryptology ePrint Archive, Paper 2023/536, 2023. https://eprint.iacr.org/2023/536.