Asymptotically Optimal Validated Asynchronous Byzantine Agreement

Ittai Abraham
VMware Research

Dahlia Malkhi
VMware Research

Alexander Spiegelman
VMware Research

ABSTRACT

We provide a new protocol for Validated Asynchronous Byzantine Agreement in the authenticated setting. Validated (multi-valued) Asynchronous Byzantine Agreement is a key building block in constructing Atomic Broadcast and fault-tolerant state machine replication in the asynchronous setting. Our protocol has optimal resilience of $f < n/3$ Byzantine failures and asymptotically optimal expected $O(1)$ running time to reach agreement. Honest parties in our protocol send only an expected $O(n^2)$ messages where each message contains a value and a constant number of signatures. Hence our total expected communication is $O(n^2)$ words. The best previous result of Cachin et al. from 2001 solves Validated Byzantine Agreement with optimal resilience and $O(1)$ expected time but with $O(n^3)$ expected word communication. Our work addresses an open question of Cachin et al. from 2001 and improves the expected word communication from $O(n^3)$ to asymptotically optimal $O(n^2)$.

CCS CONCEPTS

• **Theory of computation** → **Distributed algorithms**; • **Security and privacy** → *Cryptography*.

KEYWORDS

byzantine agreement, optimal protocol, asynchronous, validated

ACM Reference Format:
Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3293611.3331612

1 INTRODUCTION

Byzantine agreement is a fundamental problem in computer science introduced by Pease, Shostak and Lamport [31] in 1980. In Byzantine Agreement there are n parties each of which has an input value, at most $f < n$ are corrupted (i.e., controlled by an adversary), and the goal of the honest parties is to decide on a unique value. Many models with various assumptions have been proposed in the literature.

Renewed interest in Byzantine Agreement follows from the need to implement Atomic Broadcast and fault tolerant state machine replication in the asynchronous setting [12, 16, 27]. In this model, Cachin et al. [10] defined the problems of Atomic Broadcast and Validated Asynchronous Byzantine Agreement (VABA) to guarantee a decision on some party's input, provided it satisfies a globally verifiable *external validity* condition. They show how to solve VABA in the *authenticated* setting, namely there is a trusted setup phase and security depends on computational hardness assumptions and the random oracle model. Their solution achieves optimal resilience $n = 3f + 1$, asymptotically optimal time $O(1)$, and $O(n^3)$ expected word communication. Improving the expected word communication for VABA in this model from $O(n^3)$ to the $O(n^2)$ is an open problem stated in [10] and has been open for almost 20 years.

This paper presents the first VABA solution with optimal resilience, asymptotically optimal time whose expected word communication is $O(n^2)$, thus closing this gap. More precisely, we prove the following theorem:

Theorem 1. *There exists a protocol among n parties that solves VABA in the authenticated setting and is secure against an adaptive adversary that controls up to $f < n/3$ parties, with expected $O(n^2)$ word communication and expected constant running time.*

Complexity Measures. Bracha [9] shows that even strictly weaker primitives than Asynchronous Byzantine agreement can only be solved when the number of parties n is larger than $3f$ where f is the maximum number of parties the adversary can corrupt. We therefore say that a solution has *optimal resilience* if it solves Byzantine agreement for $n = 3f + 1$. A theorem of Fischer, Lynch and Paterson [19] states that any protocol solving Asynchronous Agreement must have a non-terminating execution even in the face of a single (benign) failure. Ben-Or [7] shows that randomization can be used to make such non-terminating executions become events with probability 0. Feldman and Micali [18] show that Asynchronous Byzantine Agreement can be solved with optimal resilience $n = 3f + 1$ and with an expected $O(1)$ asynchronous running time (where running time is the maximum duration as defined by Canetti and Rabin [13] and is essentially the number of steps when the protocol is embedded into a lock-step timing model). We therefore say that a solution has *asymptotically optimal time* if it solves Byzantine agreement using an expected $O(1)$ running time. We show in the full paper [2] that a recent lower bound of Abraham et al. [1] implies that any protocol solving Asynchronous Byzantine Agreement against an adaptive adversary (and without a constant error probability) must have the honest parties send expected $\Omega(n^2)$ messages. We therefore say that a solution has *asymptotically optimal word communication* if it solves Byzantine agreement using an expected $O(n^2)$ messages and each message contains just a single *word* where we assume a word contains a constant number of signatures and domain values.

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07…$15.00
https://doi.org/10.1145/3293611.3331612

Multi-valued Agreement. A simplified version of the agreement problem is the *binary agreement problem* in which the inputs of the parties are restricted to the set $\{0, 1\}$. A fundamental work by Cachin, Kursawe, and Shoup [11] was the first to give an optimal algorithm in terms of time, resilience, and word communication in the random oracle model, which they formalized to fit the distributed settings. In particular, the algorithm withstands up to $f < n/3$ Byzantine failures, runs in constant expected number of asynchronous views (rounds), and the expected communication cost is $O(n^2)$ messages of the size of one or two RSA signatures [33]. A more recent work by Mostéfaoui et al. [28] shows how to achieve optimality without any cryptographic assumptions besides the existence of a common random coin[1].

As for the multi-valued Byzantine agreement, the original problem specification due to Lamport et al. [31] was motivated by the following setting: Four computers in control of a space-shuttle cockpit need to reach agreement on a sensor reading, despite one being potentially faulty. The problem was captured via the following condition:

Definition 1 (Weak validity [15]). *If all honest parties propose a value v, then every honest party that terminates decides v.*

Note that while the Weak Validity condition is well defined, it says nothing about a situation in which parties propose different values, allowing them to (1) return some default value $\perp$ that indicates that no agreement was reached or (2) agree on a value proposed by a corrupted party. Mostéfaoui et al. [29] consider a slightly stronger property in which only a value proposed by an honest party or $\perp$ are allowed to be returned [2]. However, honest parties may still decide $\perp$ if they initially disagree. In particular, it is not clear how this slightly stronger validity property can be used to solve Atomic Broadcast [10].

Cachin et al. formulated in [10] a problem specification that captures the practical settings where parties propose updates to a replicated state. Agreement is formed on a sequence of updates, hence a non-default decision is needed in order to make progress. To prevent updates from rogue parties, the model is extended with an *External Validity* predicate as follows:

Definition 2 (External validity). *If an honest party decides on a value v, then v is externally valid.*

Mostéfaoui et al. presented in [29] a signature-free deterministic reduction from their binary agreement protocol [28] that solves asynchronous Byzantine Agreement with Weak validity. It has optimal resilience and asymptotically optimal time and word communication. Unfortunately, BA with Weak validity only does not solve Atomic Broadcast or State Machine Replication (SMR).

Cachin et al. gave in [10] a reduction from their binary agreement algorithm [11] to VABA and also showed how to use it in order to implement an atomic broadcast. Their VABA protocol provides external validity, has optimal resilience, asymptotically optimal time, and expected message complexity $O(n^3)$. That paper explicitly mentions the open problem of improving the expected word communication from $O(n^3)$ to $O(n^2)$.

Our Contribution. The main contribution of this paper is solving this open question. Just like [10], our protocol solves Asynchronous Byzantine agreement with external validity (VABA), has optimal resilience and asymptotically optimal time. Improving on [10], our expected word communication is also asymptotically optimal. In particular, honest parties send a total expected $O(n^2)$ messages, which is optimal and each message is roughly the size of one or two threshold signatures.

Our protocol is secure against an adaptive adversary. This follows from using adaptively secure threshold signatures of Libert et al. [25] and adaptively secure common coin protocol of Loss and Moran [26]. Cachin et al. [10] note that their binary protocol [11] and their Validated protocol [10] also immediately generalize to be secure against adaptive adversaries by using the primitives above.

Techniques and Challenges. At a high level, the main conceptual contribution of this paper is a new approach for solving Byzantine agreement in the *asynchronous model* by using view-change based techniques that were traditionally used only in the *partially synchronous model* [14, 17, 24]. Traditionally, all view-change based protocols depend on some timeout mechanism. To adopt view-change based protocols to full asynchronous models and obtain optimal word communication against an adaptive adversary our work needs to overcome three core challenges (1) remove timeouts (2) provide safety and liveness against an an adaptive adversary and (3) reduce communication to a minimum.

Unlike previous constructions (e.g., [10, 27]), our protocol does not go through a randomized binary agreement black-box. Instead, much like Katz and Koo's Synchronous Byzantine agreement protocol [22], in each view, we run n parallel leader-based threads and then use a random leader election primitive to decide which leader is elected in hindsight. To adopt this idea to the asynchronous model each leader-based thread is a separate instance of the leader-based paradigm of [14] that uses view-change to replace leaders safely in asynchronous settings.

The idea of letting n parties concurrently broadcast their values and then using a leader election to determine which value should be further considered was first introduced in [8]. However, while in [8] they use this value as an input to a Byzantine agreement instance, we use a view-change mechanism to determine if this value can be safely decided or should it be adopted to the following views otherwise. By opening the black box agreement protocols used by both [10] and [8] we are able to reduce the communication complexity to a minimum. Just like [22], not all honest parties in our protocol reach agreement in the same view. To guarantee safety between different views we use a view-change protocol that guarantees that new leaders can propose only safe values. To guarantee liveness between different views, instead of using a timeout, the trigger is the knowledge that $n - f$ leader-based threads have made sufficient progress inside their protocol.

On the one hand, to obtain $O(n^2)$ word communication per view, we need each of the n leader-based protocols to use just $O(n)$ words and we need the global view change protocol to use just $O(n^2)$ words. On the other hand, to guarantee progress we must guarantee that our view change protocol will allow progress even in

[1] while the construction of [28] requires only $O(n^2)$ bits given a common coin, the word communication of the resulting binary Byzantine Agreement protocol is dominated by the common random coin protocol that requires threshold signatures and $O(n^2)$ word communication.

[2] Note that this property is much weaker than the one required for strong consensus [20, 30] since parties are allowed to return $\perp$.

asynchronous settings. To balance between frugal communication and liveness we adopt a four step leader-based protocol that is inspired by an approach taken in the partial synchronous model by Yin et al. [34].

To obtain the optimal $O(1)$ expected time against an adaptive adversary, the next challenge is to guarantee that when the first honest party enters the leader election phase, there is a constant fraction of potential leaders such that if one of them is elected then all honest parties will decide in constant time. By electing the leader after many leaders have completed their work we limit the adaptive power of the adversary. Moreover, if the elected leader did not complete its broadcast, we need a mechanism to allow parties to abandon the elected leader's broadcast before running the global view-change protocol.

Cryptography vs Full information. In this paper we assume an environment with a trusted setup providing authenticated but asynchronous communication channels, and a computationally bounded adversary that cannot read the private state of non-faulty parties.

It is natural to ask if similar results can be obtained in the *full information model* where the adversary is computationally unbounded and can see the state of all parties. Recent years has seen major advances in this model. Kapron et al. [21] solve ABA with just $O(n^2)$ bits against a static full information adversary that controls less than $n/6$ parties. King and Saia [23] solve ABA against a dynamic full information adversary that controls less than $n/500$. Bar Joseph and Ben Or [4] show that there is a fundamental gap between the full information model and the cryptographic model. They prove that any randomized solution against an adaptive full information fail-stop adversary that controls a constant fraction of the parties must take at least $\Omega(\sqrt{n/\log n})$ rounds in the synchronous model. By using cryptographic assumptions we overcome this lower bound and obtain the asymptotically optimal $O(1)$ expected rounds and $O(n^2)$ word communication in the asynchronous model. We are not aware of any lower bound for obtaining $O(n^2)$ bits against a dynamic full information adversary with optimal resilience in the asynchronous model.

2 MODEL

In order to reason about distributed algorithms in cryptographic settings we adopt the model defined in [10, 11]. We consider an asynchronous message passing system consisting of a set Π of n parties, an adaptive adversary, and a trusted dealer. The adversary may control up to $f < n/3$ parties during an execution. An adaptive adversary is not restricted to choose which parties to corrupt at the beginning of an execution, but is free to corrupt (up to f) parties on the fly. Note that once a party is corrupted, it remains corrupted, and we call it *faulty*. A party that is never corrupted in an execution is called *honest*. To be able to use the threshold signature from [25] and the coin tossing from [26] we assume the cryptographic random oracle model. In addition, as in [6], we treat a hash function like a random oracle.

We assume an initial setup before every execution in which the trusted dealer generates the initial states of all parties, and we assume that the adversary cannot obtain the states of honest parties at any time during an execution.

Computation. Following [10, 11], we use standard modern cryptographic assumptions and definitions. We model the computations made by all system components as probabilistic Turing machines, and bound the number of computational basic steps allowed by the adversary by a polynomial in a *security parameter* k. A function $\epsilon(k)$ is *negligible* in k if for all $c > 0$ there exists a k_0 s.t. $\epsilon(k) < 1/k^c$ for all $k > k_0$. A computational problem is called *infeasible* if any polynomial time probabilistic algorithm solves it only with negligible probability. Note that by the definition of infeasible problems, the probability to solve at least one such problem out of a polynomial in k number of problems is negligible. Intuitively, this means that for any protocol P that uses a polynomial in k number of infeasible problems, if P is correct provided that the adversary does not solve one of its infeasible problems, then the protocol is correct except with negligible probability. We assume that the number of parties n is bounded by a polynomial in k.

Communication. We assume asynchronous links controlled by the adversary, that is, the adversary can see all messages and decide when and what messages to deliver. In order to fit the communication model with the computational assumptions, we restrict the adversary to perform no more than a polynomial in k number of computation steps between the time a message m from an honest party p_i is sent to an honest party p_j and the time m is delivered by p_j[3]. In addition, for simplicity, we assume that messages are *authenticated* in a sense that if an honest party p_i receives a message m indicating that m was sent by an honest party p_j, then m was indeed generated by p_j and sent to p_i at some prior time. This assumption is reasonable since it can be easily implemented with standard symmetric-key cryptographic techniques [5] in our model.

Termination. Note that the traditional definition of the liveness property in distributed system, which requires that all correct (honest) parties *eventually* terminate provided that all messages between correct (honest) parties eventually arrive, does not make sense in this model. This is because the traditional definition allows the following:

- Unbounded delivery time between honest parties, which potentially gives the adversary unbounded time to solve infeasible problems.
- Unbounded runs that potentially may consist of an unbounded number of infeasible problems, and thus the probability that the adversary manages to solve one is not negligible.

Following Cachin et al. [10, 11], we address the first concern by restricting the number of computation steps the adversary makes during message transmission among honest parties. So as long as the total number of messages in the protocol is polynomial in k, the error probability remains negligible. To deal with the second concern, we do not use a standard liveness property in this paper, but instead we reason about the total number of messages required for all honest parties to terminate. We adopt the following definition from [10, 11]:

Definition 3 (Uniformly Bounded Statistic). *Let X be a random variable. We say that X is probabilistically uniformly bounded*

[3]Note that although this restriction gives some upper bound on the communication in terms of the adversary local speed, the model is still asynchronous since speeds of different parties are completely unrelated.

if there exist a fixed polynomial $T(k)$ and a fixed negligible functions $\delta(l)$ and $\epsilon(k)$ such that for all $l, k \geq 0$,

$$Pr[X > lT(k)] \leq \delta(l) + \epsilon(k)$$

With the above definition Cachin et al. [10, 11] define a progress property that makes sense in the cryptographic settings:

- *Efficiency: The number of messages generated by the honest parties is probabilistically uniformly bounded*

The efficiency property implies that the probability of the adversary to solve an infeasible problem is negligible, which makes it possible to reason about the correctness of the primitives' properties. However, note that this property can be trivially satisfied by a protocol that never terminates but also never sends any messages. Therefore, in order for a primitive to be meaningful in this model, Cachin et al. [10, 11] require another property:

- *Termination*[4]: If all messages sent by honest parties have been delivered, then all honest parties terminated.

In this paper we consider both efficiency and termination properties as defined in [10, 11]. However, note that when considering an adaptive adversary, it is also possible to define a slightly weaker termination property:

- *Weak termination*: If all messages sent by parties before they were corrupted have been delivered, then all honest parties terminated.

Note that while any protocol that satisfies termination satisfies weak termination as well, a lower bound for termination does not apply for weak termination. Indeed our lower bound (see full paper [2]) is for protocols that obtain the termination property. We leave the study of lower bounds for protocols with weak termination as an open question.

Complexity. We use the following standard complexity notions (see for example Cannetti and Rabin [13]). We measure the expected *word communication* of our protocol as the maximum over all inputs and applicable adversaries of the expected total number of *words* sent by honest parties where expectation is taken over the random inputs of the players and of the adversary. We assume a finite domain $\mathcal{V}$ of valid values for the Byzantine agreement problem, and say that a word can contain a constant number of signatures (see Section and domain values 2.1). We measure the expected *running time* of our protocol as the maximum over all inputs and applicable adversaries of the expected *duration* where expectation is taken over the random inputs of the players and of the adversary. The duration of an execution is the total time until all honest players have terminated divided by the longest delay of a message in this execution. Essentially the duration of an execution is the number of steps taken if this execution is re-run in lock-step model where each message takes exactly one time step.

Following Cachin et al. [10] (see Lemma 1 therein), in order to show that our view-based protocol runs in an expected constant running time and has expected $O(n^2)$ word communication, it is enough to show that:

- every view consists of $R(k) = O(n^2)$ messages that consist of one word, and
- the total number of messages is probabilistically uniformly bounded by R.

2.1 Cryptographic abstractions

The main focus of this paper is on a novel distributed algorithm, which uses cryptographic tools as black-boxes. To this end, we present our protocol assuming the existence of two cryptographic abstractions:

- **Threshold signatures scheme.** We assume that each party p_i has a private function $share\text{-}sign_i$, and we assume 3 public functions: *share-validate*, *threshold-sign*, and *threshold-validate*. Informally, given $n-f$ validated shares, the function *threshold-sign* returns a valid threshold signature.
- **Threshold coin-tossing.** We assume that each party p_i has a private function $coin\text{-}share_i$, and we assume 2 public functions: *coin-share-validate* and *coin-toss*. Informally, given $f+1$ validated coin shares, the function *coin-toss* returns a unique and pseudorandom number from the range $[1, \ldots, n]$.

The for formal definitions and implementation details can be found in the full paper [2].

2.2 Validated asynchronous byzantine agreement (VABA)

In this paper we follow Cachin et al. [10] and define a (multi-valued) Byzantine agreement with an external validity function we call *EX-VABA-VAL*. The purpose of this function is to determine whether a value is externally valid for agreement, and the validity property of the VABA requires parties to decide only on externally valid values. To rule out trivial solutions in which parties always decide on some pre-defined externally valid value, we could try to add a requirement that only a value that was actually proposed by some party (honest or not) can be decided. However, since byzantine parties can propose one value and act as if they proposed a different value, this requirement seems to be impossible to achieve. Instead, we add another property to the VABA problem defined in [10], which we call *quality*. The *quality* property bounds the probability that the decision value was proposed by an honest party. Note that every probability that is grater than zero rules out trivial solutions, however, to capture the "fairness" of the decision value we require this probability to be $1/2$ [5]. We conjecture that higher probability is impossible to achieve in our model. The formal definition of the VABA protocol is given below.

Definition 4 (Validated Byzantine Agreement). *A protocol solves validated Byzantine agreement if it satisfies the following properties except with negligible probability:*

- *Validity: If an honest party decides an a value v, then* EX-VABA-VAL$(v) = true$.
- *Quality: The probability of choosing a value that was proposed by an honest party is at least* $1/2$.
- *Agreement: All honest parties that terminate decide on the same value.*

[4] Called liveness in [11], but we find this name confusing since it is not a liveness [3] property.

[5] Note that although they do not explicitly define it, the protocol of Cachin et al. [10, 11] already obtains this Quality property.

- *Termination: If all honest parties start with externally valid values and all messages sent among honest parties have been delivered, then all honest parties decide.*
- *Efficiency: The number of messages generated by the honest parties is probabilistically uniformly bounded.*

3 ASYMPTOTICALLY OPTIMAL VABA PROTOCOL

In this section we give a protocol for asynchronous byzantine agreement, secure against an adaptive adversary that controls up to $f < n/3$ parties, with expected word communication $O(n^2)$ and expected running time $O(1)$. We present a modular implementation, which consists of three sub protocols: a simple two-round broadcast primitive we call *Provable-Broadcast*, a simple *Leader-Election* protocol, and another primitive, we call *Proposal-Promotion*, which is built on top of 4 sequential instances of Provable-Broadcast. In Section 3.1, we give an overview of the protocols, and in Section 3.2 we present detailed pseudocode and description.

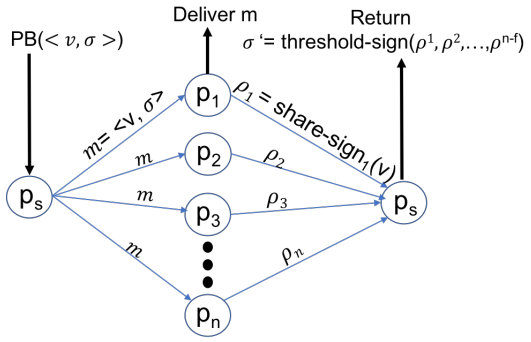

Figure 1: Provable-Broadcast illustration. The message external validation by all parties is omitted. When a party receives a value v and a proof σ from the sender, it first uses σ to externally validate v, and only then deliver, sign, and send its share ν back to the sender.

3.1 Our VABA protocol overview

Provable-Broadcast. The Provable-Broadcast is a simple two-round broadcast that adds an external validity to the basic echo multicast protocol from [10, 32]. In the first round, the sender sends a message, that contains a value and a proof for external validity, to all parties. In the second round, each party first validates, with an external validation function (which implements a logic of the protocol that is implemented on top of the broadcast), that the message is valid. Then, it (1) delivers the message (value and proof), (2) threshold-signs it, and (3) sends the signed share back to the sender. When the sender gets $n - f$ properly signed shares, it combines them into one threshold signature and returns it. An illustration of the Provable-Broadcast protocol appears in Figure 1. Informally, the provability property of this simple broadcast satisfies the following: (1) An honest sender returns a threshold signature which he can later use to prove that at least $f + 1$ honest parties delivered the message; and (2) a faulty sender cannot produce two such proofs for two different messages. This proof is later used by the sender for external validity in other parts of the VABA protocol.

Proposal-Promotion. As we explain below, the VABA protocol works in a view-based manner, where in each view the parties participate in n concurrent Proposal-Promotion sub-protocols. Each party i is acting as the leader of the i^{th} Proposal-Promotion invocation and tries to promote its proposal by distributing them to all other parties. The distribution inside a Proposal-Promotion sub-protocol consists of 4 sequential steps, each of which invokes an instances of a Provable-Broadcast. An illustration of the sub-protocol appears in Figure 2. All parties participating in an instance of a Proposal-Promotion maintain 3 local variables: **key**, **lock**, and **commit**. The **key** variable stores the message (value and proof) delivered in the second step (second instance of the Provable-Broadcast), the **lock** variable stores the message delivered in the third step, and the **commit** variable stores the message delivered in the fourth step. For clarity, we refer to these delivery events as **key** delivery, **lock** delivery, and **commit** delivery, respectively.

A leader (the initiating party) of a Proposal-Promotion invocation that successfully completes gets a proof of its completeness, which is the threshold signature output of the last Provable-Broadcast instance therein. As for the external validity of the Provable-Broadcast steps inside a Proposal-Promotion, we distinguish between the first instance and the last 3. For the first instance (first step) the leader provides a proof *key* that the proposed value (the one to be promoted) is safe for the current view of the VABA protocol. (More details on this below). For each of the other 3 instances, the leader provides the threshold signature output of the preceding Provable-Broadcast instance, which is a proof of its successful completion. Informally, the provability property together with the external validity of Provable-Broadcast guarantee the following:

- All values delivered by honest parties in the **key**, **lock**, and **commit** deliveries are (1) equal and (2) satisfy the safety logic of the higher level VABA protocol for this view.
- A valid completeness proof indicates that at least $f + 1$ honest parties delivered **commit**. In addition, if an honest party delivered **commit**, then at least $f + 1$ honest parties previously delivered **lock**, and if an honest party delivered **lock**, then at least $f + 1$ honest parties previously delivered **key**.

Leader-Election. The Leader-Election primitive is a simple one-round protocol that uses the cryptographic threshold coin-tossing abstraction to elect a unique leader [6] among the parties for each view. In every view R, each party p_i uses *coin-share*$_i(R)$ to produce a share ρ_i and sends it to all other parties. When a party gets $f + 1$ proper shares for view R it uses the function *coin-toss* to compute the leader of view R. Informally, the main properties are that (1) all honest parties agree on the leader, (2) the adversary cannot predict who is the leader unless one honest party participates, and (3) every party has an equal probability to became the leader of every view.

[6] Note that every instance of a Proposal-Promotion has a unique leader, but here we choose among them one unique leader for each view.

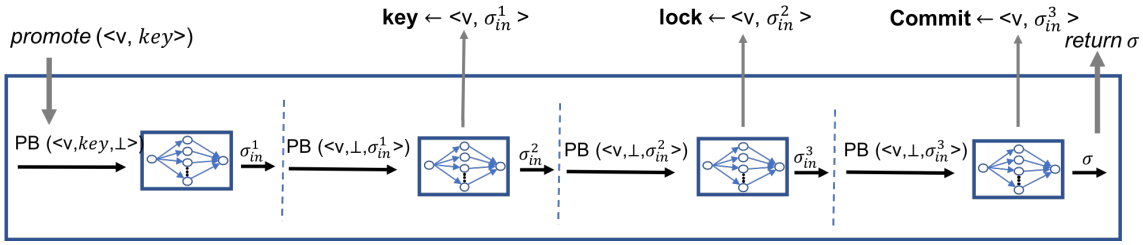

Figure 2: Illustration of the 4 steps of a Proposal-Promotion sub protocol.

Putting everything together into the VABA protocol. The Proposal-Promotion and the Leader-Election sub-protocols are used as building blocks for our VABA protocol. The protocol works in a view-based manner, where each view consists of three phases: *Leader-nomination*, *Leader-election*, and *View-change*. In the *Leader-nomination Phase*, each party promotes its proposal value via the Proposal-Promotion sub-protocol. Parties wait to learn that $n - f$ instances of Proposal-Promotion, initiated by different parties, have *completed* by getting $n - f$ such proofs. (Recall that the *promote* function returns a proof of completion that indicates that at least $f + 1$ honest parties delivered **commit** in that instance). Then, in the *Leader-election Phase*, parties choose a unique leader uniformly at random, via the Leader-Election primitive. Finally, in the *View-change Phase*, parties learn what happened in the elected leader's Proposal-Promotion protocol and update their local state accordingly. This is done by an all-to-all communication, where each party sends to all other parties the **key**, **lock**, and **commit** it delivered (if any) in the Proposal-Promotion sub-protocol initiated by the chosen leader. If they learn that some party delivered **commit**$(\langle v, \sigma \rangle)$ (where σ is a valid proof for v to be committed), they can decide v. Otherwise, they need to carefully adopt a value, (by analysing what **key** and **lock** were delivered), and continue to the next view.

The local state of each party includes two essential variables *LOCK* and *KEY*, which are updated at the end of each round (based on the received view-change messages) and used by the first instance of the Provable-Broadcast in every Proposal-Promotion to determine whether the promoted value is externally valid. Informally, the *KEY* variable stores a view number *view*, a value v and a proof σ s.t. (1) a **key** $= \langle v, \sigma \rangle$ was received in the *View-change* phase of *view*, (2) σ is a proof for v's delivery in the second instance of the Provable-Broadcast in the Proposal-Promotion of *view*, and (3) *view* is the highest view that satisfies (1) and (2). The *LOCK* variable stores a view number that corresponds to the highest view in which a view-change message with a valid **lock** (contains $\langle v, \sigma \rangle$ s.t. σ is a proof for v's delivery in the third instance of the Provable-Broadcast in the Proposal-Promotion of *view*) was received. Informally, the external validity function in the first instance of the Provable-Broadcast of each Proposal-Promotion checks that the promoted value is attached to a valid key that was obtained in view that is at least large as the the view of the local variable *LOCK*. An high-level illustration appears in Figure 3, and more details are given in the description below.

Our protocol guarantees that at least $n - f$ Proposal-Promotion instances complete in the broadcast phase before a leader is elected. If the elected leader has completed its Proposal-Promotion (implying that at least $f + 1$ honest parties delivered a **commit**) then even an adaptive adversary cannot prevent progress. Otherwise, the view-change phase ensures that agreement is not violated even if a bad leader is elected. Since the probability to choose a leader that completed its Proposal-Promotion is constant, the number of views in the protocol is constant in expectation. More concretely, the probability to choose a completed broadcast is greater than 2/3, and thus the number of views in expectation is less than 3/2.

Communication complexity. Each of the **key**, **lock**, and **commit** variables stores one value and one threshold signature. Therefore, the total word complexity of the View-change phase is $O(n^2)$. The word complexity of a single instance of Proposal-Promotion is $O(n)$, which leads to total $O(n^2)$ word complexity of the Leader-nomination phase. Since the Leader-election phase has a single round all-to-all communication with messages of size 1 word, we get that all to all, our VABA protocol has an optimal $O(n^2)$ word complexity in expectation.

3.2 Detailed VABA protocol description

In this section we give a detailed description of our VABA protocol. A formal proof and a complexity analysis appear in the full paper [2]. For better readability we use the top down approach. We first present the VABA protocol and then describe the sub-protocols it uses. The formal definition of the properties as well as the pseudocode of the simple Leader-election primitive are deferred to Appendix B. As defined in the model, there is an external function, called *EX-VABA-VAL*(v), which checks if v is a valid value for the byzantine agreement. An instance of the high-level VABA protocol is identified with a parameter *id*. Every instance of every sub-protocol is identified with a parameter *ID*, which extends *id*. For example, the *ID* of p_i's Proposal-Promotion in view *view* is $\langle id, i, view \rangle$, and the *ID* of the second instance of the Provable-Broadcast therein is $\langle \langle id, i, view \rangle, 2 \rangle$.

Local variables. We start by presenting the local variables parties maintain (see Algorithm 1). All parties maintain two cross-view variables, *LOCK* and *KEY*:

- The *LOCK* variable stores the highest view number for which the party ever received a view-change message that includes a **lock** that was delivered in the Proposal-Promotion of the chosen leader of this view.
- The *KEY* variable stores the 3-tuple: view, proof and value, derived from the maximum view for which the party ever

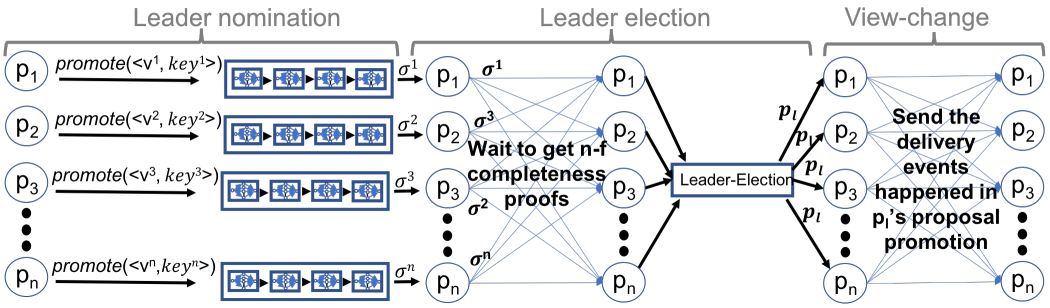

Figure 3: An illustration of a single view in the VABA protocol. Blue thin arrows represent messages and black thick arrows represent invocations and responds of sub-protocols.

received a view-change message that includes a **key** (composing of value and proof) that was delivered in the Proposal-Promotion of the chosen leader of this view.

In addition, all parties maintain per-view variables: For a view $view$, each party stores the elected leader of this view in $Leader[view]$, and few variables, $PPdone[view]$, $PPskip[view]$, $skip[view]$, which are used to make sure that at least $n - f$ parties completed their Proposal-Promotion before some honest party moves to the leader election phase.

Algorithm 1 Local variables initialization for party p_i.

$LOCK \leftarrow 0$
$KEY \leftarrow \langle 0, v_i, \perp \rangle$ with selectors *view*, *value*, *proof* ▷ v_i is p_i's input
for every $view \geq 1$, **initialize**:
  $Leader[view] \leftarrow \perp$
  $PPdone[view] \leftarrow 0$
  $PPskip[view] \leftarrow \{\}$
  $skip[view] \leftarrow false$

The pseudocode of the top VABA protocol appears in algorithms 2 and 3. We next describe the phases in each view:

Leader nomination phase. Each party promotes, using a Proposal-Promotion sub-protocol, the value and *key* it has adopted from the previous views, as determined in the View-Change Phase (explained below); at view 1, a party promotes its input and an empty key. Parties participate in n concurrent Proposal-Promotions, and each party sends a DONE notification about the completion of its own Proposal-Promotion carrying its output for proof. When a party receives $n-f$ such notifications, it sends a signature share on a SKIP-SHARE message. For termination, each party waits to obtain (directly or indirectly) a combined threshold SKIP signature, forwards it to others, and moves to the Leader-election phase (even if its Proposal-Promotion has not completed due to *abandon* invocation of other parties).

Leader election phase. Once a party enters the Leader-election Phase, it abandons all the Proposal-Promotion instances (more details on *abandon* is given below). The parties elect a leader via the leader election primitive and continue to the next phase as if only the chosen leader's Proposal-Promotion ever occurred. From now on, we refer to the Proposal-Promotion of the leader of a view $view$ as the Proposal-Promotion of view $view$, and refer to its delivery events as the deliveries of view $view$.

Algorithm 2 Validated asynchronous byzantine agreement with identification id: protocol for party p_i.

1: $view \leftarrow 1$
2: **while** true **do**
3: $ID \leftarrow \langle id, i, view \rangle$ ▷ Leader nomination phase
4: $key \leftarrow \langle KEY.view, KEY.proof \rangle$
5: $\sigma \leftarrow$ promote$(ID, \langle KEY.value, key \rangle)$
6: **wait** for $promote$ to return or $skip[view]$ to become *true*
7: **if** $skip[view] = false$ **then**
8: send "id, DONE, $view$, $KEY.value$, σ" to all parties
9: **wait** until $skip[view] = true$
 ▷ Leader election phase
10: **for all** k=1,...,n **do**
11: $abandon(\langle id, k, view \rangle)$ ▷ abandon Proposal-Promotion with $ID = \langle id, k, view \rangle$
12: $Leader[view] \leftarrow elect(\langle id, view \rangle)$
 ▷ View-change phase
13: $ID_{leader} = \langle id, Leader[view], view \rangle$
14: send "id, VIEW-CHANGE, $view$, $getKey(ID_{leader})$, $getLock(ID_{leader})$, $getCommit(ID_{leader})$" to all parties
15: **wait for** VIEW-CHANGE messages from $n - f$ different parties
16: $view \leftarrow view + 1$

View-change phase. In the View-change Phase of every view, parties report their delivery events (value and proof) of this view (delivery in the chosen leader's Proposal-Promotion) to all other parties. Each party waits to collect $n-f$ reports. Recall that Proposal-Promotion provides the following guarantees: If some honest party delivers a **commit**, then $f + 1$ honest parties deliver **lock**, hence all honest parties recieve this **lock** in the view-change exchange. Similarly, if some honest party delivers a **lock**, then $f + 1$ honest parties deliver a **key**, hence all honest parties receive this **key**.

Once a party has collected $n - f$ view-change messages, it processes them as follows. If it receives a **commit** with a value v, it decides v. Otherwise, if it receives a **lock**, it increases its $LOCK$ variable to the current view. Last, if it receives a **key**, it updates its KEY variable to store the current view and the received **key**. If it

Algorithm 3 Validated asynchronous byzantine agreement with identification *id*: messages.

1: **upon receiving** "id, DONE, $view$, v, σ" from party p_k for the first time **do**
2: **if** *threshold-validate*($\langle\langle\langle id, k, view\rangle, 4\rangle, v\rangle, \sigma$) **then**
3: $PPdone[view] \leftarrow PPdone[view] + 1$
4: **if** "SKIP-SHARE" message was not sent yet in view $view$ **then**
5: **if** $PPdone[view] = n - f$ **then**
6: $\rho \leftarrow$ *sigh-share*($\langle id$, SKIP, $view\rangle$)
7: send "id, SKIP-SHARE, $view$, ρ" to all parties

8: **upon receiving** "id, SKIP-SHARE, $view$, ρ" from party p_k for the first time in view $view$ **do**
9: **if** *share-validate*($\langle id$, SKIP, $view\rangle$, k, ρ) **then**
10: $PPskip[view] \leftarrow PPskip[view] \cup \{\rho\}$
11: **if** $|PPskip[view]| = n - f$ **then**
12: $\sigma \leftarrow$ *threshold-sign*($PPskip[view]$)
13: send "id, SKIP, $view$, σ" to all parties

14: **upon receiving** "id, SKIP, $view$, σ" **do**
15: **if** *threshold-validate*($\langle id$, SKIP, $view\rangle$, σ) = true **then**
16: $skip[view] \leftarrow true$
17: **if** "SKIP" message was not sent yet in view $view$ **then**
18: send "id, SKIP, $view$, σ" to all parties

19: **upon receiving** "id, VIEW-CHANGE, $view$, $\langle v_2, \sigma_2\rangle$, $\langle v_3, \sigma_3\rangle$, $\langle v_4, \sigma_4\rangle$" **do**
20: *leader* $\leftarrow$ *Leader*[$view$]
21: **if** $v_4 \neq \perp$ **then**
22: **if** *threshold-validate*($\langle\langle\langle id$, *leader*, $view\rangle, 3\rangle, v_4\rangle, \sigma_4$) **then**
23: **decide** v_4
24: **if** $v_3 \neq \perp \wedge view > lock$ **then**
25: **if** *threshold-validate*($\langle\langle\langle id$, *leader*, $view\rangle, 2\rangle, v_3\rangle, \sigma_3$) **then**
26: $LOCK \leftarrow view$
27: **if** $v_2 \neq \perp \wedge view > key.view$ **then**
28: **if** *threshold-validate*($\langle\langle\langle id$, *leader*, $view\rangle, 1\rangle, v_2\rangle, \sigma_2$) **then**
29: $KEY \leftarrow \langle view, v_2, \sigma_2\rangle$

did not reach a decision, a party adopts the value v of its (up-to-date) *KEY* variable and moves to the next view, where it *promotes* v together with *KEY* as proof for the external validation function (*EX-PB-VAL*) of the Provable-Broadcast.

As mentioned above, a party participates in a Proposal-Promotion only if the message $m = \langle v, \langle \mathcal{R}, \sigma_{\mathbf{key}}\rangle\rangle$ (note that $\langle \mathcal{R}, v, \sigma_{\mathbf{key}}\rangle$ = *KEY*) passes the external validation test. The external validation *EX-PB-VAL* includes a crucial *key-locking mechanism* (see Algorithm 5, lines 18 to 28). In particular, in view $j > 1$, a party checks that the $\sigma_{\mathbf{key}}$ is valid for v and $\mathcal{R}$ (meaning that $\sigma_{\mathbf{key}}$ is a proof that **key**($\langle v, \sigma_{\mathbf{key}}\rangle$) could have been deliverd by an honest party in the Proposal-Promotion of view $\mathcal{R}$), and that the view $\mathcal{R}$ is at least as large as the *LOCK* variable. We prove in the full paper [2] that the key-locking mechanism together with the fact that parties abandon all broadcasts before sending the view-change messages guarantee agreement and satisfy progress. Here we give some intuition for the proof:

- Lock Safety: If some party has a proof for **commit** delivery in view $\mathcal{R}$, then at least $f+1$ honest parties previously delivered **lock** and thus locked (*lock* = $\mathcal{R}$) in view $\mathcal{R}$.

Algorithm 4 Provable-Broadcast with identification *ID*: Protocol for the sender

Local variables initialization:
$S = \{\}$

1: **upon** *PB*(ID,$\langle v, \sigma\rangle$) invocation **do**
2: send "ID, SEND, $\langle v, \sigma\rangle$" to all parties
3: **wait** until $|S| = n - f$
4: **return** *threshold-sign*(S)

5: **upon receiving** "ID, ACK, ρ_k" form party p_k for the first time **do**
6: **if** *share-validate*($\langle ID, v\rangle$, k, ρ_k) = *true* **then**
7: $S \leftarrow S \cup \{\rho_k\}$

- Key Safety: If some party has a proof for **commit** delivery with a value v in view $\mathcal{R}$, then it is not possible for a party to have a valid **key** that contains a value other than v in view higher than or equal to $\mathcal{R}$.
- Key Progress: If some party p_i is locked in view $\mathcal{R}$, then at least $f+1$ honest parties delivered a **key** in $\mathcal{R}$ before sending the view-change messages of view $\mathcal{R}$, and thus all honest parties will have a *KEY* with view at least $\mathcal{R}$. Meaning that all honest parties will have a key to "unlock" p_i in the next view.

We now continue to the detailed description of the VABA sub-protocols:

Provable-Broadcast. In addition to what is mention in the overview above, Provable-Broadcast also exposes a *PB-abandon(ID)* API, which parties invoke to explicitly stop their participation in the Provable-Broadcast protocol associated with identification *ID* – no message is delivered and no signed share is sent after *PB-abandon(ID)* is invoked. A Provable-Broadcast of a message $m = \langle v, \sigma\rangle$ with identification *ID* is denoted *PB*(*ID*,*m*). The external validation function used by the Provable-Broadcast is denoted by *EX-PB-VAL*(ID, $\langle v, \sigma\rangle$). This function has access to the local variables of the high-level VABA protocol. For better readability, the formal definition of the properties satisfied by Provable-Broadcast is deferred to Appendix A. The pseudocode of the sub-protocol and the *EX-PB-VAL*((ID, $\langle v, \sigma\rangle$)) function appear in Algorithms 4 and 5. The *EX-PB-VAL*((ID, $\langle v, \sigma\rangle$)) function implements an important logic of the VABA protocol. The proof σ consists of two proofs $\langle view, \sigma_{key}\rangle$ and σ_{in}. If *ID* is an identification of a first Provable-Broadcast instance of some Proposal-Promotion sub-protocol, then $\langle view, \sigma_{key}\rangle$ is passed to function called *check-key* to check if v is a safe proposal for the current view by verifying that (1) σ_{key} is a valid **key** from view $view$ and (2) $view$ is not smaller than *LOCK*. Otherwise, *EX-PB-VAL* checks that σ_{in} is a valid output of the preceding instance of Provable-Broadcast by verifying that it is the correct threshold signature.

Proposal-Promotion. Besides the *promote* function mentioned in the overview, Proposal-Promotion also has an *abandon* function that invokes *PB-abandon* on all 4 instances of Provable-Broadcast therein. In addition, it also exposes *getKey*, *getLock*, and *getCommit* API for the VABA protocol to be able to get the delivered **key**, **lock**, and **commit**, respectively. For clarity and readability we do not

Algorithm 5 Provable-Broadcast with identification ID: Protocol for a party p_i

Local variables initialization:
$stop \leftarrow false$

1: **upon receiving** "ID, SEND, $\langle v, \sigma \rangle$" from the sender **do**
▷ parse ID as $\langle\langle id, k, j\rangle, l\rangle$; P_k is the sender
2: **if** $stop = false \wedge$ *EX-PB-VAL(ID,*$\langle v, \sigma \rangle$*)* $=$ *true* **then**
3: $stop \leftarrow true$
4: $\rho_i \leftarrow$ *share-sign*$_i(\langle ID, v \rangle)$
5: **deliver** $\langle v, \sigma \rangle$
6: send "ID, ACK, ρ_i" to the sender

7: **upon** PB-abandon(ID) **do**
8: $stop \leftarrow true$

9: **procedure** EX-PB-VAL(ID, $\langle v, \sigma \rangle$)
10: parse ID as $\langle \overline{ID}, step \rangle$ ▷ $\overline{ID}$ is the identification of the higher level Proposal-Promotion; step is the Provable-Broadcast instance therein
11: parse σ as $\langle key, \sigma_{in} \rangle$
12: **if** $step = 1$ **then** ▷ first instance of PB in a Proposal-Promotion
13: **if** *check-key*(v,key) **then**
14: **return** true
15: **if** $step > 1 \wedge$ *threshold-validate*$(\langle\langle \overline{ID}, step - 1\rangle, v\rangle, \sigma_{in})$ **then**
16: **return** true
17: **return** false

18: **procedure** CHECK-KEY(v, key)
19: **if** *EX-VABA-VAL*(v) = false **then** ▷ external VABA validity check
20: **return** false
21: parse key as $\langle view, \rho \rangle$
22: **if** $view \neq 1$ **then** ▷ need to validate the key
23: **if** *threshold-validate*$(\langle\langle\langle id, Leader[view], view\rangle, 1\rangle, v\rangle, \rho)$ = false **then**
24: **return** false
25: **if** $view \geq LOCK$ **then**
26: **return** true
27: **else**
28: **return** false ▷ the key was obtained in a view that is smaller than the view on which LOCK is locked

Algorithm 6 Proposal-Promotion with identification ID: Protocol for all parties.

Local variables initialization:
key = **lock** = **commit** = $\langle \perp, \perp \rangle$

1: **upon** delivery($\langle ID, step\rangle, \langle v, \langle key, \sigma_{in}\rangle\rangle$) **do**
2: **if** $step = 2$ **then**
3: **key** $\leftarrow \langle v, \sigma_{in} \rangle$
4: **if** $step = 3$ **then**
5: **lock** $\leftarrow \langle v, \sigma_{in} \rangle$
6: **if** $step = 4$ **then**
7: **commit** $\leftarrow \langle v, \sigma_{in} \rangle$

8: **upon** abandon(ID) **do**
9: **for** $step = 1, \ldots, 4$ **do**
10: *PB-abandon*$(\langle ID, step\rangle)$

11: **Upon** getKey(ID) return **key**
12: **Upon** getLock(ID) return **lock**
13: **Upon** getCommit(ID) return **commit**

Algorithm 7 Proposal-Promotion with identification ID: Protocol for a sender.

1: **upon** promote(ID,$\langle v, key\rangle$) invocation **do**
2: $\sigma_{in} \leftarrow \perp$
3: **for** $step = 1, .., 4$ **do**
4: $\sigma_{in} \leftarrow PB(\langle ID, step\rangle, \langle v, \langle key, \sigma_{in}\rangle\rangle)$
5: **return** σ_{in}

define formal properties for the Proposal-Promotion sub-protocol. Instead, we use it for exposition modularity and prove the VABA protocol by using the Provable-Broadcast properties directly. The pseudocode of Proposal-Promotion appears in Algorithms 7 and 6.

4 DISCUSSION

Our protocol addresses an open problem introduced by Cachin et al. [10] and reduces the expected word communication from $O(n^3)$ to $O(n^2)$ against an asynchronous adaptive adversary. We also show that in the standard definition of an asynchronous adaptive adversary this expected word communication is asymptotically optimal for any protocol that obtains the standard definition of termination (liveness) as defined [10, 11]. An interesting open question is related to protocols that obtain weak termination in the adaptive setting: is there a $\Omega(n^2)$ lower bound against an adaptive adversary that is required to deliver all messages sent by parties before they are corrupted? or does there exist a protocol with near linear expected word communication under this weak termination property?

ACKNOWLEDGEMENTS

We thank Christian Cachin and the anonymous reviewers for their high quality feedback.

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. Cryptology ePrint Archive, 2018.
[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. *arXiv preprint*, 2018.
[3] Bowen Alpern and Fred B Schneider. Defining liveness. Technical report, 1985.
[4] Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *PODC*, volume 98, pages 193–199, 1998.
[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 419–428, New York, NY, USA, 1998. ACM.
[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
[7] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, 1983.
[8] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.
[9] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, November 1987.

[10] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology*, 2001.
[11] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18, 2000.
[12] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild (keynote talk). In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, 2017.
[13] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, New York, NY, USA, 1993. ACM.
[14] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
[15] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
[16] Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada*, 2018.
[17] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
[18] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 148–161, New York, NY, USA, 1988. ACM.
[19] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.
[20] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220. ACM, 2003.
[21] Bruce M Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALG)*, 6(4):68, 2010.
[22] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, 2009.
[23] Valerie King and Jared Saia. Byzantine agreement in polynomial expected time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 401–410. ACM, 2013.
[24] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 1998.
[25] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theor. Comput. Sci.*, 645:1–24, 2016.
[26] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018. https://eprint.iacr.org/2018/235.
[27] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, 2016.
[28] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $o(1)$ expected time. *Journal of the ACM (JACM)*, 62(4):31, 2015.
[29] Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54, 2017.
[30] Gil Neiger. Distributed consensus revisited. *Information processing letters*, 49(4):195–201, 1994.
[31] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.
[32] Michael K Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, 1994.
[33] Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000.
[34] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain. *PODC*, 2019.

Appendix A PROVABLE-BROADCAST PROPERTIES

Provable-Broadcast with identification ID satisfies the following properties except with negligible probability:

- **PB-Integrity:** An honest party delivers a message at most once.
- **PB-Validity:** If an honest party p_i delivers m, then *EX-PB-VAL*$_i(id, m) = true$.
- **PB-Abandon-ability:** An honest party does not deliver any message after it invokes *PB-abandon*(ID).
- **PB-Provability:** For all v, v', if the sender can produce strings σ, σ' s.t. *threshold-validate*$(\langle id, v\rangle, \sigma) = true$ and *threshold-validate*$(\langle id, v'\rangle, \sigma') = true$, then (1) $v = v'$ and (2) $f + 1$ honest parties delivered a message m s.t. $m.v = v$.
- **PB-Termination:** If the sender is honest, no honest party invokes *PB-abandon*(ID), all messages among honest parties arrive, and the message m that is being broadcast is externally valid, then (1) all honest parties deliver m, and (2) *PB*(ID,m) returns (to the sender) σ, which satisfies *threshold-validate*$(\langle ID, m.v\rangle, \sigma) = true$.

Note that an Provable-Broadcast does not guarantee any agreement property, however, when several instances are combined together (e.g., Proposal-Promotion) strong and useful properties can be guaranteed.

Appendix B LEADER-ELECTION: PROPERTIES AND PSEUDOCODE

The Leader-Election primitive provides one operation to elect a unique party (called a leader) among the parties. An instance of a leader election primitive is identified via an identification *ID*, and exposes an operation *elect*(ID) to all parties, which returns a party $p \in \Pi$. Formal definitions are given below and the pseudocode appears in Algorithm 8.

A protocol for leader election associated with id *id* satisfies the following properties except with negligible probability.

- **Termination:** If $f + 1$ honest parties invoke *elect*(), and all messages among honest parties arrive, then all invocations by honest parties return.
- **Agreement:** All invocations of *elect*(id) by honest parties return the same party.
- **Validity:** If an invocation of *elect*(id) by an honest party returns, it returns a party p with probability $1/|\Pi|$ for every $p \in \Pi$.
- **Unpredictability:** The probability of the adversary to predict the returned value of *elect*(id) invocation by an honest party before it returns is at most $1/|\Pi| + \epsilon(k)$, where $\epsilon(k)$ is a negligible function.

Algorithm 8 Leader election. Protocol for party p_i

Local variables initialization:
1: $\Sigma \leftarrow \{\}$

2: **upon** *elect*(id) **do**
3: $\quad \rho_i \leftarrow$ *coin-share*$_i(ID)$
4: $\quad$ send "SHARE, ID, ρ_i" to all parties
5: $\quad$ **wait** until $|\Sigma| = f + 1$
6: $\quad$ **return** *coin-toss*(ID, Σ)

7: **upon receiving** "SHARE, ID, ρ_j" from p_j for the first time **do**
8: $\quad$ **if** *coin-share-validate*$(id, j, \rho_j) = true$ **then**
9: $\quad\quad \Sigma \leftarrow \Sigma \cup \{\rho_j\}$