Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience

Ittai Abraham[1], Srinivas Devadas[2], Danny Dolev[3], Kartik Nayak[1,4], and Ling Ren[1,5]

[1] VMware Research – {iabraham,nkartik,lingren}@vmware.com
[2] MIT – devadas@mit.edu
[3] Hebrew University of Jerusalem – danny.dolev@mail.huji.ac.il
[4] Duke University
[5] University of Illinois at Urbana-Champaign

Abstract. We present new protocols for Byzantine agreement in the synchronous and authenticated setting, tolerating the optimal number of f faults among $n = 2f + 1$ parties. Our protocols achieve an expected $O(1)$ round complexity and an expected $O(n^2)$ communication complexity. The exact round complexity in expectation is 10 for a static adversary and 16 for a strongly rushing adaptive adversary. For comparison, previous protocols in the same setting require expected 29 rounds.

1 Introduction

Byzantine agreement [24] is a fundamental problem in distributed computing and cryptography. It has been used to build fault tolerant distributed systems [5, 9, 22, 33], secure multi-party computation [7, 17], and more recently cryptocurrencies [4, 21, 28, 29]. In Byzantine agreement, a group n parties, each holding an initial input value, hope to commit on a common value; up to f parties can have Byzantine faults and deviate from the protocol arbitrarily. In a closely related problem called Byzantine broadcast, instead of each party holding an input value, there is one designated *sender* who tries to broadcast a value. To rule out trivial solutions, both problems have additional validity requirements.

Byzantine agreement and Byzantine broadcast have been studied under various combinations of assumptions, most notably timing assumptions – synchrony, asynchrony or partial synchrony, and setup assumptions – cryptography and public-key infrastructure (PKI). It is now well understood that these assumptions drastically affect the fault tolerance bounds. In particular, Byzantine broadcast and Byzantine agreement both require $f < n/3$ under partial synchrony or asynchrony. But under synchrony with digital signatures and PKI, Byzantine agreement can be solved with $f < n/2$ while Byzantine broadcast can be solved with $f < n - 1$.

In this paper, we consider Byzantine agreement in the synchronous and authenticated (i.e., assuming digital signatures and PKI) setting. The efficiency metrics we consider are (1) round complexity, i.e., the number of rounds of communication before the protocol terminates, and (2) communication complexity, i.e., the amount of information exchanged between parties during the protocol. For convenience, we measure communication complexity using the number of signatures exchanged between parties. Assuming each signature has λ bits, multiplying our communication complexity by λ yields the asymptotic communication complexity in bits.

In the synchronous and authenticated setting, Dolev and Strong gave a deterministic Byzantine broadcast protocol for $f < n - 1$ [12]. Their protocol achieves $f + 1$ round complexity and $O(n^2 f)$ communication complexity. The $f + 1$ round complexity matches the lower bound for deterministic protocols [12, 15]. To further improve round complexity, randomized protocols have been introduced [6, 14, 16, 31]. The most efficient

A preliminary draft of the paper appear on ePrint in 2017 [2]. The current version improves and subsumes the Byzantine agreement part of the preliminary draft.

protocol to our knowledge is proposed by Katz and Koo [19], which solves Byzantine agreement for $f < n/2$ in expected 29 rounds. [6]

In this work, we improve communication complexity to expected $O(n^2)$ and round complexity to expected 16. Our protocols use threshold signatures [8, 32] to reduce communication complexity and a random leader election subroutine to reduce round complexity. The random leader election subroutine can be constructed using common-coin protocols, and there exist constructions with a single round and $O(n^2)$ communication in the literature [8, 27]. The protocol by Cachin et al. [8] is secure against a static adversary whereas the protocol by Loss and Moran [27] is secure against an adaptive adversary. With these, we achieve the following result.

Theorem 1. *Synchronous authenticated Byzantine agreement can be solved for $f < n/2$ with*

– *expected 10 rounds and expected $O(n^2)$ communication against a static adversary assuming a single-round common-coin protocol,*
– *expected 16 rounds and expected $O(n^2)$ communication against a strongly rushing adaptive adversary assuming an adaptively secure single-round common-coin protocol.*

It is worth noting that our protocols work even in the presence of a very powerful adversary, which we call a *strongly rushing adaptive adversary*. The adversary can adaptively decide which f parties to corrupt and when to corrupt them. And by "strongly rushing", we mean that if the adversary decides to corrupt a party h after observing messages sent from h to any other party in round r, it can remove h's round-r messages from the network before they reach other honest parties. In comparison, a standard rushing adversary can decide its own round-r messages after learning honest parties' round-r messages, but if it corrupts h in round r, it cannot "take back" or alter h's round-r messages to other parties. The Dolev-Strong and Katz-Koo protocols also work against such a strongly rushing adaptive adversary.

The $O(1)$ expected round complexity is clearly asymptotically optimal. A natural question is whether or not the expected quadratic communication can be further improved. In a follow-up work [1], building on a work by Dolev and Reischuk [11], we show that $\Omega(f^2)$ expected messages are necessary against a *strongly rushing* adaptive adversary. King-Saia [20] and our follow-up work [1] solve Byzantine agreement using sub-quadratic communication. Not surprisingly, these protocols work against a standard rushing adaptive adversary but not a strongly rushing adaptive one.

1.1 Technical Overview

We first describe our core protocol, which ensures agreement (referred to as safety for the rest of the paper) and termination as required by Byzantine broadcast/agreement, but provides a weak notion of validity. Specifically, it achieves

– **Termination:** all honest parties eventually commit,
– **Agreement/safety:** all honest parties commit on the same value, and
– **Validity:** if all honest parties start with certificates for the same value v, and no Byzantine party starts with a certificate for a contradictory value, then all honest parties commit on v.

In Section 4 we will describe how to obtain these certificates to solve Byzantine broadcast or Byzantine agreement.

The core protocol runs in iterations. In each iteration, a unique leader is elected. Each new leader picks up the state left by previous leaders and proposes a value in its iteration. Parties then cast votes on the leader's value v. In more detail, each iteration consists of 4 rounds. The first three rounds are conceptually similar to Paxos and PBFT: (1) the leader learns the states of the system, (2) the leader proposes a value, and (3) parties vote on the value. If a party receives $f + 1$ votes for the same value and does not detect leader

[6] Katz and Koo [19] did not analyze communication complexity in their paper. Based on our understanding, their unrolled protocol in the appendix can achieve $O(n^2)$ communication complexity by similarly incorporating threshold signatures and a quadratic common-coin protocol.

equivocation, it commits on that value. We then add another round: (4) if a party commits, it notifies all other parties about the commit; upon receiving a notification, other parties *accept* the committed value and will vouch for that value to future leaders.

Ideally, if the leader is honest, all honest parties commit v upon receiving $f+1$ votes for v at the end of that iteration. A Byzantine leader can easily waste its iteration by not proposing. But it can also perform the following more subtle attacks: (1) send contradicting proposals to different honest parties, or (2) send a proposal to some but not all honest parties. We must ensure these Byzantine behaviors do not violate safety.

The need for equivocation checks. To ensure safety in the first attack, parties engage in an all-to-all round of communication to forward the leader's proposal to each other for an equivocation check. If a party detects leader equivocation, i.e., sees two conflicting signed proposals from the leader, it does not commit even if it receives $f+1$ votes.

The need for a notify round. Using the second attack, a Byzantine leader can make some, but not all, honest parties commit on a value v. If the other honest parties do not know that v has been committed, they may commit $v' \neq v$ in a subsequent iteration. Therefore, whenever an honest party h commits on a value v, h needs to *notify* all other honest parties of its commit. h can do this by broadcasting the $f+1$ votes it received. When another party h' receives such a notification, it "accepts" the value v. If a party has accepted v and receives a proposal $v' \neq v$ in a later iteration, it will not vote for v' unless it is shown a proof that voting for v' is safe. The details can be found in Section 3.

Safety, termination, and validity. Safety is preserved because when an honest party commits, (1) no other party can commit a different value in the same iteration (due to equivocation checks), and (2) no other value can gather enough votes in subsequent iterations (due to notify by the honest party). Validity follows from a similar argument: if all honest parties start the protocol with the same certified (i.e., accepted) value v and Byzantine parties do not have a different certified value, only v can gather enough votes. Termination is achieved when some honest party h receives $f+1$ notify messages. At this point, h sends these $f+1$ notifications to all other parties and terminates. The $f+1$ notifications h sends will ensure termination of all other parties in the next round. If an honest leader emerges, all parties terminate in its iteration.

Round complexity and communication complexity. Since there are $f+1$ honest out of $2f+1$ parties, by electing a random leader in every iteration, the protocol terminates in 2 iterations in expectation. Depending on the adversarial model, each iteration ranges from 4 to 7 rounds. Each round uses $O(n^2)$ messages (all-to-all) and each message is either a single signature or a single $(f+1)$-out-of-n threshold signature. Thus, the protocol runs in expected $O(1)$ rounds and uses expected $O(n^2)$ communication.

Paxos, PBFT, XPaxos, and our protocol. Abstractly, this core protocol resembles the synod algorithm in Paxos [23] but is adapted to the synchronous and Byzantine setting. The main idea of the synod algorithm is to ensure *quorum* intersection [23] at one *honest* party. The core idea of Paxos is to form a quorum of size $f+1$ before committing a value. With $n = 2f+1$, two quorums always intersect at one party, which is honest in Paxos. This honest party in the intersection will force a future leader to respect the committed value. In order to tolerate f Byzantine faults, PBFT [9] uses quorums of size $2f+1$ out of $n = 3f+1$, so that two quorums intersect at $f+1$ parties, among which one is guaranteed to be honest. Similar to PBFT, we also need to ensure quorum intersection at $f+1$ parties. But this requires new techniques with $n = 2f+1$ parties in total. On the one hand, an intersection of size $f+1$ seems to require quorums of size $1.5f+1$. (An subsequent work called Thunderella [30] uses this quorums size to improve the optimistic case.) On the other hand, a quorum size larger than $f+1$ (the number of honest parties) seems to require participation from Byzantine parties and thus loses liveness. As described in the core protocol, our synchronous *notify* round forms a *post-commit quorum* of size $2f+1$, which intersects with any *pre-commit quorum* of size $f+1$ at $f+1$ parties. This satisfies the requirement of one honest party in the intersection. Moreover, since parties in the post-commit quorum only receive messages, liveness is not affected.

Our protocol also shares some similarity to XPaxos [26]. In XPaxos, a view-change involves changing a set of $f+1$ active replicas (instead of only changing the leader). So far as all the active replicas in the old view notify all the active replicas in the new view, there will be one honest replica in the new view that can carry state across views. However, XPaxos makes progress only if all $f+1$ active replicas are honest. In comparison, our protocol only requires the leader to be honest to make progress.

Achieving Byzantine broadcast and Byzantine agreement. The core protocol already ensures safety and termination, so we only need some technique to boost its weaker validity to what Byzantine broadcast/agreement require. Our protocol achieves this using a single round of all-to-all communication before invoking the protocol. This allows us to avoid the standard transformation of composing n parallel Byzantine broadcasts to achieve Byzantine agreement. As a result, our Byzantine agreement protocol has the same asymptotic round/communication complexity as the core protocol.

2 Model

We assume synchrony. If an honest party i sends a message to another honest party j at the beginning of a round, the message is guaranteed to reach by the end of that round. We describe the protocol assuming lock-step execution, i.e., parties enter and exit each round simultaneously. Later in Section **??**, we will present a clock synchronization protocol to bootstrap lock-step execution from bounded message delay.

We assume digital signatures and trusted setup. In the trusted setup phase, a trusted dealer generates public/private key pairs for digital signatures and other cryptographic primitives for each party, and certifies each party's public keys. We use $\langle x \rangle_i$ to denote a message x signed by party i, i.e., $\langle x \rangle_i = (x, \sigma)$ where σ is a signature of message x produced by party i using its private signing key. For efficiency, it is customary to sign the hash digest of a message. A message can be signed by multiple parties (or the same party) in layers, i.e., $\langle \langle x \rangle_i \rangle_j = \langle x, \sigma_i \rangle_j = (x, \sigma_i, \sigma_j)$ where σ_i is a signature of x and σ_j is a signature of $x \mid\mid \sigma_i$ ($\mid\mid$ denotes concatenation). When the context is clear, we omit the signer and simply write $\langle x \rangle$ or $\langle \langle x \rangle \rangle$.

We require a random leader election subroutine. As mentioned, this subroutine can be instantiated using common-coin protocols [8, 27] or verifiable random functions [28]. It may also be left to higher level protocols. For example, a cryptocurrency may elect leaders based on proof of work.

We assume a strongly rushing adaptive adversary. After the trusted setup phase, the adversary can adaptively decide which f parties to corrupt and when to corrupt each of them as the protocol executes. Note, however, that the adversary is not *mobile*: it cannot un-corrupt a Byzantine party to restore its corruption budget. The adversary is also strongly rushing. In each round, the adversary observes any party i's message to any other party j. If the adversary decides to corrupt i at this point, it controls which other honest parties (if any) i sends messages to and what messages i sends them *in that round.*

3 A Synchronous Byzantine Synod Protocol

3.1 Core Protocol

Our core protocol is a synchronous Byzantine synod protocol with $n = 2f+1$ parties. The goal of the core synod protocol is to guarantee that all honest parties eventually commit (termination) on the same value (agreement). In addition, it achieves the following notion of validity: if (1) all honest parties start with the same value and have a *certificate* for this value, and (2) the adversary does not start with a certificate for a contradictory value, then all honest parties commit on this value. In Section 4, we show how to obtain these certificates using a single pre-round to achieve Byzantine broadcast and Byzantine agreement. For ease of exposition, we will temporarily assume a static adversary in Section 3.1 while presenting the core protocol. A static adversary has to decide which parties to corrupt after the trusted setup phase and before the protocol starts.

We now describe the protocol in detail. When a leader proposes a value v in iteration k, we say the proposal has rank k and write them as a tuple (v, k). The first iteration has $k = 1$. Each party i internally maintains states $\mathsf{accepted}_i = (v_i, k_i, \mathcal{C}_i)$ across iterations to record its *accepted* proposal. Initially, each party i initializes $\mathsf{accepted}_i := (\bot, 0, \bot)$. If party i later accepts (v, k), it sets $\mathsf{accepted}_i := (v, k, \mathcal{C})$ such that $\mathcal{C}$ *certifies* that v is legally accepted in iteration k. $\mathcal{C}$ consists of $f+1$ commit requests for proposal (v, k) (see the protocol for details). We also say $\mathcal{C}$ certifies, or is a certificate for, (v, k). Proposals are ranked by the iteration number in which they are made. Namely, (v, k) is ranked higher than, lower than, or equal to (v', k') if $k > k'$, $k < k'$ and $k = k'$, respectively. Certificates are ranked by the proposals they certify. When we say a party "broadcasts" a message, we mean it sends the message to all parties including itself.

Round 0 (elect) All parties participate in the threshold coin-tossing scheme from [8]. Their scheme costs a single round and outputs a random string to all parties. The random string modulo n defines a random leader L_k for the current iteration k. We henceforth write L_k as L for simplicity.

Round 1 (status) Each party i sends a $\langle k, \mathsf{status}, v_i, k_i, \mathcal{C}_i \rangle_i$ message to L to report its current accepted value.
At the end of this round, if party i reports the highest certificate to L (i could be L itself), L sets $\mathsf{accepted}_L = (v_L, k_L, \mathcal{C}_L) := (v_i, k_i, \mathcal{C}_i)$. If no party reports a certificate, L chooses v_L freely and sets $k_L := 0$ and $\mathcal{C}_L := \perp$.

Round 2 (propose) L broadcasts a signed proposal $\langle \langle k, \mathsf{propose}, v_L \rangle_L, k_L, \mathcal{C}_L \rangle_L$.
At the end of this round, party i sets $v_{L \to i} := v_L$ if the certificate it receives in the above leader proposal is no lower than what i reported to the leader, i.e., if $k_L \geq k_i$. Otherwise (leader is faulty), it sets $v_{L \to i} := \perp$.

Round 3 (commit) If $v_{L \to i} \neq \perp$, then party i forwards the proposal $\langle k, \mathsf{propose}, v_{L \to i} \rangle_L$ to all other parties and broadcasts a $\langle k, \mathsf{commit}, v_{L \to i} \rangle_i$ request.
At the end of this round, if party i is forwarded a properly signed proposal $\langle k, \mathsf{propose}, v' \rangle_L$ in which $v' \neq v_{L \to i}$, it does not commit in this iteration (leader has equivocated). Else, if party i receives $f + 1$ $\langle k, \mathsf{commit}, v \rangle_j$ requests in all of which $v = v_{L \to i}$, it commits on v and sets its internal state $\mathcal{C}_i$ to be these $f + 1$ commit requests concatenated. In other words, party i commits if and only if it receives $f + 1$ matching commit requests and does not detect leader equivocation.

Round 4 (notify) If party i has committed on v at the end of the previous round, it sends a notification $\langle \langle \mathsf{notify}, v \rangle_i, \mathcal{C}_i \rangle_i$ to every other party.
At the end of this round, if party i receives a $\langle \langle \mathsf{notify}, v \rangle_j, \mathcal{C} \rangle_j$ message, it accepts v by setting $\mathsf{accepted}_i = (v_i, k_i, \mathcal{C}_i) := (v, k, \mathcal{C})$. If party i receives multiple valid notify messages with different values (how this can happen is explained at the end of Section 3.2), it can accept an arbitrary one. Lastly, party i increments the iteration counter k and enters the next iteration.

Early and non-simultaneous termination. At any point during the protocol, if a party gathers *notification headers* (excluding certificates) $\langle \mathsf{notify}, v \rangle$ from $f + 1$ distinct parties, it sends these $f + 1$ notification headers to all other parties and terminates. This ensures that when the first honest party terminates, all other honest parties receive $f + 1$ notification headers and terminate in the next round.

3.2 Safety, Termination, and Validity

In this section, we prove that the core protocol in Section 3.1 provides safety, termination and a weak notion of validity.

Safety. We first give some intuition to aid understanding. The scenario to consider for safety is when an honest party h commits on a value v^* in iteration k^*. We first show that Byzantine parties cannot hold a certificate for a value other than v^* in iteration k^*. Thus, all other honest parties accept v^* at the end of iteration k^* upon receiving notify from the honest party h. Thus, a value other than v^* cannot gather enough votes in iteration $k^* + 1$, and hence cannot be committed or accepted in iteration $k^* + 1$, and hence cannot gather enough votes in iteration $k^* + 2$, and so on. Safety then holds by induction.

We now formalize the above intuition by proving the following lemma about certificates: once an honest party commits, all certificates in that iteration and future iterations can only certify its committed value.

Lemma 1. *Suppose party h is the first honest party to commit and it commits on v^* in iteration k^*. If a certificate $\mathcal{C}$ for (v, k^*) exists, then $v = v^*$.*

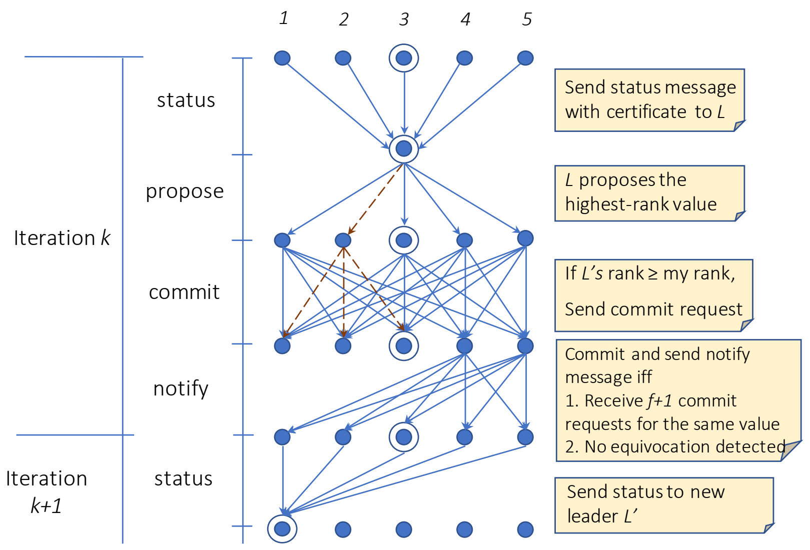
1
2
3
4
5
Iteration k
status
propose
commit
notify
Iteration k+1
status
Send status message with certificate to L
L proposes the highest-rank value
If L's rank ≥ my rank, Send commit request
Commit and send notify message iff
1. Receive f+1 commit requests for the same value
2. No equivocation detected
Send status to new leader L'

Fig. 1: An example iteration of the core protocol. In this example, $f = 2$, $n = 2f + 1 = 5$, parties 3 and 4 are Byzantine. **1.** (status) Each party sends its current states to $L = 3$. **2.** (propose) No party has committed or accepted any value, so L can propose any value of its choice. L equivocates and sends one proposal to party 4 (shown by dashed red arrow) and a different proposal to honest parties. **3.** (commit) Honest parties forward L's proposal and send commit requests to all parties. Party 4 only sends to parties $\{3, 4, 5\}$. Parties 1 and 2 receive $f + 1$ commit requests for the blue value and do not detect equivocation, so they commit. Party 5 detects leader equivocation and does not commit despite also receiving $f + 1$ commit requests for the blue value. **4.** (notify) Parties 1 and 2 notify all other parties. On receiving a valid notification, party 5 accepts the blue value. **5.** (status) The parties send status messages to the new leader $L' = 1$ for iteration $k + 1$.

Proof. $\mathcal{C}$ must consist of $f + 1$ commit requests for v. At least one of these comes from an honest party (call it h_1). Thus, h_1 must have received a proposal for v from the leader, and must have forwarded the proposal to all other parties. If $v \neq v^*$, h would have detected leader equivocation, and would not have committed on v^* in this iteration. So we have $v = v^*$.

Lemma 2. *If at the start of iteration k, (1) every honest party i has a certificate for (v, k_i), and (2) all conflicting certificates are lower ranked, i.e, any certificate for (v', k') where $v \neq v'$ must have $k' < k_i$ for all honest i, then the above two conditions will hold at the end of iteration k.*

Proof. Suppose for contradiction that some party (honest or Byzantine) acquires a *higher* certificate than what it had previously for $v' \neq v$. Then it must receive from one honest party (call it h) a $\langle k, \text{commit}, v' \rangle_h$ request in iteration k. Note that h has a certificate for (v, k_h) at the start of iteration k. In order for h to send a commit request for v', the leader L_k must show a certificate for (v', k') such that $k' \geq k_h$, which contradicts condition (2).

A simple induction shows that the above two conditions, if true at the start of an iteration, will hold true forever.

Theorem 2 (Safety). *If two honest parties commit on v and v' respectively, then $v = v'$.*

Proof. Suppose party h is the first honest party to commit, and it commits on v^* in iteration k^*. After the notify round of iteration k^*, every honest party receives a certificate for (v^*, k^*) and accepts v^*. Furthermore, due to Lemma 1, there cannot be a certificate for (v, k^*) in iteration k^* for $v \neq v^*$. Thus, the two conditions in Lemma 2 hold at the end of iteration k^*. So no certificate for a value other than v^* can be formed from this point on. In order for an honest party to commit on v, there must be a certificate for (v, k) where $k \geq k^*$. Therefore, $v = v^*$. Similarly, $v' = v^*$, and we have $v = v'$.

Termination. We now show that an honest leader will guarantee all honest parties terminate by the end of that iteration.

Theorem 3 (Termination). *If the leader L_k in iteration k is honest, then every honest party terminates one round after iteration k (or earlier).*

Proof. The honest leader L_k will send a proposal to all parties. It will propose a value reported by the highest certificate it collects in the status round. This certificate will be no lower than any certificate held by honest parties. Additionally, the unforgeability of digital signatures prevents Byzantine parties from falsely accusing L of equivocating. Therefore, all honest parties will send commit requests for v, receive $f+1$ commit requests for v, commits on v, send notification headers for v, receive $f+1$ notification headers for v (this is the end of iteration k), and terminate in the next round. (It is possible that they receive $f+1$ notification headers and terminate at any earlier time.)

Validity. We now discuss the validity achieved by our core protocol. In the theorem, we assume the existence of initial certificates for $(v, 0)$ that are input to our core protocol. These initial certificates will be provided by higher-level protocols that invoke the core protocol (c.f. Section 4).

Theorem 4 (Validity). *All honest parties will commit on v if (1) every honest party starts with an initial certificate $\mathcal{C}$ certifying v, and (2) no Byzantine party has a certificate $\mathcal{C}'$ certifying $v' \neq v$.*

Proof. The proof is straightforward from Lemma 2 and Theorem 3. The input constraints satisfy the two conditions for Lemma 2 with each $k_i = 0$. Due to Lemma 2, for all subsequent iterations, only v can have certificates and thus, only v can be committed. By Theorem 3, when an honest leader emerges, all honest parties will commit on v.

Finally, we mention an interesting scenario that does not have to be explicitly addressed in the proofs. Before any honest party commits, Byzantine parties may obtain certificates for multiple values in the same iteration. In particular, the Byzantine leader proposes two values v and v' to all the f Byzantine parties. (An example with more than two values is similar.) Byzantine parties then exchange f commit requests for both values among them. Additionally, the Byzantine leader proposes v and v' to different honest parties. Now with one more commit request for each value from honest parties, Byzantine parties can obtain certificates for both v and v', and can make honest parties accept different values by showing them different certificates (notify messages). However, this will not lead to a safety violation because no honest party would have committed in this iteration: the leader has equivocated to honest parties, so all honest parties will detect equivocation from forwarded proposals and thus refuse to commit. This scenario showcases the necessity of both the synchrony assumption and the use of digital signatures for our protocol. Lacking either one, equivocation cannot be reliably detected and any protocol will be subject to the $f < n/3$ bound. For completeness, we note that the above scenario will not lead to a violation of the termination property, either. At the end of the iteration, honest parties may accept either value. But in the next iteration, they can still vote for either value despite having accepted the other, since the two values have the same rank.

3.3 Random Leader Election against an Adaptive Adversary

The protocol presented so far does not achieve expected constant rounds against an adaptive adversary. The adversary learns who the leader L is after the elect round in an iteration. It can then immediately corrupt L and prevent it from sending any proposal. This way, the adversary forces the protocol to run for f iterations.

A first modification towards adaptive security is to move the elect round after the propose round and before the commit round. The hope is that, by the time L is corrupted, all honest parties have already received its proposal. This means every party should act as a potential leader before L_k is revealed, i.e., in status and propose rounds to collect status and make a proposal. From the commit round onward, only L's proposal is relevant.

However, this idea alone is not sufficient. At the end of the elect round, after learning the identity of L, the adversary corrupts L, signs an equivocating proposal using L's secret key and forwards it to all honest parties. Honest parties will detect equivocation from L and will not commit in this iteration. We are again forced to run the protocol for f iterations.

To this end, we need to add a step for each party to "prepare" its proposal before the leader is revealed. Afterwards, only "prepared" proposals are considered in equivocation checking. The prepare step should guarantee that, if a party h is honest throughout the prepare process but becomes corrupted afterwards, an adversary cannot construct a "prepared" equivocating proposal on h's behalf. We achieve the prepare step in two rounds as follows.

Round P1 ($\mathsf{prepare}_1$) Each party i broadcasts its proposal $\langle v_i, k\rangle_i$.

Round P2 ($\mathsf{prepare}_2$) If party j receives a proposal $\langle v_i, k\rangle_i$ from party i in the previous round, party j signs the proposal and sends $\langle v_i, k\rangle_j$ back to party i.

We say a proposal (v_i, k) is prepared if it carries $f+1$ signatures from distinct parties. Each honest party will be able to prepare its proposal. If party i is honest in the two prepare rounds and becomes corrupted only afterwards, preparing a conflicting proposal on party i's behalf requires forging at least one honest party's signature, which a computationally bounded adversary cannot do.

The core protocol against a strongly rushing adaptive adversary now has 7 rounds: status, $\mathsf{prepare}_1$, $\mathsf{prepare}_2$, propose, elect, commit, and notify. Proofs for safety and validity remain unchanged from the static case. Proof of termination and round complexity analysis also hold once we observe that (1) there is a $> 1/2$ chance that each leader L_k is honest up to the point at which it is revealed, (2) if L_k is still honest by the end of the propose round of iteration k, all honest parties will consider its proposal valid and terminate one round after iteration k.

We remark that leader election based on verifiable random function [28], when combined with our prepare rounds, achieves expected 2 iterations against a (normal) rushing adaptive adversary. But it will run into f iterations against a strongly rushing adaptive adversary, who can prevent a leader from announcing its rank after receiving it.

3.4 Round Complexity and Communication Complexity

The first honest leader will ensure termination. The random leader election subroutine ensures a $(f+1)/(2f+1) > 1/2$ probability that each leader is honest, so the core protocol terminates in expected 2 iterations, plus one extra round to forward $f+1$ notify. Thus, if an iteration requires r rounds, our core protocol requires $2r+1$ rounds to terminate in expectation. If the adversary is adaptive and strongly rushing, each iteration requires $r = 7$ rounds. If the adversary is adaptive and normal rushing, the elect round can happen in parallel to propose, and each iteration has $r = 6$ rounds. If the adversary is static (rushing or otherwise), we do not need the two prepare rounds, and the elect round can happen in parallel to either status or propose, giving $r = 4$ rounds per iteration.

Next, we analyze the communication complexity. We will show that each round consumes $O(n^2)$ communication. Hence, the core protocol requires expected $O(n^2)$ communication (whether the adversary is adaptive or static, rushing or not). First of all, note that although a certificate consists of $f+1$ signatures, its size can be reduced to a single signature using threshold signatures [8, 18, 25, 32].

1. In the status round, every party is reporting its currently accepted certificate to every other party (every party can potentially be the leader since the leader identity has not been revealed).
2. In $\mathsf{prepare}_1$, every party sends a signed proposal, which is $O(1)$ in size, to every other party.
3. In $\mathsf{prepare}_2$, every party sends back a doubly signed proposal, which is $O(1)$ in size, to every other party.
4. In the propose round, every party sends a proposal, which carries a certificate, to every other party. (A proposal need not contain status messages, following the suggestion of the HotStuff protocol [3]).
5. In the elect round, the common-coin protocol by Loss and Moran [27] requires $O(n^2)$ communication.
6. In the commit round, every party sends an $O(1)$-sized commit message to every other party.
7. In the notify round, every party sends a notify message, which carries a certificate, to every other party.
8. Lastly, before termination, every party sends $f+1$ notification headers $\langle \mathsf{notify}, v\rangle$, which can be reduced to a single threshold signature, to every other party.

4 Byzantine Broadcast and Agreement

In this section, we describe how to use the core protocol to solve synchronous authenticated Byzantine broadcast and agreement for the $f < n/2$ case. For both problems, we design a "pre-round" to let honest parties obtain initial certificates and then invoke the core protocol.

Byzantine broadcast. In Byzantine broadcast, a designated *sender* tries to broadcast a value to n parties. A solution needs to satisfy three requirements:

(termination) all honest parties eventually commit,
(agreement) all honest parties commit on the same value, and
(validity) if the sender is honest, then all honest parties commit on the value it broadcasts.

Let L_s be the designated sender. In the pre-round, L_s broadcasts a signed value $\langle v_s \rangle_{L_s}$ to every party. Such a signed value by the sender is an initial certificate certifying $(v_s, 0)$. We then invoke the core protocol. Safety and termination are satisfied due to Theorems 2 and 3. If the designated sender is honest, each honest party has a certificate for $(v_s, 0)$ and no conflicting initial certificate can exist, satisfying the condition for Theorem 4. Thus, validity is satisfied.

Byzantine agreement. In Byzantine agreement, every party holds an initial input value. A solution needs to satisfy the same termination and agreement requirements as in Byzantine broadcast. There exist a few different validity notions. We adopt a common one known as strong unanimity [13]:

(validity) if all honest parties hold the same input value v, then they all commit on v.

In the pre-round, every party i broadcasts its value $\langle v_i \rangle_i$. $f + 1$ signatures from distinct parties for the same value v form an initial certificate for $(v, 0)$. We then invoke the core protocol. Safety and termination are satisfied due to Theorems 2 and 3. If all honest parties have the same input value, then they will have an initial certificate for v and no conflicting initial certificate can exist, satisfying the condition for Theorem 4. Thus, validity is satisfied.

The efficiency of the protocols is straightforward given the analysis of the core protocol. Both protocols require one more round than the core protocol and the same $O(n^2)$ communication complexity as the core protocol.

5 Clock Synchronization

An important question is how practical the synchrony assumption is, which will be the topic of this section. The synchrony assumption essentially states that all honest replicas' messages arrive in time. This requires two properties: (i) a bounded message delay and (ii) locked step execution, i.e., honest replicas enter each round roughly at the same time. The second property is important because, if replica i enters a round much earlier than replica j, then i may end up finishing the round too soon without waiting for j's message to arrive. In our protocol, for example, this could prevent i from detecting leader equivocation and result in a safety violation.

The XFT paper provided some justification for the bounded message delay assumption in certain applications [26]. But we still need a mechanism to enforce locked step execution. To this end, we will use the following clock synchronization protocol, which may be interesting outside Byzantine agreement. It is a variation of the clock synchronization protocol by Dolev et al. [10]. The key change is to have parties sign independently in parallel (as opposed to sequentially) to facilitate the use of threshold signatures.

The protocol will be executed at known time intervals. We call each interval a "day".

Round 0 (sync) When party i's clock reaches the beginning of day X, it sends a $\langle \mathsf{sync}, X \rangle_i$ message to all parties including itself.

Round 1 (new-day) The first time a party j receives $f+1$ $\langle \mathsf{sync}, X\rangle$ messages from distinct parties (either as $f+1$ separate sync messages or within a single new-day message), it
- sets its clock to the beginning of day X, and
- sends all other parties a new-day message, which is the concatenation of $f+1$ $\langle \mathsf{sync}, X\rangle$ messages from distinct parties.

The above protocol bootstraps lock-step synchrony from the message delay bound Δ and a clock drift bound. Each sync message is triggered by a party's own local clock, independent of when day X would start for other parties. The $f+1$ sync messages can be replaced with a threshold signature for better efficiency. The protocol refreshes honest parties' clock difference to at most the message delay bound Δ at the beginning of each day. The first honest party to start a new day will broadcast a new-day message, which makes all other honest parties start the new day within Δ time. Obtaining a new-day message also means at least one honest party has sent a valid sync message, ensuring that roughly one day has indeed passed since the previous day. We can then set the duration of each round to $2\Delta + \phi$ where ϕ is the maximum clock drift between two honest parties in a "day".

Acknowledgments

We thank Dahlia Malkhi and Benjamin Chan for many useful discussions.

References

1. Ittai Abraham, T-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. *arXiv preprint*, 1805.03391, 2018.
2. 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, Report 2018/1028, 2018. `https://eprint.iacr.org/2018/1028`.
3. Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. *arXiv preprint arXiv:1803.05069*, 2018.
4. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. OPODIS, 2017.
5. Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.
6. 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*, pages 27–30. ACM, 1983.
7. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.
8. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.
9. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
10. Danny Dolev, Joseph Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.
11. Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.
12. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
13. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.
14. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

15. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information processing letters*, 14(4):183–186, 1982.
16. 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.
17. Shafi Goldwasser, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proc. of the 19th Annual ACM STOC*, volume 87, pages 218–229, 1987.
18. Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint 1804.01626*, 2018.
19. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*, volume 4117, pages 445–462. Springer, 2006.
20. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM*, 58(4):18, 2011.
21. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium*, pages 279–296. USENIX Association, 2016.
22. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.
23. Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
24. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
25. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.
26. Shengyun Liu, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 485–500. USENIX Association, 2016.
27. Julian Loss and Tal Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. Cryptology ePrint Archive 2018/235, 2018.
28. Silvio Micali. Algorand: The efficient and democratic ledger. arXiv:1607.01341, 2016.
29. Rafael Pass and Elaine Shi. Feasibilities and infeasibilities for achieving responsiveness in permissionless consensus. In *International Symposium on Distributed Computing*. Springer, 2017.
30. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.
31. Michael O. Rabin. Randomized Byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE, 1983.
32. Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.
33. Lidong Zhou, Fred Schneider, and Robbert van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.