# ALGORAND AGREEMENT
## Super Fast and Partition Resilient Byzantine Agreement

Jing Chen Sergey Gorbunov Silvio Micali Georgios Vlachos
{jing, sergey, silvio, georgios@algorand.com}

April 25, 2018

Abstract

We present a simple Byzantine agreement protocol with leader election, that works under $> 2/3$ honest majority and does not rely on the participants having synchronized clocks. When honest messages are delivered within a bounded worst-case delay, agreement is reached in expected constant number of steps when the elected leader is malicious, and is reached after two steps when the elected leader is honest. Our protocol is resilient to arbitrary network partitions with unknown length, and recovers fast after the partition is resolved and bounded message delay is restored.

We will briefly discuss how the protocol applies to blockchains in a permissionless system. In particular, when an honest leader proposes a block of transactions, the first voting step happens in parallel with the block propagation. Effectively, after the block propagates, a certificate is generated in just one step of voting.

# 1 Introduction

In this short manuscript, we describe a fast Byzantine agreement protocol with leader election, which is safe even in asynchronous networks. Algorand ledger will be based on the permissionless version of this protocol.[1] In traditional Byzantine agreements, users try to agree on one of their starting values. In a Byzantine agreement with leader election, users try to agree on a value proposed by a leader.

Our protocol is simple and reaches agreement quickly when the network is not partitioned. In particular, it achieves the following desirable properties under $> \frac{2}{3}$ honest majority:

- **Fast Agreement.** When the network has bounded delay —that is, all honest messages propagate in the network within a given time bound—, an agreement is reached in a constant expected time. In particular, when the leader is honest, his proposed value is agreed upon after two steps of communication.
- **Arbitrary Partition Resilience (i.e., Asynchronous Safety).** When the network is partitioned (especially, the Adversary has complete control on message delivery and messages may be delayed arbitrarily long), our protocol ensures safety of the system so that no two honest users will finish the protocol with different outputs.

[1] For historical references, extensions, and evaluation of Algorand ledger we refer the interested readers to [1, 2, 3, 4].

- **Fast Recovery from Network Partition.** After the network recovers from a partition and restores bounded delay, an agreement is reached in constant expected time.

# 2 Preliminaries

**Cryptographic Primitives.** We shall rely on two well known cryptographic tools: a hash function $H$ modeled as a random oracle, and digital signatures. More precisely, each player $i$ has a public/secret key pair from a digital signature scheme. Each player holds her secret key privately. Public keys are known to every player in the system. We denote $i$'s signature of a string $x$ by $sig_i(x)$. To ensure signer identities and messages are retrievable from signatures, we define

$$SIG_i(x) \triangleq (i, x, sig_i(x)).$$

The signature scheme is secure under adaptive chosen message attacks and enjoy the following *uniqueness property*: for each public/secret key pair —even the maliciously generated ones— and each message $m$, there is only one string that is accepted as the signature of $m$ relative to that public key.[2]

**Temporary Simplifying Assumptions.** In this manuscript, we simplify the protocol description by making the following assumptions, that will soon be relaxed.

1. The setting is permissioned, with a fixed set of users. The set of all players is $N$, the cardinality of $N$ is $n = 3t + 1$, and the number of malicious players is $t$.

2. Each user $i$ has a private input $v_i$ at the beginning of the protocol. The set of possible inputs is denoted by $V$, and there is a special symbol $\perp \notin V$. The users try to agree on a value in $V$.

3. All players have access to a public random string $R$, which has been selected randomly and independently of the players' public keys.

**The Adversary.** The Adversary perfectly coordinates all malicious players. He learns the messages sent by honest players and then chooses the messages sent by the malicious players. However, the Adversary cannot forge honest players' signatures or break the hash function.

**From permissioned to permissionless.** The protocol described in this manuscript can be generalized to the permissionless setting as in the original Algorand protocol, where the Adversary can corrupt users adaptively and instantaneously, but cannot control more than 1/3 of the total stake in the system. An execution of the permissionless protocol corresponds to one round in the Algorand blockchain, where users agree on a block of transactions. Similarly, the original Algorand paper also describes how the random string $R$ can be generated and updated as the blockchain grows.

[2]Our protocol also works using verifiable random functions (VRFs) [5], which can be constructed under concrete complexity assumptions.

# 3 The Agreement Protocol

For simplicity, we first describe the protocol in an idealized network setting, Communication Setting 1. Next, we describe the changes that lead to a protocol that achieves the three desirable properties highlighted in our introduction in a concrete network setting, Communication Setting 2.

***Communication Setting 1.*** The players communicate, in steps, over a synchronous propagation network. Honest users send messages at the start of a step and such messages are received by all honest users by the end of the step. Moreover, all messages seen by an honest user $i$ before the start of step $s$ will be seen by all honest users by the end of step $s$, as user $i$ helps propagate those messages.

***Communication Setting 2.*** The players communicate over a propagation network. Users do not have synchronized clocks, but their individual timers have the same speed. The network may be arbitrarily partitioned for an unknown amount of time, during which the Adversary has full control on the delivery of messages. When the network is not partitioned, a message propagated by an honest user is received by all honest users within time $\lambda$. However, the Adversary fully controls the delivery orders of different messages. All messages sent by honest users during a partition are delivered to honest users after the partition is resolved, within time $c\lambda$ for some constant $c$.

## 3.1 Notions and Notations

The protocol is a 5-step loop. For conceptual clarity we describe the loop as 5-step periods. Each period has a leader, defined as follows.

**Definition 3.1. Credential:** *User $i$'s* credential $\sigma_i^p$ *for a period $p$ is* $SIG_i(R, p)$.

**Definition 3.2. Leader:** *The* leader $\ell_p$ *for period $p$ is the user* $\arg\min_{j \in N} H(SIG_j(R, p))$.

*When a user $i$* identifies his own leader *for period $p$, $\ell_{i,p}$, $i$ sets $\ell_{i,p}$ to be the user* $\arg\min_{j \in S_i} H(SIG_j(R, p))$*, where $S_i$ is the set of all users from which $i$ has received valid period-$p$ credentials.*

Our protocol will refer to three types of messages, defined below.

**Definition 3.3. Cert-vote:** *User $i$'s* cert-vote *for a value $v$ for period $p$ is the signature* $SIG_i(v, \text{“}cert\text{”}, p)$.

*We say a user $i$* cert-votes *a value $v$ for period $p$ when he propagates* $SIG_i(v, \text{“}cert\text{”}, p)$. *We say a user $i$ has* certified *for period $p$ if he has cert-voted a value $v$ for period $p$.*

**Definition 3.4. Soft-vote:** *User $i$'s* soft-vote *for a value $v$ for period $p$ is the signature* $SIG_i(v, \text{“}soft\text{”}, p)$.

*We say a user $i$* soft-votes *a value $v$ for period $p$ when he propagates* $SIG_i(v, \text{“}soft\text{”}, p)$.

**Definition 3.5. Next-vote:** *User $i$'s* next-vote *for a value $v$ for period $p$ is the signature* $SIG_i(v, \text{“}next\text{”}, p)$.

*We say a user $i$* next-votes *a value $v$ when he propagates* $SIG_i(v, \text{“}next\text{”}, p)$.

## 3.2 The Protocol in Communication Setting 1

Users start in period 1, and after step 5 of period $p$ moves to step 1 of period $p+1$. User $i$ starts with a private value $v_i$, and there is a special symbol $\perp$ different from the users' private values.

**Period $p$**

STEP 1: [Value Proposal]

- *If* $\Big(p=1\Big)$ *OR* $\Big((p\geq 2)$ *AND* ($i$ *has received* $2t+1$ *next-votes for* $\perp$ *for period* $p-1)\Big)$, *then* $i$ *proposes* $v_i$, *which he propagates together with his period* $p$ *credential;*
- *Else if* $\Big(p\geq 2\Big)$ *AND* $\Big(i$ *has received* $2t+1$ *next-votes for some value* $v\neq\perp$ *for period* $p-1\Big)$, *then* $i$ *proposes* $v$, *which he propagates together with his period* $p$ *credential.*

STEP 2: [The Filtering Step]

- *If* $\Big(p=1\Big)$ *OR* $\Big((p\geq 2)$ *AND* ($i$ *has received* $2t+1$ *next-votes for* $\perp$ *for period* $p-1)\Big)$, *then* $i$ *identifies his leader* $\ell_{i,p}$ *for period* $p$ *and soft-votes the value* $v$ *proposed by* $\ell_{i,p}$*;*
- *Else if* $\Big(p\geq 2\Big)$ *AND* $\Big(i$ *has received* $2t+1$ *next-votes for some value* $v\neq\perp$ *for period* $p-1\Big)$, *then* $i$ *soft-votes* $v$.

STEP 3: [The Certifying Step]

- *If* $i$ *sees* $2t+1$ *soft-votes for some value* $v\neq\perp$, *then* $i$ *cert-votes* $v$.

STEP 4: [The Period's First Finishing Step]

- *If* $i$ *has certified some value* $v$ *for period* $p$, *he next-votes* $v$*;*
- *Else he next-votes* $\perp$.

STEP 5: [The Period's Second Finishing Step]

- *If* $i$ *sees* $2t+1$ *soft-votes for some value* $v\neq\perp$ *for period* $p$ *and has not next-voted* $v$ *in Step 4, then* $i$ *next-votes* $v$.[a]

[a] By the end of Step 5, an honest user $i$ is guaranteed to see $2t+1$ next-votes for some value $v$, which may or may not be $\perp$. Thus Steps 1 and 2 of the next period is well defined.

In Communication Setting 1, Steps 4 and 5 can be combined into one step. We keep them separate to better align with Communication Setting 2.

**The Halting Condition**

User $i$ HALTS the moment he sees $2t+1$ cert-votes for some value $v$ for the same period $p$, and sets $v$ to be his output. Those cert-votes form a *certificate* for $v$.

## 3.3 The Protocol in Communication Setting 2

In this setting, each user $i$ keeps a timer $clock_i$ which he resets to 0 every time he starts a new period. As long as $i$ remains in the same period, $clock_i$ keeps counting. The users' individual timers do not need to be synchronized or almost synchronized. We only require they have the same speed.

The halting condition is the same as in Communication Setting 1 and is omitted from the description below.

**Period $p$**

All honest users start period 1 at the same time.[a] User $i$ starts period $p \geq 2$ the first moment he receives $2t+1$ next-votes for some value $v$ for period $p-1$, and only if he has not yet started a period $p' > p$.[b] User $i$ sets his *starting value* for period $p \geq 2$, $st_i^p$, to $v$. For $p = 1$, $st_i^1 \triangleq \perp$. The moment user $i$ starts period $p$, he finishes all previous periods and resets $clock_i$ to 0.

STEP 1: [Value Proposal] User $i$ does the following when $clock_i = 0$.

- *If* $\Big(p = 1\Big)$ *OR* $\Big((p \geq 2)$ *AND* ($i$ *has received* $2t+1$ *next-votes for* $\perp$ *for period* $p-1$)$\Big)$, *then* $i$ *proposes* $v_i$, *which he propagates together with his period* $p$ *credential;*
- *Else if* $\Big(p \geq 2\Big)$ *AND* $\Big(i$ *has received* $2t+1$ *next-votes for some value* $v \neq \perp$ *for period* $p-1\Big)$, *then* $i$ *proposes* $v$, *which he propagates together with his period* $p$ *credential.*

STEP 2: [The Filtering Step] User $i$ does the following when $clock_i = 2\lambda$.

- *If* $\Big(p = 1\Big)$ *OR* $\Big((p \geq 2)$ *AND* ($i$ *has received* $2t+1$ *next-votes for* $\perp$ *for period* $p-1$)$\Big)$, *then* $i$ *identifies his leader* $\ell_{i,p}$ *for period* $p$ *and soft-votes the value* $v$ *proposed by* $\ell_{i,p}$*;*
- *Else if* $\Big(p \geq 2\Big)$ *AND* $\Big(i$ *has received* $2t+1$ *next-votes for some value* $v \neq \perp$ *for period* $p-1\Big)$, *then* $i$ *soft-votes* $v$.

STEP 3: [The Certifying Step] User $i$ does the following when $clock_i \in (2\lambda, 4\lambda)$.

- *If* $i$ *sees* $2t+1$ *soft-votes for some value* $v \neq \perp$, *then* $i$ *cert-votes* $v$.

STEP 4: [The Period's First Finishing Step] User $i$ does the following when $clock_i = 4\lambda$.

- *If* $i$ *has certified some value* $v$ *for period* $p$, *he next-votes* $v$*;*
- *Else if* $\Big(p \geq 2\Big)$ *AND* $\Big(i$ *has seen* $2t+1$ *next-votes for* $\perp$ *for period* $p-1\Big)$, *he next-votes* $\perp$.
- *Else he next-votes his starting value* $st_i^p$.

STEP 5: [The Period's Second Finishing Step] User $i$ does the following when $clock_i \in (4\lambda, +\infty)$, until he is able to finish period $p$.

- *If* $i$ *sees* $2t+1$ *soft-votes for some value* $v \neq \perp$ *for period* $p$, *then* $i$ *next-votes* $v$.
- *If* $\Big(p \geq 2\Big)$ *AND* $\Big(i$ *sees* $2t+1$ *next-votes for* $\perp$ *for period* $p-1\Big)$ *AND* $\Big(i$ *has not certified in period* $p\Big)$, *then* $i$ *next-votes* $\perp$.

[a] Even if different users start period 1 hours apart in time, it is as if the network has been partitioned and, once all honest users have started, the protocol guarantees the three desirable properties described in the introduction.

[b] When the network is not partitioned, an honest user always goes through periods in order. During a partition and shortly after a partition is resolved, however, an honest user may see enough next-votes for a value $v'$ for a period $p'-1$ with $p' > p$ and start period $p'$, before he sees enough next-votes for $v$ for period $p-1$. In this case, he will skip period $p$.

# 4 Analysis Sketch

Below we sketch the key points for the three desired properties of our protocol in Communication Setting 2, and we will use the following definitions.

**Definition 4.1. Potential starting value for period p:** *A value $v$ that has been next-voted by $t+1$ honest users for period $p-1$.*

**Definition 4.2. Certified value for period p:** *A value $v$ that has been cert-voted by $2t+1$ users for period $p$.*

**Definition 4.3. Potentially certified value for period p:** *A value $v$ that has been cert-voted by $t+1$ honest users for period $p$.*

Note that the Adversary can turn a potentially certified value into a certified value, by adding cert-votes of the $t$ malicious users.

**Fast Agreement without Partition.**

- If the period-1 leader $\ell_1$ is honest, then every honest user $i$ identifies $\ell_1$ as his leader in Step 2, thus soft-votes the leader's proposed value $v$. As there are $2t+1$ honest users and they only soft-vote for $v$, in Step 3 every honest user sees $2t+1$ soft-votes for $v$ by time $3\lambda$, and no other value $v'$ has these many soft-votes. Thus honest users all cert-vote $v$ by time $3\lambda$. Accordingly, all honest users see $2t+1$ cert-votes for $v$ for period 1 and output $v$ by time $4\lambda$.

  Moreover, from the moment the first honest user $i$ finishes the protocol, all honest users finish within time $\lambda$, as $i$ has helped propagating the cert-votes he sees.

- If there is no certified value for period 1 (which only happens if the leader $\ell_1$ is malicious), all honest users move to period 2 by time $6\lambda$, and they move within time $\lambda$ apart.

  Indeed, if no honest user has cert-voted in Step 3, then all honest users next-vote $\perp$ (which is their starting values for period 1) in Step 4. Thus they all see these votes and move to period 2 by time $5\lambda$. (They may or may not have finished Step 5.)

  If some honest users have cert-voted in Step 3, then there exists a value $v$ which has $2t+1$ soft-votes and no other value can have these many soft-votes. Thus those honest users have all cert-voted for $v$, and there are at most $t+1$ of them (otherwise $v$ is potentially certified and the Adversary can make it certified by adding $t$ cert-votes from malicious users). Since those honest users have helped propagating the soft-votes for $v$ by time $4\lambda$, all honest users see $2t+1$ soft-votes for $v$ by time $5\lambda$. Thus they all next-vote $v$ (in Step 4 or 5) by time $5\lambda$, and all see $2t+1$ next-votes for the same value by time $6\lambda$. Note that some honest users may have next-voted for $\perp$ in Step 4 as well, thus there may also exist $2t+1$ next-votes for $\perp$.

- More generally, if there is no certified value for period $p \geq 2$, all honest users move to period $p+1$ by their own time $8\lambda$, and they move within time $\lambda$ apart.

  The extra $2\lambda$ time compared with period 1 is because the honest users start period $p$ not at the same time but within time $\lambda$ apart. The invariant remains that all honest users finish a step within time $\lambda$ apart.

  More over, there exist at most two values each of which has $2t+1$ next-votes for period $p$, and one of them is necessarily $\perp$.

- In each period $p$, the leader is honest with probability $> 2/3$. If a period $p \geq 2$ is reached and $\ell_p$ is honest, then all honest users finish the protocol in period $p$ by their own time $6\lambda$, with the same output $v \neq \perp$.

  Indeed, in period $p-1$, if there exists a certified value $v$, then $v \neq \perp$, at least $t+1$ honest users have cert-voted for $v$ and helped propagating the $2t+1$ soft-votes for $v$ by the end of their Step 3. Thus at least $t+1$ honest users next-voted $v$ in Step 4 and did not next-vote anything else in period $p-1$. The other honest users next-voted $v$ in Step 5 in period $p-1$. So there do not exist $2t+1$ next-votes for $\perp$ and all honest users move to period $p$ with starting value $v$. In period $p$, the leader $\ell_p$ proposes $v$ in Step 1 and all honest users soft-vote $v$ in Step 2. In Step 3, by their own time $4\lambda$, all honest users have cert-voted $v$. Thus all of them finish the protocol by their own time $6\lambda$, with output $v$.

  If there is no certified value in period $p-1$, then the leader $\ell_p$ may propose his private input $v_{\ell_p}$ or a value $v \neq \perp$ for which she has seen $2t+1$ next-votes from period $p-1$. In the first case, all honest users will follow the leader and soft-vote $v_{\ell_p}$; in the second case, all honest users have seen enough next-votes for $v$ and will soft-vote $v$ in Step 2. In both cases, all honest users will cert-vote the same value in Step 3 and finish the protocol with that value.

- Combining the above facts together, if the period-1 leader $\ell_1$ is malicious, then the protocol takes in expectation at most 2.5 periods and at most $16\lambda$ time. Moreover, all honest users finish within time $\lambda$ apart.

**Arbitrary Partition Resilience.** The following properties hold even during a network partition.

- For each period $p$, at most one value is certified or potentially certified.
- If a value $v$ is potentially certified for period $p$, then only $v$ can receive $2t+1$ next-votes for period $p$. Thus, the unique potential starting value for period $p+1$ is $v$.
- If a period $p$ has a unique potential starting value $v \neq \perp$, then only $v$ can be certified for period $p$. Moreover, honest users will only next-vote $v$ for period $p$, so the unique potential starting value for period $p+1$ is $v$. Inductively, any future periods $p' > p$ can only have $v$ as a potential starting value. Thus, once a value is potentially certified, it becomes the unique value that can be certified or potentially certified for any period, and no two honest users will finish the protocol with different outputs.

**Fast Recovery from Network Partition.** The following properties hold after a network partition is resolved.

- If some honest user has seen a certificate during the partition, then all honest users will receive the certificate within time $\lambda$ after the partition is resolved and they will all HALT.
- Else, let $p$ be the highest period that some honest user is working on when the partition is resolved. After time $\lambda$, all honest users will also start period $p$ as they receive $2t+1$ next-votes for period $p-1$. Soon after, all honest users will next-vote the same value $v$ (which may be $\perp$) for period $p$, and they will all start period $p+1$ within time $\lambda$ apart.
- Once all honest users start the same period $p$ within time $\lambda$ apart, we are back in the no-partition case.

# 5 Extensions

**Producing a certificate in one voting step.** When our protocol is used to implement the Algorand blockchain, the proposed values are hashes of blocks and are propagated in parallel with the actual blocks. Our protocol allows the users to soft-vote for the hashes in Step 2 without seeing the blocks. As hashes and soft-votes are short messages and propagate much faster than blocks, by the time most honest users receive the actual block $B$, they should have already received $2t+1$ soft-votes for $H(B)$ when the leader is honest. Thus most honest users cert-vote $H(B)$ the moment they receive $B$, and a certificate is produced in only one voting step after the block is propagated.

**Dynamic adversary in permissionless settings.** In a permissionless system where the Adversary can corrupt users dynamically, the (small) committees of Steps 4 and 5 may all be corrupted during a partition after sending out their next-votes, and all their messages may be pocketed by the Adversary, in which case their votes may not be propagated to all honest users after the partition is resolved. Since the next-votes are the means of moving to the next period, we introduce new steps and committees in order to make progress after a partition. In particular, in the permissionless protocol, for each period $p$ we add steps $6, 7, 8, \ldots$, where even-numbered steps are essentially copies of Step 4 and odd-numbered steps are essentially copies of Step 5. The corresponding rule for moving to period $p+1$ would be seeing $2t+1$ next-votes for some value $v$ from the same step of period $p$.

# 6 Discussions

**The Optimality of 2/3 Honest Majority.** Following the classic literature on Byzantine agreements, no agreement protocol that works under $\leq 2/3$ honest majority can be partition resilient. Thus our protocol has the optimal dependence on the honesty ratio among all partition-resilient agreement protocols.

**Short timers.** It is not necessary for a user $i$'s timer to be able to keep track of time forever. Indeed, we only need that users' individual timers can count up to a short fixed interval —the amount of time it takes a user to reach Step 5 after starting a period, when there is no partition. In practice this interval is no more than a few minutes, depending on how fast a block propagates. If a long network partition happens, the timers will all be reset shortly after the partition is resolved and the protocol still achieves the three desired properties given in the Introduction.

**Player replaceability.** Our protocol is *player-replaceable* as in the original Algorand protocol, which allows us to change the set of users that vote in each step, tolerating an Adversary who is able to corrupt users instantaneously. Indeed, in a permissionless system, we use Algorand's cryptographic self-selection to select small voting committees, and the committee members use ephemeral keys to sign their votes. We will describe the permissionless protocol and the corresponding conditions for committee selection in another manuscript.

# References


[1] Jing Chen, Silvio Micali. ALGORAND. In arXiv report http://arxiv.org/abs/1607.01341 Version 9.

[2] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos , Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In SOSP '17.

[3] Sergey Gorbunov and Silvio Micali, Democoin: A Publicly Verifiable and Jointly Serviced Cryptocurrency. In Cryptology ePrint Archive, Report 2015/521.

[4] Silvio Micali, ALGORAND: The Efficient and Democratic Ledger. In arXiv report http://arxiv.org/abs/1607.01341 Version 1.

[5] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable Random Functions. In FOCS '99.