1 Model

We consider a static set of n processes with known identities, communicating by reliable point-to-point channels, in a complete graph. Messages are uniquely identifiable.

Synchrony. The network is asynchronous. Processes may crash; at most f crashes occur.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which's invoked with the functions $\mathsf{RB\text{-}cast}(m)$ and $\mathsf{RB\text{-}received}(m)$. There exists a shared object called DenyList (DL) (defined below) that is interfaced with the functions $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$ and $\mathsf{READ}()$.

Notation. Let Π be the finite set of process identifiers and let $n \triangleq |\Pi|$. Two authorization subsets are $\Pi_M \subseteq \Pi$ (processes allowed to issue APPEND) and $\Pi_V \subseteq \Pi$ (processes allowed to issue PROVE). Indices $i, j \in \Pi$ refer to processes, and p_i denotes the process with identifier i. Let $\mathcal{M}$ denote the universe of uniquely identifiable messages, with $m \in \mathcal{M}$. Let $\mathcal{R} \subseteq \mathbb{N}$ be the set of round identifiers; we write $r \in \mathcal{R}$ for a round. We use the precedence relation $\prec$ for the DL linearization: $x \prec y$ means that operation x appears strictly before y in the linearized history of DL. For any finite set $A \subseteq \mathcal{M}$, $\mathsf{ordered}(A)$ returns a deterministic total order over A (e.g., lexicographic order on (*senderId*, *messageId*) or on message hashes). For any round $r \in \mathcal{R}$, define $\mathsf{Winners}_r \triangleq \{\, j \in \Pi \mid (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}(r) \,\}$, i.e., the set of processes whose $\mathsf{PROVE}(r)$ appears before the first $\mathsf{APPEND}(r)$ in the DL linearization. We denoted by $\mathsf{PROVE}^{(j)}(r)$ or $\mathsf{APPEND}^{(j)}(r)$ the operation $\mathsf{PROVE}(r)$ or $\mathsf{APPEND}(r)$ invoked by process j.

2 Primitives

2.1 Reliable Broadcast (RB)

RB provides the following properties in the model.

- **Integrity**: Every message received was previously sent. $\forall p_i : \mathsf{RB\text{-}received}_i(m) \Rightarrow \exists p_j : \mathsf{RB\text{-}cast}_j(m)$.
- **No-duplicates**: No message is received more than once at any process.
- **Validity**: If a correct process broadcasts m, every correct process eventually receives m.

2.2 Group Election Object

We consider a Groupe Election object ($\mathsf{GE}[r]$) per round $r \in \mathcal{R}$ with the following properties.

There are three operations: $\mathsf{CANDIDATE}(r), \mathsf{CLOSE}(r)$ and $\mathsf{READGE}(r)$ such that:

- **Termination.** A $\mathsf{CANDIDATE}(r), \mathsf{CLOSE}(r)$ or $\mathsf{READGE}(r)$ operation invoked by a correct process always returns.
- **Election.** If there exists at least one $\mathsf{CLOSE}(r)$ operation and let $\mathsf{CLOSE}(r)^\star$ denote the first $\mathsf{CLOSE}(r)$ in the linearization order. If some correct process p invokes $\mathsf{CANDIDATE}(r)$ and the invocation of $\mathsf{CANDIDATE}(r)$ appears before $\mathsf{CLOSE}(r)^\star$ in the linearization order, then $\mathsf{Winners}_r \neq \emptyset$.

- **Prefix Inclusion.** If $\mathsf{CLOSE}(r)^\star$ exists, then there exists a set $\mathsf{Winners}_r \subseteq \Pi$ such that, for any process p_j: $p_j \in \mathsf{Winners}_r$ iff p_j invokes $\mathsf{CANDIDATE}(r)$ and its $\mathsf{CANDIDATE}(r)$ operation is linearized before $\mathsf{CLOSE}(r)^\star$.
- **Stability.** If $\mathsf{CLOSE}(r)^\star$ exists, then every $\mathsf{READGE}(r)$ operation linearized after $\mathsf{CLOSE}(r)^\star$ returns exactly $\mathsf{Winners}_r$.
- **READ Validity.** The invocation of $op = \mathsf{READGE}(r)$ by a process p returns the list of valid invocations of $\mathsf{CANDIDATE}(r)$ that appears before op in the linearization order along with the names of the processes that invoked each operation.

3 Group Election Object Consensus Number

Definition 1. We assume a synchronous DenyList (DL) object as in [?] with the following properties.

The DenyList object type supports three operations: APPEND, PROVE, and READ. These operations appear as if executed in a sequence Seq such that:

- **Termination.** A PROVE, an APPEND, or a READ operation invoked by a correct process always returns.
- **APPEND Validity.** The invocation of $\mathsf{APPEND}(x)$ by a process p is valid if:
  - $p \in \Pi_M \subseteq \Pi$; **and**
  - $x \in S$, where S is a predefined set.

  Otherwise, the operation is invalid.
- **PROVE Validity.** If the invocation of a $op = \mathsf{PROVE}(x)$ by a correct process p is not valid, then:
  - $p \notin \Pi_V \subseteq \Pi$; **or**
  - A valid $\mathsf{APPEND}(x)$ appears before op in Seq.

  Otherwise, the operation is valid.
- **PROVE Anti-Flickering.** If the invocation of a operation $op = \mathsf{PROVE}(x)$ by a correct process $p \in \Pi_V$ is invalid, then any $\mathsf{PROVE}(x)$ operation that appears after op in Seq is invalid.
- **READ Validity.** The invocation of $op = \mathsf{READ}()$ by a process $p \in \pi_V$ returns the list of valid invocations of PROVE that appears before op in Seq along with the names of the processes that invoked each operation. item **Anonymity.** Let us assume the process p invokes a $\mathsf{PROVE}(v)$ operation. If the process p' invokes a $\mathsf{READ}()$ operation, then p' cannot learn the value v unless p leaks additional information.

We assume that $\Pi_M = \Pi_V = \Pi$ (all processes can invoke $\mathsf{APPEND}$and PROVE).

Lemma 1 (From DenyList to Group Election)**.** *For any fixed value $r \in S$, one DenyList object can be used to wait-free implement a Group Election object* $\mathsf{GE}[r]$*.*

Proof. We implement the operations of $\mathsf{GE}[r]$ using the operations of the DenyList as follows.

```
function CANDIDATE($r$)
    PROVE($r$)
```

```
end function

function CLOSE($r$)
    APPEND($r$)
end function

function READGE($r$)
    $P \leftarrow$ READ()
    return $\{j : (j, \mathsf{prove}(r)) \in P\}$
end function
```

Termination. Termination follows from the Termination property of DenyList operations. Consider now a fixed sequential history Seq of the DenyList.

Prefix Inclusion. Let $\mathsf{APPEND}(r)^\star$ denote the first valid $\mathsf{APPEND}(r)$ in Seq, if it exists. From the PROVE Validity and anti-flickering properties of the DenyList, a process p_j has a valid $\mathsf{PROVE}(r)$ in Seq if and only if its $\mathsf{PROVE}(r)$ invocation appears before $\mathsf{APPEND}(r)^\star$ in Seq. Hence, by construction, $p_j \in \mathsf{Winners}_r$ iff p_j invokes $\mathsf{CANDIDATE}(r)$ and its $\mathsf{CANDIDATE}(r)$ is linearized before $\mathsf{CLOSE}(r)^\star$ where we identify $\mathsf{CLOSE}(r)^\star$ with $\mathsf{APPEND}(r)^\star$. This is exactly the Prefix inclusion property.

Stability. Moreover, after $\mathsf{APPEND}(r)^\star$, no new $\mathsf{PROVE}(r)$ can become valid (anti-flickering), so every subsequent $\mathsf{READ}^\star()$ returns the same set of valid $\mathsf{PROVE}(r)$ invocations. Consequently, every $\mathsf{READGE}(r)$ linearized after $\mathsf{CLOSE}(r)^\star$ returns the same set $\mathsf{Winners}_r$, which proves Stability.

Election. Finally, if some process invokes $\mathsf{CANDIDATE}(r)$ before $\mathsf{CLOSE}(r)^\star$, its proof is valid and thus appears in the set returned by $\mathsf{READ}^\star()$. Hence $\mathsf{Winners}_r \neq \emptyset$, which proves Election.

Validity. Validity is immediate from the construction of $\mathsf{Winners}_r$: a process belongs to $\mathsf{Winners}_r$ only if it invoked $\mathsf{PROVE}(r)$, i.e., only if it invoked $\mathsf{CANDIDATE}(r)$.

Thus the constructed object satisfies all properties of a Group Election object. □

Lemma 2 (From Group Election to DenyList)**.** *Fix a value $r \in S$. A Group Election object* $\mathsf{GE}[r]$ *can be used to wait-free implement an DenyList.*

Proof. We implement the DenyList for value r as follows. We use one Group Election object $\mathsf{GE}[r]$.

The operations are implemented as follows.

```
function APPEND($r$)
    CLOSE($r$)
end function

function PROVE($r$)
    CANDIDATE($r$)
    $W_r \leftarrow$ READGE($r$)
    if $p \in W_r$ then
        return True
    else
        return False
    end if
end function

function READ()
    $W \leftarrow \bigcup_{\forall r \in S} \mathsf{READGE}(r)$
    return $\{(p, r) \mid p \in W_r\}$
end function
```

Termination. Termination follows from the Termination property of Group Election operations. Consider now a fixed sequential history Seq of the Group Election.

APPEND Validity. By construction, any process invoking $\mathsf{APPEND}(r)$ invokes $\mathsf{CLOSE}(r)$. By definition of Group Election, $\mathsf{CLOSE}(r)$ is always valid.

PROVE Validity. By definition $\Pi_V = \Pi$, so any process invoking $\mathsf{PROVE}(r)$ is in Π_V. So the case $p \notin \Pi_V$ cannot happen. Now, if some process invokes $\mathsf{APPEND}(r)$ before the invocation of $\mathsf{PROVE}(r)$ in Seq, then by the Prefix Inclusion property of Group Election, the set $\mathsf{Winners}_r$ is already fixed and any subsequent $\mathsf{CANDIDATE}(r)$ cannot be in $\mathsf{Winners}_r$. Hence the invocation of $\mathsf{PROVE}(r)$ is invalid. Conversely, if no $\mathsf{APPEND}(r)$ appears before $\mathsf{PROVE}(r)$ in Seq, then by the Election property of Group Election, if some correct process invoked $\mathsf{CANDIDATE}(r)$ before any $\mathsf{CLOSE}(r)$, then $\mathsf{Winners}_r \neq \emptyset$ and hence the invoking process belongs to $\mathsf{Winners}_r$. Thus its invocation of $\mathsf{PROVE}(r)$ is valid. This proves PROVE Validity.

PROVE Anti-Flickering. If some invocation of $\mathsf{PROVE}(r)$ is invalid, then some $\mathsf{APPEND}(r)$ must have appeared before it in Seq. By the Prefix Inclusion property of Group Election, the set $\mathsf{Winners}_r$ is fixed after the first $\mathsf{CLOSE}(r)$, so any subsequent $\mathsf{CANDIDATE}(r)$ cannot be in $\mathsf{Winners}_r$. Hence any subsequent invocation of $\mathsf{PROVE}(r)$ is also invalid. This proves PROVE Anti-Flickering.

READ Validity. Finally, by construction, the invocation of $\mathsf{READ}()$ returns the list of valid invocations of PROVE that appear before it in Seq along with the names of the processes that invoked each operation. This proves READ Validity.

□

Theorem 3 (Consensus number of Group Election)**.** *Let $|\Pi_V| = k$. The Group Election object type with verifier set Π_V has consensus number k. In particular, when $\Pi_V = \Pi$, the consensus number of Group Election is $|\Pi|$.*

Proof. We first recall that, for a DenyList object with $|\Pi_V| = k$ (a k-DenyList), the consensus number is exactly k.

Lower bound. By Lemma 1, for any fixed value $r \in S$, one k-DenyList object can be used to wait-free implement a Group Election object $\mathsf{GE}[r]$ over the same set of processes. Since the k-DenyList has consensus number k, it follows that the Group Election type has consensus number at least k.

Upper bound. Conversely, by Lemma 2, one Group Election object can be used to wait-free implement a DenyList object restricted to value r. This restricted DenyList satisfies the same specification as the original k-DenyList on value r, and in particular it has consensus number k. Therefore, the consensus number of the Group Election type cannot exceed k.

Combining the two bounds, we obtain that the consensus number of the Group Election object type is exactly $k = |\Pi_V|$. When we instantiate $\Pi_V = \Pi$, we get that the consensus number of Group Election is $|\Pi|$. □

4 Target Abstraction: Atomic Reliable Broadcast (ARB)

Processes export $\mathsf{AB\text{-}broadcast}(m)$ and $\mathsf{AB\text{-}deliver}(m)$. ARB requires total order:

$$\forall m_1, m_2,\ \forall p_i, p_j : \quad \mathsf{AB\text{-}deliver}_i(m_1) < \mathsf{AB\text{-}deliver}_i(m_2) \Rightarrow \mathsf{AB\text{-}deliver}_j(m_1) < \mathsf{AB\text{-}deliver}_j(m_2),$$

plus Integrity/No-duplicates/Validity (inherited from RB and the construction).

5 ARB over RB and DL

Theorem 4 (RB + Group Election implements F-ARB)**.** *In an asynchronous message-passing system with crash failure. We can wait-free implement a FIFO-Atomic Reliable Broadcast from a Reliable Broadcast (RB) primitive and one Group Election object* $\mathsf{GE}[r]$ *per round* $r \in \mathbb{N}$.

Proof. By Theorem 3, the Group Election object type with verifier set Π_V has consensus number $|\Pi_V|$. In particular, when $\Pi_V = \Pi$, using one instance $\mathsf{GE}[r]$ per round r we can implement wait-free consensus among all processes in Π.

It is well known that, in a crash-prone asynchronous message-passing system, consensus and atomic (total order) broadcast are equivalent (defago et al): given consensus, one can implement atomic broadcast by using an infinite sequence of consensus instances to decide the sequence of messages to deliver, and conversely atomic broadcast can be used to implement consensus by deciding a single value in the first position of the total order.

In our setting, we already have a Reliable Broadcast (RB) primitive, which provides RB-Validity, RB-Agreement, and RB-Integrity for the dissemination of messages. Using the consensus power provided by the Group Election objects, we can therefore apply the standard reduction from consensus to atomic broadcast: each position (or *slot*) in the global delivery sequence is chosen by a consensus instance, whose proposals are messages that have been RB-delivered but not yet ordered. This yields an atomic reliable broadcast (ARB) primitive.

To obtain FIFO-Atomic Reliable Broadcast (F-ARB), it suffices to enforce per-sender FIFO order on top of ARB. This can be done in the usual way by tagging each message broadcast by a process p_i with a local sequence number $s \in \mathbb{N}$, and by ensuring that only the message with the smallest pending sequence number for p_i is ever proposed to a consensus instance. As a result, for every sender p_i, messages with tags (p_i, s) and (p_i, t) with $s < t$ are decided (and thus delivered) in this order at all processes.

Hence, RB plus Group Election objects is sufficient to implement FIFO-Atomic Reliable Broadcast. □

6 BFT-ARB over RB and DL

6.1 Model extension

We consider a static set of n processes with known identities, communicating by reliable point-to-point channels, in a complete graph. Messages are uniquely identifiable.

Synchrony. The network is asynchronous. Processes may crash or be byzantine; at most $f = \frac{n}{2} - 1$ processes can be faulty.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which's invoked with the functions $\mathsf{RB\text{-}cast}(m)$ and $\mathsf{RB\text{-}received}(m)$. There exists a shared object called DenyList (DL) (defined below) that is interfaced with the functions $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$ and $\mathsf{READ}()$.

Byzantine behaviour A process exhibits Byzantine behavior if it deviates arbitrarily from the specified algorithm. This includes, but is not limited to, the following actions:

- Invoking primitives ($\mathsf{RB\text{-}cast}$, APPEND, PROVE, etc.) with invalid or maliciously crafted inputs.
- Colluding with other Byzantine processes to manipulate the system's state or violate its guarantees.
- Delaying or accelerating message delivery to specific nodes to disrupt the expected timing of operations.
- Withholding messages or responses to create inconsistencies in the system's state.

Byzantine processes are constrained by the following:

- They cannot forge valid cryptographic signatures or threshold shares without the corresponding private keys.
- They cannot violate the termination, validity, or anti-flickering properties of the DL object.
- They cannot break the integrity, no-duplicates, or validity properties of the RB primitive.

Notation. Let Π be the finite set of process identifiers and let $n \triangleq |\Pi|$. Two authorization subsets are $M \subseteq \Pi$ (processes allowed to issue APPEND) and $V \subseteq \Pi$ (processes allowed to issue PROVE). Indices $i, j \in \Pi$ refer to processes, and p_i denotes the process with identifier i. Let $\mathcal{M}$ denote the universe of uniquely identifiable messages, with $m \in \mathcal{M}$. Let $\mathcal{R} \subseteq \mathbb{N}$ be the set of round identifiers; we write $r \in \mathcal{R}$ for a round. We use the precedence relation $\prec$ for the DL linearization: $x \prec y$ means that operation x appears strictly before y in the linearized history of DL. For any finite set $A \subseteq \mathcal{M}$, $\mathsf{ordered}(A)$ returns a deterministic total order over A (e.g., lexicographic order on (*senderId*, *messageId*) or on message hashes). For any round $r \in \mathcal{R}$, define $\mathsf{Winners}_r \triangleq \{\, j \in \Pi \mid (j, \mathsf{prove}(r)) \prec \mathsf{APPEND}(r) \,\}$, i.e., the set of processes whose $\mathsf{PROVE}(r)$ appears before the first $\mathsf{APPEND}(r)$ in the DL linearization. We denoted by $\mathsf{PROVE}^{(j)}(r)$ or $\mathsf{APPEND}^{(j)}(r)$ the operation $\mathsf{PROVE}(r)$ or $\mathsf{APPEND}(r)$ invoked by process j.

6.2 Primitives

6.2.1 BFT DenyList

We consider a DL object that satisfies the following properties despite the presence of up to f byzantine processes:

- **Termination.** Every operation $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$, and $\mathsf{READ}()$ invoked by a correct process eventually returns.
- **APPEND/PROVE/READ Validity.** The preconditions for invoking $\mathsf{APPEND}(x)$, $\mathsf{PROVE}(x)$, and $\mathsf{READ}()$ are respected (cf. #2.2). The return values of these operations conform to the sequential specification of the DL object.
- **APPEND Justification.** For any element x, if an operation $\mathsf{APPEND}(x)$ invoked by a correct process completes successfully, then there exists at least one valid $\mathsf{PROVE}(x)$ operation that that precedes this $\mathsf{APPEND}(x)$ in the DL linearization.
- **PROVE Anti-Flickering.** Once an element x has been appended to the DL by any process, all subsequent invocations of $\mathsf{PROVE}(x)$ by any process return “invalid”.

6.2.2 t-out-of-n Threshold Random Number Generator

We consider a function that with t out of n shares any process can reconstruct a deterministic random number. The function is defined as follows:

- **t-reconstruction.** Given any subset S of at least t valid shares derived from the same value r, there exists a unique value σ consistent with all shares in S, and σ can be efficiently reconstructed from S.
- **$(t-1)$-non-reconstructibility.** Given any subset S of at most $t-1$ valid shares derived from the same value r, there exist two distinct values σ and σ' that are both consistent with all shares in S. In particular, no algorithm that only sees the shares in S can always distinguish whether the underlying value is σ or σ'.
- **Per-process non-equivocation.** For any process p and value r, there is at most one valid share that p can derive from r. Formally, if σ and σ' are two valid shares output by process p from the same value r, then $\sigma = \sigma'$. In particular, a single process cannot emit two different valid shares for the same underlying value r.

Interface. For algorithmic purposes, we model the t-out-of-n threshold random number generator as providing the following interface to each process $p \in \Pi$.

- $\mathsf{SHARE}_{p_i}(r)$: On input a value r, run locally by process p_i, returns a valid share σ_r^i. By per-process share uniqueness, for any fixed p_i and r the value σ_r^i is uniquely determined.
- $\mathsf{COMBINE}(S)$: On a set S of at least t pairs (p_i, σ_r^i), returns the reconstructed value σ_r. By t-reconstruction, this value is well defined and independent of the particular set S of valid shares of size at least t.
- $\mathsf{VERIFY}(r, \sigma_{r'})$: On input a value r and a candidate value $\sigma_{r'}$, returns true if and only if there exists a set S of at least t valid shares for r such that $\mathsf{Combine}(r, S) = \sigma_{r'}$, and false otherwise. We say that $\sigma_{r'}$ is *valid for r* if $\mathsf{Verify}(r, \sigma_{r'}) = \mathsf{true}$.

6.3 Algorithm

6.3.1 Variables

Each process p_i maintains the following local variables:

```
current ← $0$
received ← $\emptyset$
delivered ← $\emptyset$
prop$[r][j] \leftarrow \bot, \forall r, j$
$X_r \leftarrow \bot, \forall r$
resolved$[r] \leftarrow \bot, \forall r$
```

Algorithm A AB-broadcast

```
D1  function ABCAST($m$)
D2      $S \leftarrow$ (received \ delivered) $\cup \{m\}$
D3      RB-cast$(prop, S, r, i)$
D4      wait until $|X_r| \geq f + 1$
D5      $\sigma_r \leftarrow$ COMBINE$(X_r)$
D6      PROVE$(\sigma_r)$; APPEND$(\sigma_r)$;
D7      RB-cast$(submit, S, \sigma_r, r, i)$
D8  end function
```

Algorithm B AB-deliver

```
E1  function AB-DELIVER
E2      $r \leftarrow$ current; $\sigma_r \leftarrow$ resolved$[r]$;
E3      if $\sigma_r == \bot$ then
E4          return $\bot$
E5      end if
E6      $P \leftarrow$ READ()
E7      if $\forall j : (j, prove(\sigma_r)) \notin P$ then
E8          return $\bot$
E9      end if
E10     APPEND$(\sigma_r)$; $P \leftarrow$ READ();
E11     $W_r \leftarrow \{j : (j, \text{prove}(\sigma_r)) \in P\}$
E12     if $\exists j \in W_r$ : prop$[r][j] == \bot$ then
E13         return $\bot$
E14     end if
E15     $M_r \leftarrow \bigcup_{j \in W_r}$ prop$[r][j]$;
E16     $m \leftarrow$ ordered$(M_r)[0]$
E17     delivered ← delivered $\cup \{m\}$;
E18     if $M_r \setminus$ delivered $= \emptyset$ then
E19         current ← current $+ 1$;
E20     end if
E21     return $m$
E22 end function
```

Algorithm C RBreceived handler

```
F1 function RBRCVD($prop, S_j, r_j, j$)
F2     if $r_j \geq r$ then
F3         $\mathsf{prop}[r_j][j] = S_j$
F4         $\sigma^i_{r_j} \leftarrow \mathsf{SHARE}(r_j)$
F5         $send_j(r, \sigma^i_{r_j})$
F6     end if
F7 end function
```

Algorithm D RBreceived handler

```
G1 function RBRCVD($submit, S_j, \sigma_{r_j}, r_j, j$)
G2     if $\mathsf{VERIFY}(r_j, \sigma_{r_j})$ then
G3         $\mathsf{resolved}[r_j] \leftarrow \sigma_{r_j}$
G4     end if
G5 end function
```

Algorithm E Share received handler

```
H1 function RECEIVED($r_j, \sigma^j_{r_j}, j$)
H2     if $r_j == r$ then
H3         $X_r \leftarrow X_r \cup \sigma^j_r$
H4     end if
H5 end function
```

6.4 Example execution

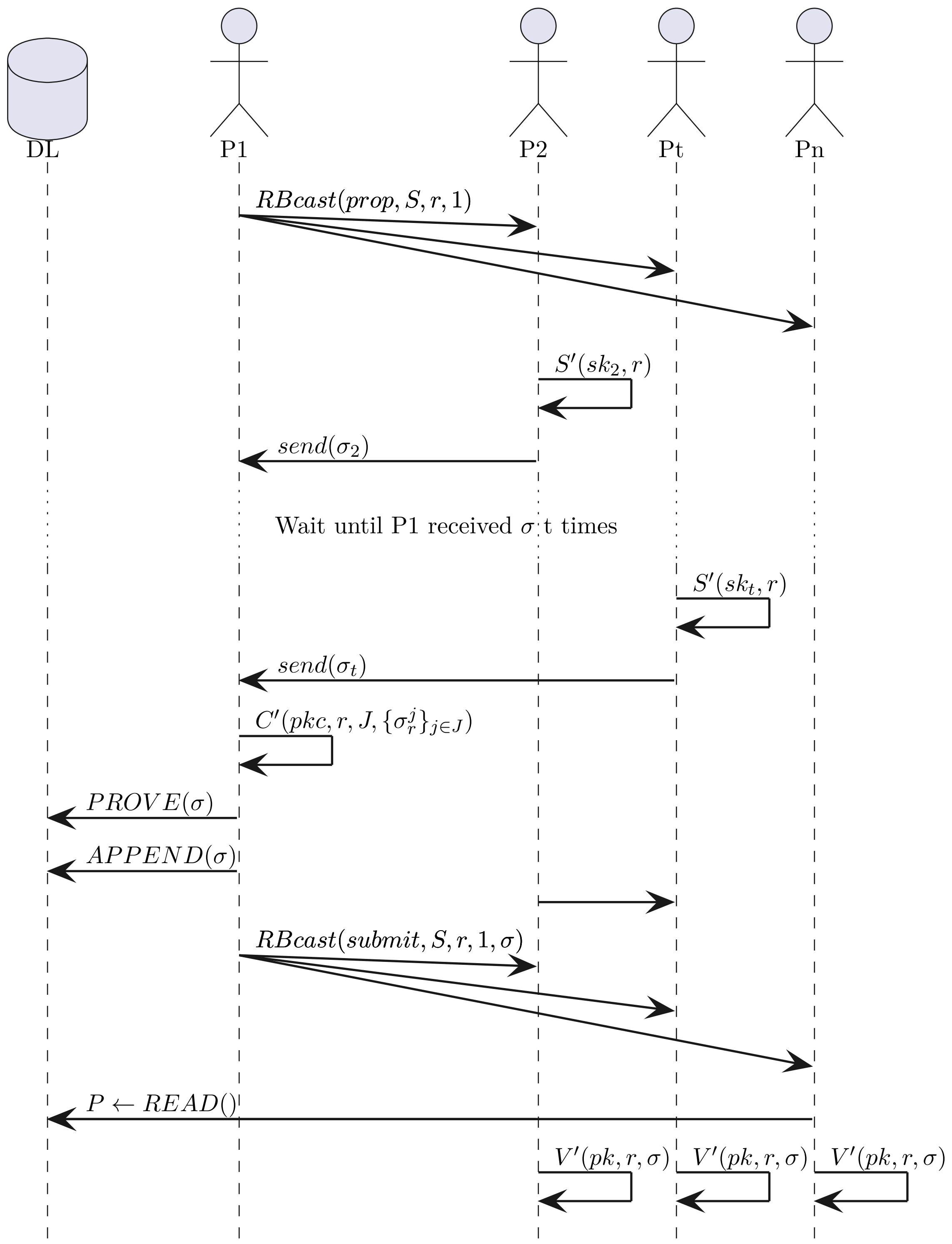

Figure 1: Expected Executions of P1 willing to send a message at round r

7 Implementation of BFT-DenyList and Threshold Cryptography

7.1 DenyList

BFT-DenyList In our algorithm we use multiple DenyList as follows:

- Let $\mathcal{DL} = \{DL_1, \dots, DL_k\}$ be the set of DenyList used by the algorithm.
- We set $k = \binom{n}{f}$.
- For each $i \in \{1, \dots, k\}$, let M_i be the set of moderators associated with DL_i according to the DenyList definition, so that $|M_i| = n - f$.
- Let $\mathcal{M} = \{M_1, \dots, M_k\}$. We require that the M_i are pairwise distinct:
$$\forall i, j \in \{1, \dots, k\},\ i \neq j \implies M_i \neq M_j.$$

Lemma 5. *$\exists M_i \in M : \forall p \in M_i$ p is correct.*

Proof. Let consider the set F of faulty processes, with $|F| = f$. We can construct the set $M_i = \Pi \setminus F$ such that $|M_i| = n - |F| = n - f$. By construction, $\forall p \in M_i$ p is correct. ☐

Lemma 6. *$\forall M_i \in M, \exists p \in M_i$ such that p is correct.*

Proof. $\forall i \in \{1, \dots, k\}, |M_i| = n - f$ with $n \geq 2f + 1$. We can say that $|M_i| \geq 2f + 1 - f = f + 1 > f$ ☐

Each process can invoke the following functions :

- $\mathsf{READ}' : () \to \mathcal{L}(\mathbb{R} \times \mathsf{prove}(\mathbb{R}))$
- $\mathsf{APPEND}' : \mathbb{R} \to ()$
- $\mathsf{PROVE}' : \mathbb{R} \to \{0, 1\}$

Such that :

Algorithm F $\mathsf{READ}'() \to \mathcal{L}(\mathbb{R} \times \mathsf{prove}(\mathbb{R}))$

```
function READ'
    $j \leftarrow$ the process invoking $\mathsf{READ}'()$
    $res \leftarrow \emptyset$
    for all $i \in \{1, \dots, k\}$ do
        $res \leftarrow res \cup DL_i.\mathsf{READ}()$
    end for
    return $res$
end function
```

Algorithm G $\mathsf{APPEND}'(\sigma) \to ()$

```
function APPEND'($\sigma$)
    $j \leftarrow$ the process invoking $\mathsf{APPEND}'(\sigma)$
    for all $M_i \in \{M_k \in M : j \in M_k\}$ do
        $DL_i.\mathsf{APPEND}(\sigma)$
    end for
end function
```

Algorithm H $\mathsf{PROVE}'(\sigma) \to \{0,1\}$

```
function PROVE'($\sigma$)
    $j \leftarrow$ the process invoking $\mathsf{PROVE}'(\sigma)$
    $flag \leftarrow false$
    for all $i \in \{1, \ldots, k\}$ do
        $flag \leftarrow flag$ OR $DL_i.\mathsf{PROVE}(\sigma)$
    end for
    return $flag$
end function
```

7.2 Threshold Cryptography

We are using the Boneh-Lynn-Shacham scheme as cryptography primitive to our threshold signature scheme. With :

- $G : \mathbb{R} \to \mathbb{R} \times \mathbb{R}$
- $S : \mathbb{R} \times \mathcal{R} \to \mathbb{R}$
- $V : \mathbb{R} \times \mathcal{R} \times \mathbb{R} \to \{0,1\}$

Such that :

- $G(x) \to (pk, sk)$: where x is a random value such that $\nexists x_1, x_2 : x_1 \neq x_2, G(x_1) = G(x_2)$
- $S(sk, m) \to \sigma_m$
- $V(pk, m_1, \sigma_{m_2}) \to k$: with $k = 1$ iff $m_1 == m_2$ and $\exists x \in \mathbb{R}$ such that $G(x) \to (pk, sk)$; otherwise $k = 0$

threshold Scheme In our algorithm we are only using the following functions :

- $G' : \mathbb{R} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R} \times (\mathbb{R} \times \mathbb{R})^n$: with $n \triangleq |\Pi|$
- $S' : \mathbb{R} \times \mathcal{R} \to \mathbb{R}$
- $C' : \mathbb{R}^n \times \mathcal{R} \times \mathbb{R} \times \mathbb{R}^t \to \{\mathbb{R}, \perp\}$: with $t \leq n$
- $V' : \mathbb{R} \times \mathcal{R} \times \mathbb{R} \to \{0,1\}$

Such that :

- $G'(x, n, t) \to (pk, pk_1, sk_1, \ldots, pk_n, sk_n)$: let define $pkc = pk_1, \ldots, pk_n$
- $S'(sk_i, m) \to \sigma^i_m$
- $C'(pkc, m_1, J, \{\sigma^j_{m_2}\}_{j \in J}) \to \sigma$: with $J \subseteq \Pi$; and $\sigma = \sigma_{m_1}$ iff $|J| \geq t, \forall j \in J : V(pk_j, m_1, \sigma^j_{m_2}) == 1$; otherwise $\sigma = \perp$.
- $V'(pk, m_1, \sigma_{m_2}) \to V(pk, m_1, \sigma_{m_2})$

References

[1] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.