1 Model 1: Crash

We consider a static set Π of n processes with known identities, communicating by reliable point-to-point channels, in a complete graph. Messages are uniquely identifiable. At most f processes can crash, with $n \geq f$.

Synchrony. The network is asynchronous.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which is invoked with the functions $\mathsf{rbcast}(m)$ and $m = \mathsf{rreceived}()$. There exists a shared object called DenyList (DL) (defined below) that is interfaced with a set O of operations. There exist three types of these operations: $\mathsf{append}(x)$, $\mathsf{prove}(x)$ and $\mathsf{read}()$.

Notation. For any indice x we defined by Π_x a subset of Π. We consider two subsets Π_M and Π_V two authorization subsets. Indices $i \in \Pi$ 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 $A \subseteq \mathcal{M}$, $ordered(A)$ returns a deterministic total order over A (e.g., lexicographic order on $(senderId, messageId)$ or on message hashes). For any operation $F \in O$,$F_i(...)$ denotes that the operation F is invoked by process p_i.

2 Primitives

2.1 Reliable Broadcast (RB)

RB provides the following properties in the model.

- **Integrity**: Every message received was previously sent. $\forall p_i : \; m = \mathsf{rreceived}_i() \Rightarrow \exists p_j : \mathsf{rbcast}_j(m)$.
- **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 DenyList Object

We assume a linearizable 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 denote the universe of valid entries to be appended to the DenyList.

  Otherwise, the operation is invalid.

- **PROVE Validity.** Let *op* the invocation of $\mathsf{prove}(x)$ by a process p_i. We said *op* to be invalid, if and only if:
  - $p \notin \Pi_V \subseteq \Pi$; **or**
  - A valid $\mathsf{append}(x)$ appears before *op* in Seq.

  Otherwise, the operation is said to be 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.

We assume that $\Pi_M = \Pi_V = \Pi$ (all processes can invoke append and prove).

3 Atomic Reliable Broadcast (ARB)

Processes export $\textsc{abroadcast}(m)$ and $m = \textsc{adeliver}()$. ARB requires total order:

$$\forall m_1, m_2, \forall p_i, p_j : \ (m_1 = \textsc{adeliver}_i()) \prec (m_2 = \textsc{adeliver}_i()) \Rightarrow (m_1 = \textsc{adeliver}_j()) \prec (m_2 = \textsc{adeliver}_j())$$

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

4 ARB over RB and DL

We present below an example of implementation of Atomic Reliable Broadcast (ARB) using a Reliable Broadcast (RB) primitive and a DenyList (DL) object according to the model and notations defined in Section 2.

4.1 Algorithm

Definition 1 (Closed round)**.** Given a DL linearization H, a round $r \in \mathcal{R}$ is *closed* in H if H contains an operation $\mathsf{append}(r)$. Equivalently, there exists a time after which every $\mathsf{prove}(r)$ is invalid in H.

4.1.1 Handlers and Procedures

Algorithm 1: ARB at process p_i

1.1 **Local Variables:**
1.2 $\quad$ $unordered \leftarrow \emptyset$, $ordered \leftarrow \epsilon$, $delivered \leftarrow \epsilon$;
1.3 $\quad$ $prop[r][j] \leftarrow \perp$, $\forall r, j$;

1.4 **for** $r = 1, 2, \ldots$ **do**
1.5 $\quad$ **wait until** $unordered \setminus ordered \neq \emptyset$;
1.6 $\quad$ $S \leftarrow (unordered \setminus ordered)$;
1.7 $\quad$ rbcast(PROP, $S, \langle r, i \rangle$); prove(r); append(r);
1.8 $\quad$ $winners[r] \leftarrow \{j : (j, r) \in \mathsf{read}()\}$;
1.9 $\quad$ **wait until** $\forall j \in winners[r]$, $prop[r][j] \neq \perp$;
1.10 $\quad$ $M \leftarrow \bigcup_{j \in winners[r]} prop[r][j]$;
1.11 $\quad$ $ordered \leftarrow ordered \cdot ordered(M)$;

1.12 **Upon** *ABROADCAST*(m)
1.13 $\quad$ $unordered \leftarrow unordered \cup \{m\}$;

1.14 **Upon** rdeliver(*PROP*, $S, \langle r, j \rangle$) *from process* p_j
1.15 $\quad$ $unordered \leftarrow unordered \cup \{S\}$;
1.16 $\quad$ $prop[r][j] \leftarrow S$;

1.17 **Upon** *ADELIVER*()
1.18 $\quad$ **if** $ordered \setminus delivered = \emptyset$ **then**
1.19 $\quad\quad$ **return** $\perp$
1.20 $\quad$ let m be the first element in $ordered \setminus delivered$;
1.21 $\quad$ $delivered \leftarrow delivered \cdot m$;
1.22 $\quad$ **return** m

4.2 Correctness

Definition 2 (First APPEND)**.** Given a DL linearization H, for any closed round $r \in \mathcal{R}$, we denote by $\mathsf{append}^{(\star)}(r)$ the earliest $\mathsf{append}(r)$ in H.

Remark 1 (Stable round closure). If a round r is closed, then there exists a linearization point t_0 of $\mathsf{append}(r)$ in the DL, and from that point on, no $\mathsf{prove}(r)$ can be valid. Once closed, a round never becomes open again.

Proof. By Definition 1, some $\mathsf{append}(r)$ occurs in the linearization H.
H is a total order of operations, the set of $\mathsf{append}(r)$ operations is totally ordered, and hence there exists a smallest $\mathsf{append}(r)$ in H. We denote this operation $\mathsf{append}^{(\star)}(r)$ and t_0 its linearization point.
By the validity property of DL, a $\mathsf{prove}(r)$ is valid iff $\mathsf{prove}(r) \prec \mathsf{append}^{(\star)}(r)$. Thus, after t_0, no $\mathsf{prove}(r)$ can be valid.
H is a immutable grow-only history, and hence once closed, a round never becomes open again.
Hence there exists a linearization point t_0 of $\mathsf{append}(r)$ in the DL, and from that point on, no $\mathsf{prove}(r)$ can be valid and the closure is stable. □

Lemma 1 (Across rounds)**.** *If there exists a r such that r is closed, $\forall r'$ such that $r' < r$, r' is also closed.*

Proof. Base. For a closed round $r = 0$, the set $\{r' \in \mathcal{R} : r' < r\}$ is empty, so the claim holds.

Induction step. Assume $r + 1$ is closed. By Definition 2, there exists an earliest operation $\mathsf{append}^{(\star)}(r + 1)$ in the DLlinearization H. Let p_j be the process that invokes this operation.

In Algorithm 1, the call to $\mathsf{append}(\cdot)$ appears at line 1.7, inside the loop indexed by rounds $1, 2, \ldots$. Therefore, if p_j reaches line 1.7 for round $r + 1$, then in the previous loop iteration (round r) process order implies that p_j has already invoked $\mathsf{append}(r)$ at the same line.

Hence an $\mathsf{append}(r)$ exists in H, so round r is closed by Definition 1. We proved:

$$(r + 1 \text{ closed}) \Rightarrow (r \text{ closed}).$$

By repeated application, if some round r is closed, then every round $r' < r$ is also closed. □

Definition 3 (Winner Invariant)**.** For any closed round r, define

$$\mathsf{Winners}_r \triangleq \{j : \mathsf{prove}_j(r) \prec \mathsf{append}^{\star}(r)\}$$

called the unique set of winners of round r.

Lemma 2 (Invariant view of closure)**.** *For any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DLview.*

Proof. Let's take a closed round r. By Definition 2, there exists $\mathsf{append}^{(\star)}(r)$ the earliest $\mathsf{append}(r)$ in the DL linearization.

Consider any correct process p_i that invokes $\mathsf{read}()$ after $\mathsf{append}^{\star}(r)$ in the DL linearization. Since $\mathsf{append}^{\star}(r)$ invalidates all subsequent $\mathsf{prove}(r)$, the set of valid tuples $(\_, r)$ retrieved by a $\mathsf{read}()$ after $\mathsf{append}^{\star}(r)$ is fixed and identical across all correct processes.

Therefore, for any closed round r, all correct processes eventually observe the same set of valid tuples $(\_, \mathsf{prove}(r))$ in their DLview. □

Lemma 3 (Well-defined winners)**.** *For any correct process p_i and round r, if p_i computes winners$[r]$ at line 1.8, then :*

- $\mathsf{Winners}_r$ *is defined;*
- *the computed winners$[r]$ is exactly* $\mathsf{Winners}_r$*.*

Proof. Lets consider a correct process p_i that reach line 1.8 to compute *winners*$[r]$.
By program order, p_i must have executed $\mathsf{append}_i(r)$ at line 1.7 before, which implies by Definition 1 that round r is closed at that point. So by Definition 3, $\mathsf{Winners}_r$ is defined.
By Lemma 2, all correct processes eventually observe the same set of valid tuples $(\_, r)$ in their DLview. Hence, when p_i executes the $\mathsf{read}()$ at line 1.8 after the $\mathsf{append}_i(r)$, it observes a set P that includes all valid tuples $(\_, r)$ such that

$$\textit{winners}[r] = \{j : (j, r) \in P\} = \{j : \mathsf{prove}_j(r) \prec \mathsf{append}^{(\star)}(r)\} = \mathsf{Winners}_r$$

□

Lemma 4 (Winners are non-empty)**.** *For any closed round r, there exists at least one process p_j that invoked $\mathsf{prove}_j(r) \prec \mathsf{append}^{\star}(r)$, so $\mathsf{Winners}_r \neq \emptyset$.*

Proof. Let r be a closed round. By Definition 1, some $\mathsf{append}(r)$ occurs in the DLlinearization H. Let p_i be the process invoking the earliest such operation, $\mathsf{append}^{(\star)}(r)$.

By program order in Algorithm 1, the call to $\mathsf{append}(\cdot)$ at line 1.7 is immediately preceded by $\mathsf{prove}(r)$. Thus p_i must have invoked $\mathsf{prove}_i(r)$ before $\mathsf{append}^{(\star)}(r)$, and by the sequence of code at line 1.7, this $\mathsf{prove}_i(r)$ executes before $\mathsf{append}^{(\star)}(r)$ in the linearization.

By Definition 3, $p_i \in \mathsf{Winners}_r$. Hence $\mathsf{Winners}_r \neq \emptyset$. □

Lemma 5 (Winners must propose)**.** *For any closed round r, $\forall i \in \mathsf{Winners}_r$, process p_i must have invoked a $\mathsf{rbcast}(PROP, S^{(i)}, \langle r, i\rangle)$ and hence any correct will eventually set $prop[r][i]$ to a non-$\perp$ value.*

Proof. Fix a closed round r. By Definition 3, for any $i \in \mathsf{Winners}_r$, there exist a valid $\mathsf{prove}_i(r)$ such that $\mathsf{prove}_i(r) \prec \mathsf{append}^{\star}(r)$ in the DL linearization. By program order, if i invoked a valid $\mathsf{prove}_i(r)$ at line 1.7 he must have invoked $\mathsf{rbcast}(PROP, S^{(i)}, \langle r, i\rangle)$ directly before.

Let take a correct process p_j, by RB *Validity*, every correct process eventually receives i's RB message for round r, which sets $prop[r][i]$ to a non-$\perp$ value at line 1.16. □

Definition 4 (Messages invariant)**.** For any closed round r and any correct process p_i such that $\forall j \in \mathsf{Winners}_r : prop^{[i)}[r][j] \neq \perp$, define

$$\mathsf{Messages}_r \triangleq \bigcup_{j \in \mathsf{Winners}_r} prop^{(i)}[r][j]$$

as the set of messages proposed by the winners of round r.

Lemma 6 (Eventual proposal closure)**.** *If a correct process p_i define M at line 1.10, then for every $j \in \mathsf{Winners}_r$, $prop^{(i)}[r][j] \neq \perp$.*

Proof. Let take a correct process p_i that computes M at line 1.10. By Lemma 3, p_i computation is the winner set $\mathsf{Winners}_r$.

By Lemma 4, $\mathsf{Winners}_r \neq \emptyset$. The instruction at line 1.10 where p_i computes M is guarded by the condition at line 1.9, which ensures that p_i has received all RB messages from every winner $j \in \mathsf{Winners}_r$. Hence, $M = \bigcup_{j \in \mathsf{Winners}_r} prop^{(i)}[r][j]$, we have $prop^{(i)}[r][j] \neq \perp$ for all $j \in \mathsf{Winners}_r$. □

Lemma 7 (Unique proposal per sender per round)**.** *For any round r and any process p_i, p_i invokes at most one $\mathsf{rbcast}(PROP, S, \langle r, i\rangle)$.*

Proof. In Algorithm 1, the only place where a process p_i can invoke $\mathsf{rbcast}(PROP, S, \langle r, i\rangle)$ is at line 1.7, which appears inside the main loop indexed by rounds $r = 1, 2, \ldots$.

Each iteration of this loop processes exactly one round value r, and within that iteration, line 1.7 is executed at most once. Since the loop variable r takes each value $1, 2, \ldots$ at most once during the execution, process p_i invokes $\mathsf{rbcast}(PROP, S, \langle r, i\rangle)$ at most once for any given round r. □

Lemma 8 (Proposal convergence)**.** *For any round r, for any correct processes p_i that execute line 1.10, we have*

$$M^{(i)} = \mathsf{Messages}_r$$

Proof. Let take a correct process p_i that compute M at line 1.10. That implies that p_i has defined *winnersr* at line 1.8. It implies that, by Lemma 3, r is closed and $winners_r = \mathsf{Winners}_r$.
By Lemma 6, for every $j \in \mathsf{Winners}_r$, $prop^{(i)}[r][j] \neq \perp$. By Lemma 7, each winner j invokes at most one $\mathsf{rbcast}(PROP, S^{(j)}, \langle r, j\rangle)$, so $prop^{(i)}[r][j] = S^{(j)}$ is uniquely defined. Hence, when p_i computes

$$M^{(i)} = \bigcup_{j \in \mathsf{Winners}_r} prop^{(i)}[r][j] = \bigcup_{j \in \mathsf{Winners}_r} S^{(j)} = \mathsf{Messages}_r.$$

□

Lemma 9 (Inclusion)**.** *If some correct process invokes* ABROADCAST(m)*, then there exist a round r and a process $j \in \mathsf{Winners}_r$ such that p_j invokes*

$$\mathsf{rbcast}(PROP, S, \langle r, j\rangle) \quad \textit{for some } S \textit{ with} \quad m \in S.$$

Proof. Let p_i be a correct process that invokes ABROADCAST(m). By the handler at line 1.13, m is added to *unordered*. Since p_i is correct, it continues executing the main loop.

Consider any iteration of the loop where p_i executes line 1.6 while $m \in (\textit{unordered} \setminus \textit{ordered})$. At that iteration, for some round r, process p_i constructs S containing m and invokes $\mathsf{rbcast}(PROP, S, \langle r, i\rangle)$ at line 1.7.

We distinguish two cases:

- **Case 1: p_i is a winner.** If $p_i \in \mathsf{Winners}_r$ for this round r, then by Definition 3 and program order, p_i has invoked $\mathsf{rbcast}(PROP, S, \langle r, i\rangle)$ with $m \in S$, and the lemma holds with $j = i$.
- **Case 2: p_i is not a winner.** If $p_i \notin \mathsf{Winners}_r$, then by the RB *Validity* property, all correct processes eventually $\mathsf{rdeliver}p_i$'s message. By line 1.15, each correct process p_k adds m to its own *unordered* set. Hence every correct process will eventually attempt to broadcast m in some subsequent round.

  Since there are infinitely many rounds and finitely many processes, and by Lemma 4 every closed round has at least one winner, there must exist a round r' and a correct process $p_j \in \mathsf{Winners}_{r'}$ such that $m \in (\textit{unordered} \setminus \textit{ordered})$ when p_j constructs its proposal S at line 1.6 for round r'. Hence p_j invokes $\mathsf{rbcast}(PROP, S, \langle r', j\rangle)$ with $m \in S$.

In both cases, there exists a round and a winner whose proposal includes m. □

Lemma 10 (Broadcast Termination)**.** *A correct process which invokes* ABROADCAST(m)*, eventually exits the function and returns.*

Proof. Consider a correct process p_i that invokes ABROADCAST(m). The statement is true if $\exists r_1$ such that $r_1 \geq r_{max}$ and if $(i, r_1) \in P$; or if $\exists r_2$ such that $r_2 \geq r_{max}$ and if $\exists j : (j, r_2) \in P \wedge m \in prop[r_2][j]$ (like guarded at B8).

Consider that there exists no round r_1 such that p_i invokes a valid $\mathsf{prove}(r_1)$. In this case p_i invokes infinitely many $\mathsf{rbcast}(PROP, S, \langle \_, i\rangle)$ at line 1.7 with $m \in S$ (line 1.6).
The assumption that no $\mathsf{prove}(r_1)$ invoked by p is valid implies by DL *Validity* that for every round $r' \geq r_{max}$, there exists at least one $\mathsf{append}(r')$ in the DL linearization, hence by Lemma 4 for every possible round r' there at least a winner. Because there is an infinite number of rounds, and a finite number of processes, there exists at least one process p_k that invokes infinitely many valid $\mathsf{prove}(r')$ and by extension infinitely many ABROADCAST(_). By RB *Validity*, p_k eventually receives p_i 's RB messages. Let call t_0 the time when p_k receives p_i 's RB message.
At t_0, p_k executes Algorithm 1 and sets $\textit{received} \leftarrow \textit{received} \cup \{S\}$ with $m \in S$ (line 1.15). For the first invocation of ABROADCAST(_) by p_k after the execution of `RReceived()`, p_k must include m in his proposal S at line 1.6 (because m is pending in j's $\mathsf{received} \setminus \textit{delivered}$ set). There exists a minimum round r_2 such that $p_k \in \mathsf{Winners}_{r_2}$ after t_0. By Lemma 5, eventually $prop^{(i)}[r_2][k] \neq \bot$. By Lemma 7, $prop^{(i)}[r_2][k]$ is uniquely defined as the set S proposed by p_k at B6, which by Lemma 9 includes m. Hence eventually $m \in prop^{(i)}[r_2][k]$ and $k \in \mathsf{Winners}_{r_2}$.

So if p_i is a winner he satisfies the condition $(i, r_1) \in P$. And we show that if the first condition is never satisfied, the second one will eventually be satisfied. Hence either the first or

the second condition will eventually be satisfied, and p_i eventually exits the loop and returns from ABROADCAST(m). □

Lemma 11 (Validity)**.** *If a correct process p invokes* ABROADCAST(m)*, then every correct process that invokes a infinitely many times* ADELIVER() *eventually delivers m.*

Proof. Let p_i a correct process that invokes ABROADCAST(m) and p_q a correct process that infinitely invokes ADELIVER(). By Lemma 9, there exist a closed round r and a correct process $j \in \mathsf{Winners}_r$ such that p_j invokes

$$\mathsf{rbcast}(PROP, S, \langle r, j\rangle) \quad \text{with} \quad m \in S.$$

By Lemma 6, when p_q computes M at line 1.10, $prop[r][j]$ is non-$\perp$ because $j \in \mathsf{Winners}_r$. By Lemma 7, p_j invokes at most one $\mathsf{rbcast}(PROP, S, \langle r, j\rangle)$, so $prop[r][j]$ is uniquely defined. Hence, when p_q computes

$$M = \bigcup_{k \in \mathsf{Winners}_r} prop[r][k],$$

we have $m \in prop[r][j] = S$, so $m \in M$. By Lemma 8, M is invariant so each computation of M by a correct process includes m. At each invocation of $m' =$ ADELIVER(), m' is added to *delivered* until $M \subseteq$ *delivered*. Once this happens we're assured that there exists an invocation of ADELIVER() which return m. Hence m is well delivered. □

Lemma 12 (No duplication)**.** *No correct process delivers the same message more than once.*

Proof. Let consider two invokations of ADELIVER() made by the same correct process which returns m. Let call these two invocations respectively ADELIVER$^{(A)}$() and ADELIVER$^{(B)}$().

When ADELIVER$^{(A)}$() occurs, by program order and because it reached line 1.21 to return m, the process must have add m to *delivered*. Hence when ADELIVER$^{(B)}$() reached line 1.20 to extract the next message to deliver, it can't be m because $m \notin (ordered \setminus delivered)$. So a ADELIVER$^{(B)}$() which delivers m can't occur. □

Lemma 13 (Total order)**.** *For any two messages m_1 and m_2 delivered by correct processes, if a correct process p_i delivers m_1 before m_2, then any correct process p_j that delivers both m_1 and m_2 delivers m_1 before m_2.*

Proof. Consider a correct process that delivers both m_1 and m_2. By Lemma 11, there exists a closed rounds r'_1 and r'_2 and correct processes $k_1 \in \mathsf{Winners}_{r'_1}$ and $k_2 \in \mathsf{Winners}_{r'_2}$ such that

$$\mathsf{rbcast}(PROP, S_1, \langle r'_1, k_1\rangle) \quad \text{with} \quad m_1 \in S_1,$$

$$\mathsf{rbcast}(PROP, S_2, \langle r'_2, k_2\rangle) \quad \text{with} \quad m_2 \in S_2.$$

Let consider three cases :

- **Case 1:** $r_1 < r_2$. By program order, any correct process must have waited to append in *delivered* every messages in M (which contains m_1) to increment $\mathsf{current}$ and eventually set $\mathsf{current} = r_2$ to compute M and then invoke the $m_2 =$ ADELIVER(). Hence, for any correct process that delivers both m_1 and m_2, it delivers m_1 before m_2.
- **Case 2:** $r_1 = r_2$. By Lemma 8, any correct process that computes M at line 1.10 computes the same set of messages $\mathsf{Messages}_{r_1}$. By line 1.11 the messages are pull in a deterministic order defined by *ordered*(_). Hence, for any correct process that delivers both m_1 and m_2, it delivers m_1 and m_2 in the deterministic order defined by *ordered*(_).

In all possible cases, any correct process that delivers both m_1 and m_2 delivers m_1 and m_2 in the same order. □

Lemma 14 (Fifo Order)**.** *For any two distints messages m_1 and m_2 broadcast by the same correct process p_i, if p_i invokes* ABROADCAST(m_1) *before* ABROADCAST(m_2)*, then any correct process p_j that delivers both m_1 and m_2 delivers m_1 before m_2.*

Proof. Let take m_1 and m_2 broadcast by the same correct process p_i, with p_i invoking ABROADCAST(m_1) before ABROADCAST(m_2). By Lemma 11, there exist closed rounds r_1 and r_2 such that $r_1 \leq r_2$ and correct processes $k_1 \in \mathsf{Winners}_{r_1}$ and $k_2 \in \mathsf{Winners}_{r_2}$ such that

$$\mathsf{rbcast}(PROP, S_1, \langle r_1, k_1 \rangle) \quad \text{with} \quad m_1 \in S_1,$$

$$\mathsf{rbcast}(PROP, S_2, \langle r_2, k_2 \rangle) \quad \text{with} \quad m_2 \in S_2.$$

By program order, p_i must have invoked $\mathsf{rbcast}(PROP, S_1, \langle r_1, i \rangle)$ before $\mathsf{rbcast}(PROP, S_2, \langle r_2, i \rangle)$. By Lemma 7, any process invokes at most one $\mathsf{rbcast}(PROP, S, \langle r, i \rangle)$ per round, hence $r_1 < r_2$. By Lemma 13, any correct process that delivers both m_1 and m_2 delivers them in a deterministic order.

In all possible cases, any correct process that delivers both m_1 and m_2 delivers m_1 before m_2. □

Theorem 15 (FIFO-ARB)**.** *Under the assumed DL synchrony and RB reliability, the algorithm implements FIFO Atomic Reliable Broadcast.*

Proof. We show that the algorithm satisfies the properties of FIFO Atomic Reliable Broadcast under the assumed DL synchrony and RB reliability.

First, by Lemma 10, if a correct process invokes ABROADCAST(m), then it eventually returns from this invocation. Moreover, Lemma 11 states that if a correct process invokes ABROADCAST(m), then every correct process that invokes ADELIVER() infinitely often eventually delivers m. This gives the usual Validity property of ARB.

Concerning Integrity and No-duplicates, the construction only ever delivers messages that have been obtained from the underlying RB primitive. By the Integrity property of RB, every such message was previously rbcast by some process, so no spurious messages are delivered. In addition, Lemma 12 states that no correct process delivers the same message more than once. Together, these arguments yield the Integrity and No-duplicates properties required by ARB.

For the ordering guarantees, Lemma 13 shows that for any two messages m_1 and m_2 delivered by correct processes, every correct process that delivers both m_1 and m_2 delivers them in the same order. Hence all correct processes share a common total order on delivered messages. Furthermore, Lemma 14 states that for any two messages m_1 and m_2 broadcast by the same correct process, any correct process that delivers both messages delivers m_1 before m_2 whenever m_1 was broadcast before m_2. Thus the global total order extends the per-sender FIFO order of ABROADCAST.

All the above lemmas are proved under the assumptions that DL satisfies the required synchrony properties and that the underlying primitive is a Reliable Broadcast (RB) with Integrity, No-duplicates and Validity. Therefore, under these assumptions, the algorithm satisfies Validity, Integrity/No-duplicates, total order and per-sender FIFO order, and hence implements FIFO Atomic Reliable Broadcast, as claimed. □

4.3 Reciprocity

So far, we assumed the existence of a synchronous DenyList (DL) object and showed how to upgrade a Reliable Broadcast (RB) primitive into FIFO Atomic Reliable Broadcast (ARB). We now briefly argue that, conversely, an ARB primitive is strong enough to implement a synchronous DL object.

DenyList as a deterministic state machine. Without anonymity, the DLspecification defines a deterministic abstract object: given a sequence Seq of operations append(x), prove(x), and read(), the resulting sequence of return values and the evolution of the abstract state (set of appended elements, history of operations) are uniquely determined.

State machine replication over ARB. Assume a system that exports a FIFO-ARB primitive with the guarantees that if a correct process invokes ABROADCAST(m), then every correct process eventually ADELIVER(m) and the invocation eventually returns. Following the classical *state machine replication* approach such as described in Schneider [1], we can implement a fault-tolerant service by ensuring the following properties:

> **Agreement.** Every nonfaulty state machine replica receives every request.
> **Order.** Every nonfaulty state machine replica processes the requests it receives in the same relative order.

Which are cover by our FIFO-ARB specification.

Correctness.

Theorem 16 (From ARB to synchronous DL)**.** *In an asynchronous message-passing system with crash failures, assume a FIFO Atomic Reliable Broadcast primitive with Integrity, No-duplicates,Validity, and the liveness of* ABROADCAST*. Then there exists an implementation of a DenyList object that satisfies Termination, Validity, and Anti-flickering properties.*

Proof. Because the DLobject is deterministic, all correct processes see the same sequence of operations and compute the same sequence of states and return values. We obtain:

- **Termination.** The liveness of ARB ensures that each ABROADCAST invocation by a correct process eventually returns, and the corresponding operation is eventually delivered and applied at all correct processes. Thus every append, prove, and read operation invoked by a correct process eventually returns.
- **APPEND/PROVE/READ Validity.** The local code that forms ABROADCASTrequests can achieve the same preconditions as in the abstract DLspecification (e.g., $p \in \Pi_M$, $x \in S$ for append(x)). Once an operation is delivered, its effect and return value are exactly those of the sequential DLspecification applied in the common order.
- **PROVE Anti-Flickering.** In the sequential DLspecification, once an element x has been appended, all subsequent prove(x) are invalid forever. Since all replicas apply operations in the same order, this property holds in every execution of the replicated implementation: after the first linearization point of append(x), no later prove(x) can return "valid" at any correct process.

Formally, we can describe the DLobject with the state machine approach for crash-fault, asynchronous message-passing systems with a total order broadcast layer [1]. □

4.3.1 Example executions

5 BFT-ARB over RB and DL

5.1 Model extension

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

be Byzantine, and we assume $n > 3f$.

Synchrony. The network is asynchronous.

Communication. Processes can exchange through a Reliable Broadcast (RB) primitive (defined below) which is invoked with the functions rbcast(m) and $m =$ rreceived(). There exists a shared object called DenyList (DL) (defined below) that is interfaced with a set O of operations. There exist three types of these operations: append(x), prove(x) and read().

Byzantine behaviour A process is said to exhibit Byzantine behaviour if it deviates arbitrarily from the prescribed algorithm. Such deviations may, for instance, consist in invoking primitives (rbcast, append, prove, etc.) with invalid inputs or inputs crafted maliciously, colluding with other Byzantine processes in an attempt to manipulate the system state or violate its guarantees, deliberately delaying or accelerating the delivery of messages to selected nodes so as to disrupt the expected timing of operations, or withholding messages and responses in order to induce inconsistencies in the system state.

Byzantine processes are constrained by the following. They cannot forge valid cryptographic signatures or threshold shares without access to the corresponding private keys. They cannot violate the termination, validity, or anti-flickering properties of the DL object. They also cannot break the integrity, no-duplicates, or validity properties of the RB primitive.

Notation. For any indice k we defined by DL$[k]$ as the k-th DenyList object. For a given DL$[k]$ and any indice x we defined by Π_x^k a subset of Π. Still for a given k we consider $\Pi_M^k \subseteq \Pi$ and $\Pi_V^k \subseteq \Pi$ two authorization subsets for DL$[k]$. Indice $i \in \Pi$ 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 $\prec_k$ for the DL$[k]$ linearization: $x \prec_k y$ means that operation x appears strictly before y in the linearized history of DL$[k]$. For any finite set $A \subseteq \mathcal{M}$, *ordered*(A) returns a deterministic total order over A (e.g., lexicographic order on (*senderId*, *messageId*) or on message hashes). For any operation $F \in O$,$F_i(...)$ denotes that the operation F is invoked by process p_i, and by $F_i^k(...)$ the same operation invoked on the DL$[k]$ object.

5.2 Primitives

5.2.1 t-BFT-DL

We consider a t-Byzantine Fault Tolerant DenyList (t-BFT-DL) with the following properties. There are 3 operations : BFT-PROVE(x), BFT-APPEND(x), BFT-READ() such that :

Termination. Every operation BFT-APPEND(x), BFT-PROVE(x), and BFT-READ() invoked by a correct process always returns.

PROVE Validity. The invocation of $op =$ BFT-PROVE(x) by a correct process is valid iff there exist a set of correct process C such that $\forall c \in C$, c invoke $op_2 =$ BFT-APPEND(x) with $op_2 \prec op_1$ and $|C| \leq t$

PROVE Anti-Flickering. If the invocation of a operation $op =$ BFT-PROVE(x) by a correct process $p \in \Pi_V$ is invalid, then any BFT-PROVE(x) operation that appears after op in Seq is invalid.

READ Liveness. Let $op = \text{BFT-READ}()$ invoke by a correct process such that R is the result of op. For all $(i, \mathsf{prove}(x)) \in R$ there exist a valid invocation of $\text{BFT-PROVE}(x)$ by p_i.

READ Anti-Flickering. Let op_1, op_2 two $\text{BFT-READ}()$ operations that returns respectively R_1, R_2. Iff $op_1 \prec op_2$ then $R_2 \subseteq R_1$. Otherwise $R_1 \subseteq R_2$.

READ Safety. Let op_1, op_2 respectively a valid $\text{BFT-PROVE}(x)$ operation submited by the process p_i and a $\text{BFT-READ}()$ operation submited by any correct process such that $op_1 \prec op_2$. Let R the result of op_2 then $R \ni (i, x)$

5.3 DL ⇒ t-BFT-DL

Fix $3t < |M|$. Let

$$\mathcal{U} = \{\, U \subseteq M \mid |U| = |M| - t \,\}.$$

For each $U \in \mathcal{U}$, we instantiate one DenyList object DL_U whose authorization sets are

$$\Pi_M(DL_T) = S_T = U \qquad \text{and} \qquad \Pi_V(DL_T) = V.$$

$$|\mathcal{U}| = \binom{|M|}{|M| - t}.$$

Algorithm 2: t-BFT-DL implementation using multiple DL objects

```
2.1  Function BFT-APPEND(x)
2.2      for each U ∈ 𝒰 st i ∈ U do
2.3          DL_U.append(x);

2.4  Function BFT-PROVE(x)
2.5      state ← false;
2.6      for each U ∈ 𝒰 do
2.7          state ← state OR DL_U.prove(x);
2.8      return state;

2.9  Function BFT-READ()
2.10     results ← ∅;
2.11     for each U ∈ 𝒰 do
2.12         results ← results ∪ DL_U.read();
2.13     return results;
```

Lemma 17 (BFT-PROVE Validity)**.** *The invocation of $op = \text{BFT-PROVE}(x)$ by a correct process is invalid iff there exist at least $t + 1$ distinct processes in M that invoked a valid $\text{BFT-APPEND}(x)$ before op in* Seq*.*

Proof. Let $op = \text{BFT-PROVE}(x)$ be an invocation by a correct process p_i. Let $A \subseteq M$ be the set of distinct issuers that invoked $\text{BFT-APPEND}(x)$ before op in Seq.

- **Case (i):** $|A| \geq t+1$**.** Fix any $U \in \mathcal{U}$. $A \cap U \neq \emptyset$. Pick $j \in A \cap U$. Since $j \in U$, the call BFT-APPEND$_j(x)$ triggers $DL_U.\mathsf{append}(x)$, and because BFT-APPEND$_j(x) \prec op$ in Seq, this induces a valid $DL_U.\mathsf{append}(x)$ that appears before the induced $DL_U.\mathsf{prove}(x)$ by p_i. By **PROVE Validity** of DL, the induced $DL_U.\mathsf{prove}(x)$ is invalid. As this holds for every $U \in \mathcal{U}$, there is *no* component DL_U where $\mathsf{prove}(x)$ is valid, so the field *state* at line 2.7 is never becoming true, and *op* return false.
- **Case (ii):** $|A| \leq t$**.** There exists $U^\star \in \mathcal{U}$ such that $A \cap U^\star = \emptyset$. For any $j \in A$, we have $j \notin U^\star$, so BFT-APPEND$_j(x)$ does *not* call $DL_{U^\star}.\mathsf{append}(x)$. Hence no valid $DL_{U^\star}.\mathsf{append}(x)$ appears before the induced $DL_{U^\star}.\mathsf{prove}(x)$. Since also $i \in \Pi_V(DL_{U^\star})$, by **PROVE Validity** of DL the induced $DL_{U^\star}.\mathsf{prove}(x)$ is valid. Therefore, there exists a component with a valid $\mathsf{prove}(x)$, so *op* is valid.

Combining the cases yields the claimed characterization of invalidity. □

Lemma 18 (BFT-PROVE Anti-Flickering)**.** *If the invocation of a operation op =* BFT-PROVE(x) *by a correct process $p \in \Pi_V$ is invalid, then any* BFT-PROVE(x) *operation that appears after op in* Seq *is invalid.*

Proof. Let $op =$ BFT-PROVE(x) be an invocation by a correct process p_i that is *invalid* in Seq. By BFT-PROVE Validity, this implies that there exist at least $t+1$ *distinct* processes in M that invoked a *valid* BFT-APPEND(x) before *op* in Seq. Let $A \subseteq M$ denote that set, with $|A| \geq t+1$.

Fix any $U \in \mathcal{U}$. We have $A \cap U \neq \emptyset$. Pick $j \in A \cap U$. Since $j \in U$, the call BFT-APPEND$_j(x)$ triggers a call $DL_U.\mathsf{append}(x)$. Moreover, because BFT-APPEND$_j(x) \prec op$ in Seq, the induced $DL_U.\mathsf{append}(x)$ appears before the induced $DL_U.\mathsf{prove}(x)$ of *op* in the projection Seq_U.

Hence, in Seq_U, there exists a *valid* $DL_U.\mathsf{append}(x)$ that appears before the $DL_U.\mathsf{prove}(x)$ induced by *op*. By **PROVE Validity** the base DL object, the induced $DL_U.\mathsf{prove}(x)$ is therefore *invalid* in Seq_U.

Let $op' =$ BFT-PROVE(x) be any invocation such that $op \prec op'$ in Seq. Fix again any $U \in \mathcal{U}$. Hence, the $DL_U.\mathsf{prove}(x)$ induced by op' appears after the $DL_U.\mathsf{prove}(x)$ induced by *op* in Seq_U. Since the induced $DL_U.\mathsf{prove}(x)$ of *op* is invalid, by **PROVE Anti-Flickering** of DL, *every* subsequent $DL_U.\mathsf{prove}(x)$ in Seq_U is invalid.

As this holds for every $U \in \mathcal{U}$, there is no component DL_U in which the induced $\mathsf{prove}(x)$ of op' is valid. □

Lemma 19 (BFT-READ Liveness)**.** *Let op =* BFT-READ() *invoke by a correct process such that R is the result of op. For all $(i,x) \in R$ there exist a valid invocation of* BFT-PROVE(x) *by p_i.*

Proof. Let R the result of a $READ()$ operation submit by any correct process. $(i, \mathsf{prove}(x)) \in R$ implie that $\exists U^\star \in \mathcal{U}$ such that $(i,x) \in R^{U^\star}$ with $R^{U^\star}$ the result of $DL_{U^\star}.\mathsf{read}()$. By **READ Validity** $(i,x) \in R^{U^\star}$ implie that there exist a valid $DL_{U^\star}.\mathsf{prove}_i(x)$. The for loop in the BFT-PROVE(x) implementation return true iff there at least one valid $DL_U.\mathsf{prove}_i(x)$ for any $U \in \mathcal{U}$.

Hence because there exist a $U^\star$ such that $DL_{U^\star}.\mathsf{prove}_i(x)$, there exist a valid BFT-PROVE$_i(x)$.
$(i,x) \in R \implies \exists$BFT-PROVE$_i(x)$ □

Lemma 20 (BFT-READ Anti-Flickering)**.** *Let op_1, op_2 two* BFT-READ() *operations that returns respectively R_1, R_2. Iff $op_1 \prec op_2$ then $R_2 \subseteq R_1$. Otherwise $R_1 \subseteq R_2$.*

Proof. Let R_1, R_2 respectively the output of two BFT-READ() operations op_1, op_2 such that $op_1 \prec op_2$. By the implementation of BFT-READ, $R_k = \bigcup_{U \in \mathcal{U}} R_k^U$ where R_k^U is the result of $DL_U.\mathsf{read}()$ during op_k.

Because $op_1 \prec op_2$ for any $U \in \mathcal{U}$, the $DL_U.\mathsf{read}()$ induced by op_1 happen before the $DL_U.\mathsf{read}()$ induced by op_2. Hence we have for all $U, R_2^U \subseteq R_1^U$. Therefore

$$\bigcup_U R_2^U \subseteq \bigcup_U R_1^U \implies R_2 \subseteq R_1$$

□

Lemma 21 (BFT-READ Safety)**.** *Let op_1, op_2 respectively a valid BFT-PROVE(x) operation submited by the process p_i and a BFT-READ$()$ operation submited by any correct process such that $op_1 \prec op_2$. Let R the result of op_2 then $R \ni (i, x)$*

Proof. Let $op_1 = \text{BFT-PROVE}_i(x)$ be a valid operation by a correct process p_i and $op_2 = \text{BFT-READ}()$ be any BFT-READ() operation such that $op_1 \prec op_2$ in Seq. By BFT-PROVE Validity, there exist at most t distinct processes in M that invoked a valid BFT-APPEND(x) before op_1 in Seq. Let $A \subseteq M$ denote that set, with $|A| \leq t$.

There exists $U^\star \in \mathcal{U}$ such that $A \cap U^\star = \emptyset$. For any $j \in A$, we have $j \notin U^\star$, so BFT-APPEND$^{(j)}(x)$ does *not* call $DL_{U^\star}.\mathsf{append}(x)$. Hence no valid $DL_{U^\star}.\mathsf{append}(x)$ appears before the induced $DL_{U^\star}.\mathsf{prove}(x)$ of op_1. Since also $i \in \Pi_V(DL_{U^\star})$, by **PROVE Validity** of DL the induced $DL_{U^\star}.\mathsf{prove}_i(x)$ is valid.

Now, because $op_1 \prec op_2$ in Seq, the induced $DL_{U^\star}.\mathsf{prove}_i(x)$ appears before the induced $DL_{U^\star}.\mathsf{read}()$ of op_2 in $\mathsf{Seq}_{U^\star}$. By **READ Safety** of DL, the result $R^{U^\star}$ of the induced $DL_{U^\star}.\mathsf{read}()$ contains (i, x).

Finally, by the implementation of BFT-READ(), we have $R = \bigcup_{U \in \mathcal{U}} R^U$, so $(i, x) \in R$. □

Theorem 22. *For any fixed value t such that $3t < |M|$, multiple DenyList Object can be used to implement a t-Byzantine Fault Tolerant DenyList Object.*

Proof. Follows directly from the previous lemmas. □

5.4 Algorithm

Algorithm 3: t-BFT ARB at process p_i

3.1 **Local Variables:**
3.2 $unordered \leftarrow \emptyset$, $ordered \leftarrow \epsilon$, $delivered \leftarrow \epsilon$;
3.3 $prop[r][j] \leftarrow \perp, \forall j, r \in \Pi \times \mathbb{N}$;
3.4 $done[r] \leftarrow \emptyset, \forall r \in \mathbb{N}$;

3.5 **for** $r = 1, 2, \ldots$ **do**
3.6 **wait until** $unordered \setminus ordered \neq \emptyset$;
3.7 $S \leftarrow unordered \setminus ordered$; $\mathsf{rbcast}(i, \texttt{PROP}, S, r)$;
3.8 **wait until** $|\text{VALIDATED}(r)| \geq n - t$;
3.9 **foreach** $j \in \Pi$ **do**
3.10 BFT-APPEND$(\langle j, r \rangle)$;
3.11 **foreach** $j \in \Pi$ **do**
3.12 $\mathsf{send}(j, \texttt{DONE}, r)$;
3.13 **wait until** $|done[r]| \geq n - t$;
3.14 $winners[r] \leftarrow$ VALIDATED(r);
3.15 **wait until** $\forall j \in winners[r],\ prop[r][j] \neq \perp$;
3.16 $M \leftarrow \bigcup_{j \in winners[r]} prop[r][j]$;
3.17 $ordered \leftarrow ordered \cdot ordered(M)$;

3.18 **Function** *VALIDATED(r)*
3.19 **return** $\{j : |\{k : (k, r) \in$ *BFT-READ*$()\}| \geq t + 1\}$;

3.20 **Upon** *ABROADCAST*(m)
3.21 $unordered \leftarrow unordered \cup \{m\}$;

3.22 **Upon** $\mathsf{rdeliver}(\texttt{PROP}, S, \langle j, r \rangle)$ *from process* p_j
3.23 $unordered \leftarrow unordered \cup S$; $prop[r][j] \leftarrow S$;
3.24 BFT-PROVE$(\langle j, r \rangle)$;

3.25 **Upon** $\mathsf{receive}(\texttt{DONE}, r)$ *from process* p_j
3.26 $done[r] \leftarrow done[r] \cup \{j\}$;

3.27 **Upon** *ADELIVER*$()$
3.28 **if** $ordered \setminus delivered = \emptyset$ **then**
3.29 **return** $\perp$
3.30 let m be the first message in $ordered \setminus delivered$;
3.31 $delivered \leftarrow delivered \cdot \{m\}$;
3.32 **return** m

Everything below has to be updated

Definition 5 (BFT Closed round for k)**.** Given Seq^k the linearization of the BFT-DL $Y[k]$, a round $r \in \mathcal{R}$ is *closed* in Seq iff there exist at least $n - f$ distinct processes $j \in \Pi$ such that BFT-APPEND$_j(r)$ appears in Seq^k. Let call BFT-APPEND$(r)^\star$ the $(n-f)^{th}$ BFT-APPEND(r).

Definition 6 (BFT Closed round)**.** A round $r \in \mathcal{R}$ is *closed* iff for all $\mathsf{DL}[k]$, r is closed in Seq^k.

5.5 Proof of correctness

Remark 2 (BFT Stable round closure). If a round r is closed, no more BFT-PROVE(r) can be valid and thus linearized. In other words, once BFT-APPEND(r)* is linearized, no more process can make a proof on round r, and the set of valid proofs for round r is fixed. Therefore $\mathsf{Winners}_r$ is fixed.

Proof. By definition r closed means that for all process p_i, there exist at least $n-f$ distinct processes $j \in \Pi$ such that BFT-APPEND$_j(r)$ appears in Seq^k. By BFT-PROVE Validity, any subsequent BFT-PROVE(r) is invalid because at least $n-f$ processes already invoked a valid BFT-APPEND(r) before it. Thus no new valid BFT-PROVE(r) can be linearized after BFT-APPEND(r)*. Hence the set of valid proofs for round r is fixed, and so is $\mathsf{Winners}_r$. □

Lemma 23 (BFT Across rounds). *For any r, r' such that $r < r'$, if r' is closed, r is also closed.*

Proof. Let $r \in \mathcal{R}$. By Definition 6, if $r+1$ is closed, then for all $\mathsf{DL}[k]$, $r+1$ is closed in Seq^k. By the implementation, a process can only invoke BFT-APPEND($r+1$) after observing at least $n-f$ valid BFT-PROVE(r), which means that for all $\mathsf{DL}[k]$, r is closed in Seq^k. Hence by Definition 6, r is closed.

Because r is monotonically increasing, we can reccursively apply the same argument to conclude that for any r, r' such that $r < r'$, if r' is closed, r is also closed. □

Lemma 24 (BFT Progress). *For any correct process p_i such that*

$$\mathsf{received} \setminus (\mathit{delivered} \cup (\cup_{r'<r} \cup_{j \in W[r'] prop[r'][j]})) \neq \emptyset$$

with r the highest closed round in the DL linearization. Eventually $r+1$ will be closed.

Lemma 25 (BFT Winners invariant). *For any closed round r, define*

$$\mathsf{Winners}_r = \{j : \mathit{BFT\text{-}PROVE}_j(r) \prec \mathit{BFT\text{-}APPEND}^\star(r)\}$$

called the unique set of winners of round r.

Lemma 26 (BFT n-f lower-bounded Winners). *Let r a closed round, $|W[r]| \geq n-f$.*

Remark 3. Because we assume $n \geq 2f+1$, if $|W[r]| \geq n-f$ at least 1 correct have to be in $W[r]$ to progress.

Lemma 27 (BFT Winners must purpose). *Let r a closed round, for all process p_j such that $j \in W[r]$, p_j must have executed* $\mathsf{rbcast}(j, PROP, \_, r)$ *and hence any correct will eventually set $prop[r][j]$ to a non-$\perp$ value.*

Lemma 28 (BFT Messages Incariant). *For any closed round r and any correct process p_i such that $\forall j \in \mathsf{Winners}_r$: $prop^{(i)}[r][j] \neq \perp$ define*

$$\mathsf{Messages}_r = \cup_{j \in \mathsf{Winners}_r} prop^{(i)}[r][j]$$

as the set of messages proposed by the winners of round r

Lemma 29 (BFT EVentual proposal closure). *If a correct process p_i define M at line 3.16, then for every $j \in \mathsf{Winners}_r$, $prop^{(i)}[r][j] \neq \perp$.*

Lemma 30 (BFT Unique proposal per sender per round). *For any round r and any process p_i, if p_i invokes two* rbcast *call for the same round, such that* $\mathsf{rbcast}(i, PROP, S, r) \prec \mathsf{rbcast}(i, PROP, S', r)$. *Then for any correct process p_j, $prop^{(j)}[r][i] \in \{\perp, S\}$*

Theorem 31. *The algorithm implements a BFT Atomic Reliable Broadcast.*

6 Implementation of BFT-DenyList and Threshold Cryptography

6.1 DenyList

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

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

$$\forall i, j \in \{1, \ldots, k\},\ i \neq j \implies M_i \neq M_j.$$

Lemma 32. *$\exists M_i \in M : \forall p \in M_i$ 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 33. *$\forall M_i \in M, \exists p \in M_i$ such that p is correct.*

Proof. $\forall i \in \{1, \ldots, k\}, |M_i| = n - f$ 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 4: $\mathsf{read}'() \to \mathcal{L}(\mathbb{R} \times \mathsf{prove}(\mathbb{R}))$

```
4.1 j ← the process invoking read'();
4.2 res ← ∅;
4.3 forall i ∈ {1, ..., k} do
4.4     res ← res ∪ DL_i.read();
4.5 return res;
```

Algorithm 5: $\mathsf{append}'(\sigma) \to ()$

```
5.1 j ← the process invoking append'(σ);
5.2 forall M_i ∈ {M_k ∈ M : j ∈ M_k} do
5.3     DL_i.append(σ);
```

Algorithm 6: $\mathsf{prove}'(\sigma) \to \{0, 1\}$

```
6.1 j ← the process invoking prove'(σ);
6.2 flag ← false;
6.3 forall i ∈ {1, ..., k} do
6.4     flag ← flag OR DL_i.prove(σ);
6.5 return flag;
```

6.2 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 $pkc = pk_1, \ldots, pk_n$
- $S'(sk_i, m) \to \sigma_m^i$
- $C'(pkc, m_1, J, \{\sigma_{m_2}^j\}_{j \in J}) \to \sigma$: with $J \subseteq \Pi$; and $\sigma = \sigma_{m_1}$ iff $|J| \geq t, \forall j \in J : V(pk_j, m_1, \sigma_{m_2}^j) == 1$; 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.