MODULE *NoEquivocationVDF*

Consider the Gorilla model. We want to implement a broadcast abstraction such that, each round, each well-behaved player receives more messages from well-behaved players than from ill-behaved players.

We require that every message m carry a VDF evaluation $VDF(m)$ and we assume that well-behaved players discount any messages unless it has a correct VDF evaluation (such messages are considered invalid).

However, just requiring that every message m carry a correct VDF evaluation $VDF(m)$ is not enough because ill-behaved players could for example pre-compute VDF evaluations in one round to later use them in the next round in order to overwhelm the well-behaved players.

To prevent this, we additionally require that each message include a set of (valid) messages from the previous round and that this set be big enough to ensure that it includes at least one message from a well-behaved player. This ensures that ill-behaved players cannot pre-compute VDF outputs. The trick is to figure out how to ensure that a set of messages contains at least one message from a well-behaved player. This is what the algorithm below does.

EXTENDS *Integers*, *FiniteSets*

CONSTANTS

P the set of players (could be infinite)

--algorithm *NoEquivocation* **{**

variables

$msgs = [r \in Nat \mapsto [p \in P \mapsto \{\}]]$ **;** messages sent to each process each round

$round = 0$ **;** current round

$done = [p \in P \mapsto -1]$ **;** highest round in which each process participated

macro *SendAll*(m) **{**

$msgs[round] := [p \in P \mapsto msgs[round][p] \cup \{m\}]$ **;**

}

define {

if the following is true (where it's used below),

then we know that *msgs* contains a message from a well-behaved player

$ValidSet(msgs, recvd) \triangleq$

msgs is a set of messages from the previous round

recvd is what we received in the current round

in short: there is a majority among *msgs* that is a majority of a majority among *recvd*

$$
\begin{aligned}
&\exists S \in \text{SUBSET } msgs : \\
&\quad \land 2 * Cardinality(S) > Cardinality(msgs) \\
&\quad \land \exists R \in \text{SUBSET } recvd : \\
&\qquad \land 2 * Cardinality(R) > Cardinality(recvd) \\
&\qquad \land \forall r \in R : \\
&\qquad\quad \land S \subseteq r[2] \quad \text{r[2] is the set of messages attached to } r \\
&\qquad\quad \land 2 * Cardinality(S) > Cardinality(r[2])
\end{aligned}
$$

}

```
  process ( proc ∈ P )
      variables
          delivered = [r ∈ Nat ↦ {}] ;   delivered broadcast messages
  {
l0:     SendAll(⟨self⟩) ;
        done[self] := 0    ;   done for round 0
l1:     await round = 1 ;
         now deliver for round 0
         we can deliver everything since the adversary cannot precompute VDF outputs before round 0
        delivered[0] := msgs[0][self] ;
        SendAll(⟨self, delivered[0]⟩) ;   we attach all the delivered messages
        done[self] := 1 ;   done for round 1
l2:     while ( TRUE ) {
            await round = done[self] + 1 ;
            delivered[round − 1] := {m ∈ msgs[round − 1][self] : ValidSet(m[2], msgs[round − 1][self])}
            SendAll(⟨self, delivered[round − 1]⟩) ;   we attach all the messages delivered for the previous round
            done[self] := round ;
        }
   }
  process ( clock ∈ {"clock"} ) {
l0:    while ( TRUE ) {
            await ∀ p ∈ P : done[p] = round ;
            round := round + 1 ;
        }
   }
}
```