MODULE *NoEquivocation*

EXTENDS *Naturals*, *FiniteSets*, *Utilities*, *TLC*

CONSTANT P, V, *Lambda*, *Bot*

ASSUME $Distinct(\langle P,\ V,\ \{Bot\},\ \{Lambda\}\rangle)$

--algorithm *NoEquivocation* **{**
variables
$input \in [P \to V]$ **;**
$sent = [p \in P \mapsto Bot]$ **;** messages sent
$received = [p \in P \mapsto [q \in P \mapsto Bot]]$ **;** message received by p from q
$rnd = 1$ **;** $1 \,..\, 3$
$output = [p \in P \mapsto [q \in P \mapsto Bot]]$ **;** $output[p][q]$ is the message that p simulates receiving from q
$participating = [r \in \{1, 2\} \mapsto \{\}]$ **;**
$corrupted = \{\}$ **;**
define {
$TypeOkay \triangleq$
$\land\ input \in [P \to V]$
$\land\ sent \in [P \to [P \to V \cup \{Bot\}] \cup V \cup \{Bot\}]$
$\land\ received \in [P \to [P \to [P \to V \cup \{Bot\}] \cup V \cup \{Bot\}]]$
$\land\ rnd \in \{1, 2, 3\}$
$\land\ output \in [P \to [P \to \{Bot, Lambda\} \cup V]]$
$\land\ participating \in [\{1, 2\} \to \text{SUBSET } P]$
$\land\ corrupted \quad \in \text{SUBSET } P$
$HeardOf(p) \triangleq \{q \in P : received[p][q] \neq Bot\}$ heard of in the current round
$Minority(S) \triangleq \{M \in \text{SUBSET } S : 2 * Cardinality(M) < Cardinality(S)\}$
$NumHeardOf(p1,\ p2) \triangleq$ number of processes that report to $p1$ hearing from $p2$:
$Cardinality(\{q \in P : received[p1][q] \neq Bot \land received[p1][q][p2] \neq Bot\})$
$NumHeardValue(p1,\ p2,\ v) \triangleq$ number of processes that report to $p1$ hearing v from $p2$:
$Cardinality(\{q \in P : received[p1][q] \neq Bot \land received[p1][q][p2] = v\})$
$ValidOutput(p1,\ p2,\ v) \triangleq$
$\land\ 2 * NumHeardValue(p1,\ p2,\ v) > Cardinality(HeardOf(p1))$
$\land\ \forall\, q \in P : received[p1][q] \neq Bot \land received[p1][q][p2] \neq Bot \Rightarrow received[p1][q][p2] = v$
$Output(p1,\ p2) \triangleq$
IF $\exists\, v \in V : ValidOutput(p1,\ p2,\ v)$ true for at most one value v
THEN CHOOSE $v \in V : ValidOutput(p1,\ p2,\ v)$
ELSE
IF $\exists\, q \in P : received[p1][q] \neq Bot \land received[p1][q][p2] \neq Bot$
THEN *Lambda*
ELSE *Bot*
$SimulatedParticipants \triangleq \{p \in P : \exists\, q \in P : output[q][p] \neq Bot\}$
$CorrectSimulatedParticipants \triangleq participating[1] \setminus corrupted$
Now we define the correctness properties of the algorithm:
$NoEquivocation \triangleq \forall\, p1,\ p2,\ q \in P :$

```
        output[p1][q] ∈ V ∧ pc[p2] = "Done" ⇒ output[p2][q] ∈ {output[p1][q], Lambda}
    NoTampering ≜ ∀ p, q ∈ P :
        ∧ p ∈ CorrectSimulatedParticipants
        ∧ pc[q] = "Done"
        ⇒ output[q][p] = input[p]
    MinorityCorruption ≜ (∀ p ∈ P : pc[p] = "Done") ⇒
        2 * Cardinality(CorrectSimulatedParticipants) > Cardinality(SimulatedParticipants)
    Correctness ≜ NoEquivocation ∧ NoTampering ∧ MinorityCorruption
 }
 macro broadcast( v ) {
    sent := [sent EXCEPT ![self] = v]
 }
```

We now specify the simulation algorithm:

```
 process ( proc ∈ P ) {
r1:  broadcast(input[self]) ;
r2:  await rnd = 2 ;
     broadcast(received[self]) ;
r3:  await rnd = 3 ;
     output[self] := [p ∈ P ↦ Output(self, p)] ;
 }
```

Below we specify the behavior of the adversary.

```
 process ( adversary ∈ {"adversary"} ) {
a1:  await ∀ p ∈ P : pc[p] = "r2" ;
     pick a participating set:
     with ( Participating ∈ SUBSET P ) {
        when Participating ≠ {} ;
        participating[rnd] := Participating ;
     } ;
     pick a set whose messages will be tampered with:
     with ( Corrupted ∈ Minority(participating[1]) )
        corrupted := Corrupted ;
     tamper with the messages:
     with ( ByzVal ∈ [corrupted → [P → V ∪ {Bot}]] ) {
           received := [p ∈ P ↦ [q ∈ P ↦
              IF q ∈ corrupted
               in round 1, the adversary can make up any value:
               THEN ByzVal[q][p]
               ELSE
                  IF q ∈ participating[rnd] \ corrupted
                   THEN sent[q]
                   ELSE Bot]] ;
     } ;
     rnd := 2 ;
a2:  await ∀ p ∈ P : pc[p] = "r3" ;
```

```
        with ( Participating ∈ SUBSET P ) {
            when Participating ≠ {} ;
            when corrupted ∈ Minority(Participating) ;
            participating[2] := Participating ;
         } ;
         uncomment the following four lines to obtain a counter-example to MinorityCorruption under a growing adversary:
         with (Corrupted ∈ Minority(participating[2])) {
            when corrupted ⊆ Corrupted;
            corrupted := Corrupted
         };
        with ( ByzVal ∈ [corrupted → [P → [P → V ∪ {Bot}] ∪ {Bot}]] ) {
              In round 2, the adversary can only lie by omission about non-corrupted processes (because of signatures):
            when ∀ p1 ∈ P : ∀ q ∈ corrupted : ∀ p2 ∈ (P \ corrupted) :
                IF ByzVal[q][p1] ≠ Bot
                 THEN
                    IF p2 ∈ participating[1]
                     either the adversary reports not hearing from p2, or it reports p2's true input:
                     THEN ByzVal[q][p1][p2] ∈ {Bot, input[p2]}
                     ELSE ByzVal[q][p1][p2] = Bot
                 ELSE TRUE ;
            received := [p ∈ P ↦ [q ∈ P ↦
                IF q ∈ corrupted
                 THEN ByzVal[q][p]
                 ELSE
                    IF q ∈ participating[rnd] \ corrupted
                     THEN sent[q]
                     ELSE Bot]] ;
         } ;
        rnd := 3 ;
    }
}
```