MODULE $VDFConsensus$

EXTENDS $FiniteSets$, $Integers$

CONSTANTS

P the set of processes

, B the set of malicious processes

, $tAdv$ the time it takes for a malicious process to produce a message

, tWB the time it takes for a well-behaved process to produce a message

ASSUME $B \subseteq P$ malicious processes are a subset of all processes

$W \triangleq P \setminus B$ the set of well-behaved processes

$Tick \triangleq Nat$ a tick is a real-time clock tick

$Round \triangleq Nat$ a round is just a tag on a message

Processes build a DAG of messages. The message-production rate of well-behaved processes is of 1 message per tWB ticks, and that of malicious processes is of 1 message per $tAdv$ ticks. We require that, collectively, well-behaved processes produce messages at a rate strictly higher than that of malicious processes.

ASSUME $Cardinality(W) * tAdv > Cardinality(B) * tWB$

$MessageID \triangleq Nat$

A message consists of a unique ID, a round number, and a coffer containing the IDs of a set of predecessor messages:

$Message \triangleq [sender : P,\ id : MessageID,\ round : Round,\ coffer : \text{SUBSET } MessageID]$

We will need the intersection of a set of sets:

RECURSIVE $Intersection(\_)$

$Intersection(Ss) \triangleq$

  CASE

    $Ss = \{\} \rightarrow \{\}$

  $\Box$ $\exists\, S \in Ss : Ss = \{S\} \rightarrow \text{CHOOSE } S \in Ss : Ss = \{S\}$

  $\Box$ OTHER $\rightarrow$

      LET $S \triangleq (\text{CHOOSE } S \in Ss : \text{TRUE})$

      IN $S \cap Intersection(Ss \setminus \{S\})$

A set of messages is consistent when the intersection of the sets of predecessors of each message is a strict majority of the predecessors of each message.

$ConsistentSet(M) \triangleq$

  LET $I \triangleq Intersection(\{m.coffer : m \in M\})$

  IN $\forall\, m \in M : 2 * Cardinality(I) > Cardinality(m.coffer)$

A consistent chain is a subset of the messages in the DAG that potentially has some dangling pointers ($i.e.$ messages that have predecessors not in the chain) and that satisfies the following recursive predicate:

* Any set of messages which all have a round of 0 is a consistent chain.

* A set of messages C with some non-zero rounds and maximal round r is a consistent chain when, with *Tip* being the set of messages in the chain that have round r and *Pred* being the set of messages in the chain with round $r-1$, *Pred* is a strict majority of the set of predecessors of each message in *Tip* and $C \setminus Tip$ is a consistent chain. (Note that this implies that *Tip* is a consistent set)

$Max(X, Leq(\_, \_)) \triangleq$
$\quad$ CHOOSE $m \in X : \forall x \in X : Leq(x, m)$

RECURSIVE $ConsistentChain(\_)$
$ConsistentChain(M) \triangleq$
$\quad$ IF $M = \{\}$
$\quad$ THEN FALSE
$\quad$ ELSE LET $r \triangleq Max(\{m.round : m \in M\}, \leq)$ IN
$\quad\quad \lor\ r = 0$
$\quad\quad \lor$ LET $Tip \triangleq \{m \in M : m.round = r\}$
$\quad\quad\quad\quad Pred \triangleq \{m \in M : m.round = r - 1\}$
$\quad\quad\quad$ IN $\land\ \forall m \in Tip :$
$\quad\quad\quad\quad\quad \land\ Pred \subseteq m.coffer$
$\quad\quad\quad\quad\quad \land\ 2 * Cardinality(Pred) > Cardinality(m.coffer)$
$\quad\quad\quad\quad \land\ ConsistentChain(M \setminus Tip)$

Given a message DAG, the heaviest consistent chain is a consistent chain in the DAG that has a maximal number of messages.

$HeaviestConsistentChain(M) \triangleq$
$\quad$ LET $r \triangleq Max(\{m.round : m \in M\}, \leq)$
$\quad\quad Cs \triangleq \{C \in$ SUBSET $M : ConsistentChain(C)\}$
$\quad$ IN
$\quad\quad$ IF $Cs = \{\}$ THEN $\{\}$
$\quad\quad$ ELSE $Max(Cs,$ LAMBDA $C1, C2 : Cardinality(C1) \leq Cardinality(C2))$

Now we specify the algorithm

--algorithm $Algo$**{**
variables
$\quad messages = \{\}$ **;**
$\quad tick = 0$ **;**
$\quad pendingMessage = [p \in P \mapsto \langle\rangle]$ **;**
$\quad doneTick = [p \in P \mapsto -1]$ **;**
$\quad messageCount = 0$ **;** used to generate unique message *IDs*
define {
$\quad currentRound(t) \triangleq tick \div t$
$\quad wellBehavedMessages \triangleq \{m \in messages : m.sender \in P \setminus B\}$
$\quad$ possible sets of messages received by a well-behaved process:
$\quad receivedMsgsSets \triangleq$ LET $msgs \triangleq \{m \in messages : m.round < tick\}$ IN
$\quad\quad \{wellBehavedMessages \cup byzMsgs :$
$\quad\quad\quad byzMsgs \in$ SUBSET $(msgs \setminus wellBehavedMessages)\}$

```
    }
    macro sendMessage( m ) {
        messages := messages ∪ {m}
    }
    process ( clock ∈ {"clock"} ) {
tick:   while ( TRUE ) {
            wait for all processes to take their step before incrementing the tick
            await ∀ p ∈ P : doneTick[p] = tick ;
            tick := tick + 1 ;
        }
    }
    process ( proc ∈ P \ B )   a well-behaved process
    {
l1:     while ( TRUE ) {
            await tick > doneTick[self] ;
            if ( tick%tWB = 0 ) {
                Start the VDF computation for the next message:
                with ( msgs ∈ receivedMsgsSets )
                with ( predMsgs = {m ∈ msgs : m.round = currentRound(tWB) − 1} ) {
                    TODO: filter messages
                    pendingMessage[self] := [
                        sender ↦ self,
                        id ↦ messageCount + 1,
                        round ↦ currentRound(tWB),
                        coffer ↦ {m.id : m ∈ predMsgs}] ;
                    messageCount := messageCount + 1 ;
                }
            }
            else
            if ( tick%tWB = tWB − 1 )
                it's tWB − 1 because we want the message to be received by tick tWB
                sendMessage(pendingMessage[self]) ;
            else skip ;   busy computing the VDF
            doneTick[self] := tick ;
        }
    }
    process ( byz ∈ B )   a malicious process
    {
lb1:    while ( TRUE ) {
            await tick > doneTick[self] ;
            if ( tick%tAdv = 0 ) {
                Start the VDF computation for the next message:
                with ( msgs ∈ receivedMsgsSets )
                with ( rnd ∈ 0 . . currentRound(tAdv) )   can forge messages from any previous round
                with ( predMsgs = {m ∈ msgs : m.round = rnd − 1} ) {
```

$pendingMessage[self] := [$
$\quad sender \mapsto self,$
$\quad id \mapsto messageCount + 1,$
$\quad round \mapsto rnd,$
$\quad coffer \mapsto \{m.id : m \in predMsgs\}]\,\mathbf{;}$
$messageCount := messageCount + 1\,\mathbf{;}$
}
}
else
if ($tick \% tAdv = tAdv - 1$ **)**
$\quad sendMessage(pendingMessage[self])\,\mathbf{;}$
else skip ; busy computing the *VDF*
$doneTick[self] := tick\,\mathbf{;}$
} ;
}
}

$TypeOK \triangleq$
$\quad \land\ messages \in \text{SUBSET}\ Message$
$\quad \land\ pendingMessage \in [P \rightarrow Message \cup \{\langle\rangle\}]$
$\quad \land\ tick \in Tick$
$\quad \land\ doneTick \in [P \rightarrow Tick \cup \{-1\}]$

$messageWithID(id) \triangleq \text{CHOOSE}\ m \in messages : m.id = id$

$Inv1 \triangleq \forall\, m \in messages : \forall\, id \in m.coffer :$
$\quad \land\ \exists\, m2 \in messages : m2.id = id$
$\quad \land\ messageWithID(id).round = m.round - 1$