

EXTENDS *FiniteSets*, *Integers*, *TLC*

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

*TODO: I think we're going to need $Cardinality(W) * tAdv > 2 * Cardinality(B) * tWB$*

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

$MessageID \triangleq P \times Nat$

$Message \triangleq [id : MessageID, round : Round, coffer : SUBSET MessageID]$

$sender(m) \triangleq m.id[1]$

We will need the intersection of a set of sets:

RECURSIVE $Intersection(-)$

$Intersection(Ss) \triangleq$

CASE

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

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

□ 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 coffers of each message is a strict majority of the coffer 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)

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

TODO this might be too restrictive: maybe we should only require that a subset of $Pred$ be a strict majority of the set of predecessors of each message in Tip

RECURSIVE $ConsistentChain(-)$

$ConsistentChain(M) \triangleq$
 $\wedge M \neq \{\}$
 $\wedge LET r \triangleq Max(\{m.round : m \in M\}, \leq) IN$
 $\vee r = 0$
 $\vee LET Tip \triangleq \{m \in M : m.round = r\}$
 $Pred \triangleq \{m \in M : m.round = r - 1\}$
 IN $\wedge Tip \neq \{\}$
 $\wedge \exists Maj \in SUBSET Pred :$
 $\wedge Maj \neq \{\}$
 $\wedge \forall m \in Tip :$
 $\wedge \forall m2 \in Maj : m2.id \in m.coffer$
 $\wedge 2 * Cardinality(Maj) > Cardinality(m.coffer)$
 $\wedge ConsistentChain(M \setminus Tip)$

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

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

Two chains are disjoint when there is a round in which they have no messages in common:

$DisjointChains(C1, C2) \triangleq$
 LET $r1 \triangleq Max(\{m.round : m \in C1\}, \leq)$
 $r2 \triangleq Max(\{m.round : m \in C2\}, \leq)$
 IN $\exists r \in Round :$
 $\wedge r \leq r1$
 $\wedge r \leq r2$
 $\wedge \{m \in C1 : m.round = r\} \cap \{m \in C2 : m.round = r\} = \{\}$

Now we specify the algorithm

--algorithm *Algo*{

```

variables
   $messages = \{\}$ ;
   $tick = 0$ ;
   $phase = \text{"start"}$ ; each tick has two phases: "start" and "end"
   $donePhase = [p \in P \mapsto \text{"end"}]$ ;
   $pendingMessage = [p \in P \mapsto \langle \rangle]$ ;
   $messageCount = [p \in P \mapsto 0]$ ;
define {
   $currentRound \triangleq tick \div tWB$  round of well-behaved processes
   $wellBehavedMessages \triangleq \{m \in messages : sender(m) \in P \setminus B\}$ 
    possible sets of messages received by a well-behaved process:
   $receivedMsgsSets \triangleq$ 
    ignore messages from future rounds:
    LET  $msgs \triangleq \{m \in messages : m.round < currentRound\}$  IN
     $\{wellBehavedMessages \cup byzMsgs :$ 
       $byzMsgs \in \text{SUBSET}(msgs \setminus wellBehavedMessages)\}$ 
    }
  macro  $sendMessage( m )$  {
     $messages := messages \cup \{m\}$ 
  }
  process (  $clock \in \{\text{"clock"}\}$  ) {
    tick: while ( TRUE ) {
      await  $\forall p \in P : donePhase[p] = phase$ ;
      if (  $phase = \text{"start"}$  )
         $phase := \text{"end"}$ 
      else {
         $phase := \text{"start"}$ ;
         $tick := tick + 1$ 
      }
    }
  }
  process (  $proc \in P \setminus B$  ) a well-behaved process
  {
l1: while ( TRUE ) {
    await  $phase = \text{"start"}$ ;
    if (  $tick \% tWB = 0$  ) {
      Start the VDF computation for the next message:
      with (  $msgs \in receivedMsgsSets$  )
      with (  $hCC = HeaviestConsistentChain(msgs)$  )
      with (  $predMsgs = \{m \in hCC : m.round = currentRound - 1\}$  ) {
         $pendingMessage[self] := [$ 
           $id \mapsto \langle self, messageCount[self] + 1 \rangle,$ 
           $round \mapsto currentRound,$ 
           $coffer \mapsto \{m.id : m \in predMsgs\}]$ ;
         $messageCount[self] := messageCount[self] + 1$ 
      }
    }
  }
}

```

```

    }
  } ;
  donePhase[self] := "start" ;
l2:  await phase = "end" ;
    if ( tick%tWB = tWB - 1 )
      it's the end of the tWB period, the VDF has been computed
      sendMessage(pendingMessage[self]) ;
      donePhase[self] := "end" ;
    }
  }
}
process ( byz ∈ B )  a malicious process
{
l1:  while ( TRUE ) {
    await phase = "start" ;
    if ( tick%tAdv = 0 ) {
      Start the VDF computation for the next message:
      with ( maxRound = Max({m.round : m ∈ messages} ∪ {0}, ≤) )
      with ( rnd ∈ {maxRound, maxRound + 1} )
      with ( predMsgs ∈ SUBSET {m ∈ messages : m.round = rnd - 1} ) {
        when rnd > 0 ⇒ predMsgs ≠ {} ;
        pendingMessage[self] := [
          id ↦ ⟨self, messageCount[self] + 1⟩,
          round ↦ rnd,
          coffer ↦ {m.id : m ∈ predMsgs} ] ;
        messageCount[self] := messageCount[self] + 1
      }
    } ;
    donePhase[self] := "start" ;
l2:  await phase = "end" ;
    if ( tick%tAdv = tAdv - 1 )
      sendMessage(pendingMessage[self]) ;
      donePhase[self] := "end" ;
    } ;
  }
}
TypeOK ≜
  ∧ messages ∈ SUBSET Message
  ∧ pendingMessage ∈ [P → Message ∪ {⟨⟩}]
  ∧ tick ∈ Tick
  ∧ phase ∈ {"start", "end"}
  ∧ donePhase ∈ [P → {"start", "end"}]
  ∧ messageCount ∈ [P → Nat]

```

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

$$\begin{aligned} \text{Safety} &\triangleq \forall m \in \text{messages} : m.\text{round} > 0 \wedge \text{sender}(m) \notin B \Rightarrow \\ &\quad \wedge \forall m2 \in \text{wellBehavedMessages} : m2.\text{round} = m.\text{round} - 1 \Rightarrow m2.\text{id} \in m.\text{coffer} \\ &\quad \wedge \text{LET } M \triangleq \{m2 \in \text{wellBehavedMessages} : m2.\text{round} = m.\text{round} - 1\} \\ &\quad \text{IN } 2 * \text{Cardinality}(M) > \text{Cardinality}(m.\text{coffer}) \end{aligned}$$
$$\begin{aligned} \text{Inv1} &\triangleq \forall m \in \text{messages} : \\ &\quad \wedge \forall m2 \in \text{messages} : m \neq m2 \Rightarrow m.id \neq m2.id \\ &\quad \wedge \forall id \in m.\text{coffer} : \\ &\quad \quad \wedge \exists m2 \in \text{messages} : m2.id = id \\ &\quad \quad \wedge \text{messageWithID}(id).round = m.round - 1 \end{aligned}$$