

Permissionless Consensus using Verifiable Delay Functions

Giuliano Losa

December 5, 2023

Contents

| | | |
|---|---|----------|
| 1 | Messages | 1 |
| 1.1 | The depth of a message | 1 |
| 1.2 | Messages that the adversary can forge | 2 |
| 2 | The model | 4 |
| theory <i>VDFConsensus</i> | | |
| imports <i>Main HOL-Library.FSet HOL-Statespace.StateSpaceSyntax HOL-Library.State-Monad</i> | | |
| begin | | |

1 Messages

First we define a datatype of messages.

```
datatype 'a msg =  
  Val 'a — A guessable value  
| Nonce nat — A non-guessable nonce  
| VDF 'a msg — The VDF applied to a given message  
| MSet ('a msg) fset — A (finite) set of messages packed together in one message  
| MPair 'a msg 'a msg — A pair of two messages
```

1.1 The depth of a message

The depth of a message is the length of the longest VDF chain appearing in the message.

```
primrec depth :: 'a msg  $\Rightarrow$  nat where  
  depth (Val x) = 0  
| depth (Nonce n) = 0  
| depth (VDF m) = depth m + 1  
| depth (MSet s) = (if s = {} then 0 else (fMax (fimage depth s))) — This is the  
max depth of messages in the set s  
| depth (MPair m1 m2) = Orderings.max (depth m1) (depth m2)
```

lemma *depth-MSet-1*: $m \in \text{fset } s \implies \text{depth } m \leq \text{depth } (\text{MSet } s)$
by *auto*

lemma *depth-MSet-2*: $s \neq \{\}\implies \exists m . m \in \text{fset } s \wedge \text{depth } (\text{MSet } s) = \text{depth } m$
by (*auto*; *metis fMax-in fempty-iff fempty-is-fimage fimageE*)

Example

lemma *depth* ($\text{VDF } (\text{MSet } \{ | \text{VDF } (\text{Val } 0), \text{VDF } (\text{VDF } (\text{Val } 42)) | \})$) = 3
by *simp*

1.2 Messages that the adversary can forge

inductive-set *parts* :: 'a msg set \Rightarrow 'a msg set **for** *msgs* **where**
— All the parts that can be extracted from the set of messages *msgs*
 $m \in \text{msgs} \implies m \in \text{parts } \text{msgs}$
 $| \text{MPair } m1 \ m2 \in \text{parts } \text{msgs} \implies m1 \in \text{parts } \text{msgs}$
 $| \text{MPair } m1 \ m2 \in \text{parts } \text{msgs} \implies m2 \in \text{parts } \text{msgs}$
 $| \llbracket \text{MSet } s \in \text{parts } \text{msgs}; m \in \text{fset } s \rrbracket \implies m \in \text{parts } \text{msgs}$

inductive-set *synth* :: 'a msg set \Rightarrow nat set \Rightarrow 'a msg set **for** *msgs* *nonces* **where**
— This is all the messages that the attacker can synthesize if it has the set of messages *msgs*, where *nonces* is the set of nonces that have already been used
 $m \in \text{parts } \text{msgs} \implies m \in \text{synth } \text{msgs } \text{nonces}$
 $| \text{Val } v \in \text{synth } \text{msgs } \text{nonces}$ — Values can be guessed
 $| n \notin \text{nonces} \implies \text{Nonce } n \in \text{synth } \text{msgs } \text{nonces}$
 $| \llbracket m1 \in \text{parts } \text{msgs}; m2 \in \text{parts } \text{msgs} \rrbracket \implies \text{MPair } m1 \ m2 \in \text{synth } \text{msgs } \text{nonces}$
 $| \forall m \in \text{fset } s . m \in \text{parts } \text{msgs} \implies \text{MSet } s \in \text{synth } \text{msgs } \text{nonces}$

definition *synth-vdf* **where**

— The messages that the adversary can synthesis if it can compute one VDF output

$\text{synth-vdf } \text{msgs } \text{nonces} \equiv \text{let } \text{syn} = \text{synth } \text{msgs } \text{nonces} \text{ in}$
 $\bigcup m \in \text{syn} . \text{synth } (\text{syn} \cup \{ \text{VDF } m \}) \text{ nonces}$

lemma *parts-depth*:

fixes *msgs d m*

assumes $\bigwedge m . m \in \text{msgs} \implies \text{depth } m \leq d$

and $m \in \text{parts } \text{msgs}$

shows $\text{depth } m \leq d$

using *assms(2)*

proof (*induct m*)

case (*1 m*)

then show *?case*

by (*simp add: assms(1)*)

next

case (*2 m1 m2*)

then show *?case*

by *simp*

next

```

    case (3 m1 m2)
    then show ?case by simp
next
    case (4 s m)
    then show ?case
      using depth-MSet-1 by fastforce
qed

```

Main lemma: the adversary cannot forge a message that has larger depth than any message it already has.

```

lemma synth-depth:
  fixes msgs d m nonces
  assumes  $\bigwedge m . m \in \text{msgs} \implies \text{depth } m \leq d$ 
    and  $m \in \text{synth } \text{msgs } \text{nonces}$ 
  shows  $\text{depth } m \leq d$ 
  using assms(2)
proof (induct m)
  case (1 m)
  then show ?case
    using assms(1) parts-depth by auto
next
  case (2 v)
  then show ?case
    by simp
next
  case (3 n)
  then show ?case
    by auto
next
  case (4 m1 m2)
  then show ?case
    using assms(1) parts-depth by auto
next
  case (5 s)
  then show ?case
    by (metis assms(1) depth.simps(4) depth-MSet-2 less-nat-zero-code linorder-not-le parts-depth)
qed

```

```

lemma synth-vdf-depth:
  fixes msgs :: 'a msg set and m :: 'a msg and d :: nat and nonces
  assumes  $\bigwedge m . m \in \text{msgs} \implies \text{depth } m \leq d$ 
    and  $m \in \text{synth-vdf } \text{msgs } \text{nonces}$ 
  shows  $\text{depth } m \leq d+1$ 
proof -
  have  $\text{depth } m \leq d+1$  if  $m \in \text{synth } (\text{synth } \text{msgs } \text{nonces} \cup \{\text{VDF } m'\}) \text{ nonces}$ 
  and  $m' \in \text{synth } \text{msgs } \text{nonces}$  for  $m m'$ 
  proof -
    have  $\text{depth } m' \leq d$ 

```

```

    using assms(1) synth-depth that(2) by blast
  hence depth  $m'' \leq d+1$  if  $m'' \in \text{synth msgs nonces} \cup \{VDF\ m'\}$  for  $m''$ 
    by (metis Suc-eq-plus1 Un-iff assms(1) depth.simps(3) not-less-eq-eq single-
ton-iff synth-depth that trans-le-add1)
  thus ?thesis
    by (meson synth-depth that(1))
qed
thus ?thesis using assms(2)
  unfolding synth-vdf-def
  by (auto simp add:Let-def)
qed

```

2 The model

statespace ('a, 'p, 'o) *model-state* =

- 'p is the type of player IDs
- round* :: nat — The current round
- adv* :: 'p fset — The players controlled by the adversary in the current round
- wb* :: 'p fset — The well-behaved players in the current round
- vdf-processors* :: 'p \Rightarrow nat — How many parallel VDF processors a each participant has in the current round
- msgs* :: 'p \Rightarrow 'a msg fset — The mailbox of each player. TODO: probably need the sender.
- outputs* :: 'p \Rightarrow 'o
- adv-knowledge* :: 'a msg set — The information that the adversary collected, i.e. all messages ever sent
- prev-msgs* :: 'p \Rightarrow 'p \Rightarrow 'a msg fset — A history variable tracking the messages sent in the previous round
- nonces* :: nat set — set of nonces used

locale *model* = *model-state*

— Hack to fix type variable names; is there a better way? Note that a state has type 'name \Rightarrow 'value

where *project-'a-VDFConsensus-msg-FSet-fset-'p-fun=project-'a-VDFConsensus-msg-FSet-fset-'p-fun::'value*
 \Rightarrow 'p \Rightarrow 'a msg fset

and *inject-'o-'p-fun=inject-'o-'p-fun::('p \Rightarrow 'o) \Rightarrow -*

and *round=round::'name* **for** *project-'a-VDFConsensus-msg-FSet-fset-'p-fun*
inject-'o-'p-fun *round* +

fixes *send-fn* :: nat \Rightarrow 'a msg fset \Rightarrow 'a msg — Determines what message a well-behaved player sends each round, as a function of the messages it receives

and *out* :: nat \Rightarrow 'a msg fset \Rightarrow 'o — Determines an output each round

begin

definition *vdf-assumptions* **where**

vdf-assumptions *s* \equiv

fsum (*s.vdf-processors*) (*s.adv*) < *fsum* (*s.vdf-processors*) (*s.wb*)

— Adversary has less VDF processors than well-behaved players have

\wedge (*s.wb*) $\neq \{\|\}$

— At least one well-behaved player

$\wedge fsum (s.vdf-processors) (s.wb) > 0$
 — Well-behaved players can compute at least 1 VDF output

definition *wb-msg* **where**

$wb-msg\ s\ p \equiv let\ n = SOME\ nonce.\ nonce \notin (s.nonces); m = VDF\ (MPair\ (Nonce\ n)\ (MSet\ ((s.msgs)\ p)))\ in$

— We just pack all the messages received along with a fresh nonce in a VDF
 $s.<nonces := (s.nonces) \cup \{n\}, msgs := \lambda\ p.\ (s.msgs)\ p\ |\cup|\ \{|m|\}>$

TODO: fix starting from here

definition *adv-msgs* **where**

— This is the possible set of messages that the adversary can send each round

$adv-msgs\ s \equiv \{msgs.\ \exists\ vdf-msgs.\$

$vdf-msgs \subseteq synth-vdf\ (s.adv-knowledge)\ (s.nonces)$

$\wedge card\ vdf-msgs \leq fsum\ (s.vdf-processors)\ (s.adv)$

$\wedge msgs = synth\ (s.adv-knowledge)\ (s.nonces) \cup vdf-msgs\}$

definition *init* **where**

$init\ s \equiv s.round = 1 \wedge s.msgs = (\lambda\ p.\ \{\}) \wedge vdf-assumptions\ s \wedge (s.adv-knowledge)$

$= \{\}$

$\wedge (s.nonces) = \{\}$

definition *next-round* **where**

$next-round\ s\ s' \equiv$

$s'.round = (s.round) + 1$

$\wedge vdf-assumptions\ s'$

$\wedge (\exists\ ms.\ (\forall\ p.\ fset\ (ms\ p) \subseteq synth-vdf\ (s.adv-knowledge)\ \{\}))$

$\wedge s'.msgs = (\lambda\ p.\ (s.msgs)\ p\ |\cup|\ ms\ p))$

$\wedge s'.outputs = (\lambda\ p.\ out\ (s.round)\ ((s.msgs)\ p))$

$\wedge s'.adv-knowledge = (s.adv-knowledge) \cup (\bigcup\ p.\ fset\ ((s'.msgs)\ p))$

end

end