# Permissionless Consensus using Verifiable Delay Functions

Giuliano Losa

December 5, 2023

## Contents



**theory** *VDFConsensus*
**imports** *Main HOL−Library.FSet HOL−Statespace.StateSpaceSyntax*
**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* ⇒ *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*: $m \in \mathit{fset}\ s \Longrightarrow \mathit{depth}\ m \leq \mathit{depth}\ (\mathit{MSet}\ s)$
**by** *auto*

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

Example

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

## 1.2 Messages that the adversary can forge

**inductive-set** *parts* :: *'a msg set* ⇒ *'a msg set* **for** *msgs* **where**
— All the parts that can be extracted from the set of messages *msgs*
$m \in \mathit{msgs} \Longrightarrow m \in \mathit{parts}\ \mathit{msgs}$
| $\mathit{MPair}\ m1\ m2 \in \mathit{parts}\ \mathit{msgs} \Longrightarrow m1 \in \mathit{parts}\ \mathit{msgs}$
| $\mathit{MPair}\ m1\ m2 \in \mathit{parts}\ \mathit{msgs} \Longrightarrow m2 \in \mathit{parts}\ \mathit{msgs}$
| $[\![\mathit{MSet}\ s \in \mathit{parts}\ \mathit{msgs};\ m \in \mathit{fset}\ s]\!] \Longrightarrow m \in \mathit{parts}\ \mathit{msgs}$

**inductive-set** *synth* :: *'a msg set* ⇒ *nat set* ⇒ *'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
$m \in \mathit{parts}\ \mathit{msgs} \Longrightarrow m \in \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}$
| $\mathit{Val}\ v \in \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}$ — Values can be guessed
| $n \notin \mathit{nonces} \Longrightarrow \mathit{Nonce}\ n \in \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}$
| $[\![m1 \in \mathit{parts}\ \mathit{msgs};\ m2 \in \mathit{parts}\ \mathit{msgs}]\!] \Longrightarrow \mathit{MPair}\ m1\ m2 \in \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}$
| $\forall\ m \in \mathit{fset}\ s\ .\ m \in \mathit{parts}\ \mathit{msgs} \Longrightarrow \mathit{MSet}\ s \in \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}$

**definition** *synth-vdf* **where**
— The messages that the adversary can synthesis if it can compute one VDF output
$\mathit{synth\text{-}vdf}\ \mathit{msgs}\ \mathit{nonces} \equiv \mathit{let}\ \mathit{syn} = \mathit{synth}\ \mathit{msgs}\ \mathit{nonces}\ \mathit{in}$
$\bigcup\ m \in \mathit{syn}\ .\ \mathit{synth}\ (\mathit{syn} \cup \{\mathit{VDF}\ m\})\ \mathit{nonces}$

**lemma** *parts-depth*:
**fixes** *msgs d m*
**assumes** $\bigwedge\ m\ .\ m \in \mathit{msgs} \Longrightarrow \mathit{depth}\ m \leq d$
**and** $m \in \mathit{parts}\ \mathit{msgs}$
**shows** $\mathit{depth}\ m \leq d$
**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** $\bigwedge m .\ m \in msgs \Longrightarrow depth\ m \leq d$
**and** $m \in synth\ msgs\ nonces$
**shows** $depth\ m \leq d$
**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** $\bigwedge m .\ m \in msgs \Longrightarrow depth\ m \leq d$
**and** $m \in synth\text{-}vdf\ msgs\ nonces$
**shows** $depth\ m \leq d{+}1$
**proof** −
**have** $depth\ m \leq d{+}1$ **if** $m \in synth\ (synth\ msgs\ nonces \cup \{VDF\ m'\})\ nonces$ **and** $m' \in synth\ msgs\ nonces$ **for** $m\ m'$
**proof** −
**have** $depth\ m' \leq d$

**using** *assms*(*1*) *synth-depth that*(*2*) **by** *blast*
**hence** *depth* $m'' \leq d+1$ **if** $m'' \in$ *synth msgs nonces* $\cup$ {*VDF* $m'$} **for** $m''$
**by** (*metis Suc-eq-plus1 Un-iff assms*(*1*) *depth.simps*(*3*) *not-less-eq-eq singleton-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* ⇒ *nat* — How many parallel VDF processors a each participant has in the current round
*msgs* :: *'p* ⇒ *'a msg fset* — The mailbox of each player. TODO: probably need the sender.
*outputs* :: *'p* ⇒ *'o*
*adv-knowledge* :: *'a msg set* — The information that the adversary collected, i.e. all messages ever sent
*prev-msgs* :: *'p* ⇒ *'p* ⇒ *'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* ⇒ *'value*
**where** *project-'a-VDFConsensus-msg-FSet-fset-'p-fun*=*project-'a-VDFConsensus-msg-FSet-fset-'p-fun*::*'valu*
⇒ *'p* ⇒ *'a msg fset*
**and** *inject-'o-'p-fun*=*inject-'o-'p-fun*::(*'p* ⇒ *'o*) ⇒ -
**and** *round*=*round*::*'name* **for** *project-'a-VDFConsensus-msg-FSet-fset-'p-fun inject-'o-'p-fun round* +
**fixes** *send-fn* :: *nat* ⇒ *'a msg fset* ⇒ *'a msg* — Determines what message a well-behaved player sends each round, as a function of the messages it receives
**and** *out* :: *nat* ⇒ *'a msg fset* ⇒ *'o* — Determines an output each round
**begin**

**definition** *vdf-assumptions* **where**
*vdf-assumptions s* ≡
*fsum* (*s*·*vdf-processors*) (*s*·*adv*) < *fsum* (*s*·*vdf-processors*) (*s*·*wb*)
— Adversary has less VDF processors than well-behaved players have
∧ (*s*·*wb*) ≠ {||}
— 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* ≡ *let n* = *SOME nonce . nonce* ∉ (*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*) ∪ {*n*}, *msgs* := λ *p* . (*s*·*msgs*) *p* |∪| {|*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* ≡ {*msgs* . ∃ *vdf-msgs* .
*vdf-msgs* ⊆ *synth-vdf* (*s*·*adv-knowledge*) (*s*·*nonces*)
∧ *card vdf-msgs* ≤ *fsum* (*s*·*vdf-processors*) (*s*·*adv*)
∧ *msgs* = *synth* (*s*·*adv-knowledge*) (*s*·*nonces*) ∪ *vdf-msgs*}

**definition** *init* **where**
*init s* ≡ *s*·*round* = *1* ∧ *s*·*msgs* = (λ *p* . {||}) ∧ *vdf-assumptions s* ∧ (*s*·*adv-knowledge*) = {}
∧ (*s*·*nonces*) = {}

**definition** *next-round* **where**
*next-round s s'* ≡
*s'*·*round* = (*s*·*round*) + *1*
∧ *vdf-assumptions s'*
∧ (∃ *ms* . (∀ *p* . *fset* (*ms p*) ⊆ *synth-vdf* (*s*·*adv-knowledge*) {})
∧ *s'*·*msgs* = (λ *p* . (*s*·*msgs*) *p* |∪| *ms p*))
∧ *s'*·*outputs* = (λ *p* . *out* (*s*·*round*) ((*s*·*msgs*) *p*))
∧ *s'*·*adv-knowledge* = (*s*·*adv-knowledge*) ∪ (⋃ *p* . *fset* ((*s'*·*msgs*) *p*))

**end**

**end**