Verifiable Delay Functions

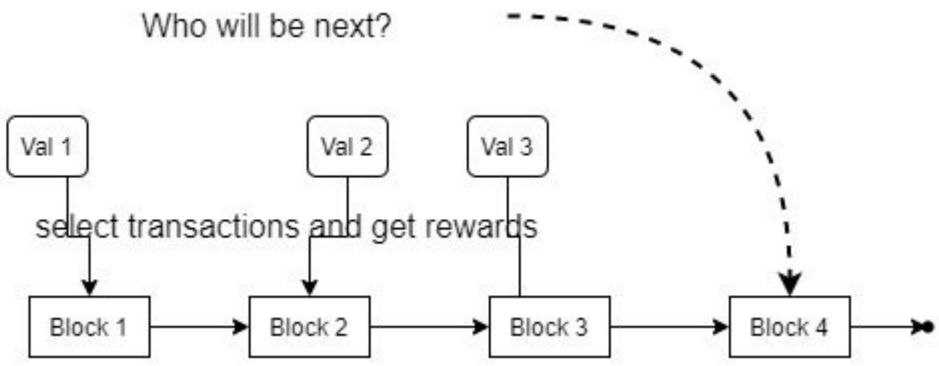
Dmitry Khovratovich

Ethereum Foundation
Cryptographic Frontier 2021

Distributed Reward Problem

1. Parties play a game and don't trust each other.
2. Winner should be selected uniformly at random.
3. Winner gets a reward.
4. Should be no way to bias the selection.

Example: Ethereum 2.0 validators decide who adds blocks in the next epoch.



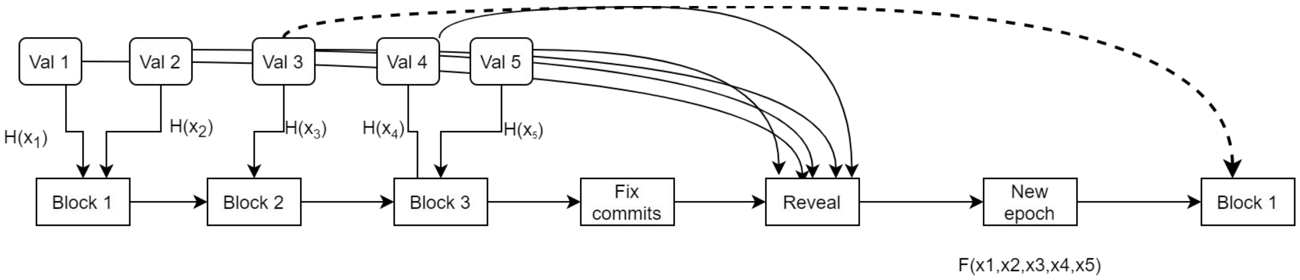
Solution at Ethereum 2.0: RanDAO

RanDAO solution:

1. Each validator commits a secret string: H(s)
2. Everyone reveals its input.
3. Hash of the result indicates a winner. H(s1,s2,...,sk)

Problem:

1. Withholding a reveal affects the result.
2. X malicious players have 2^X options.

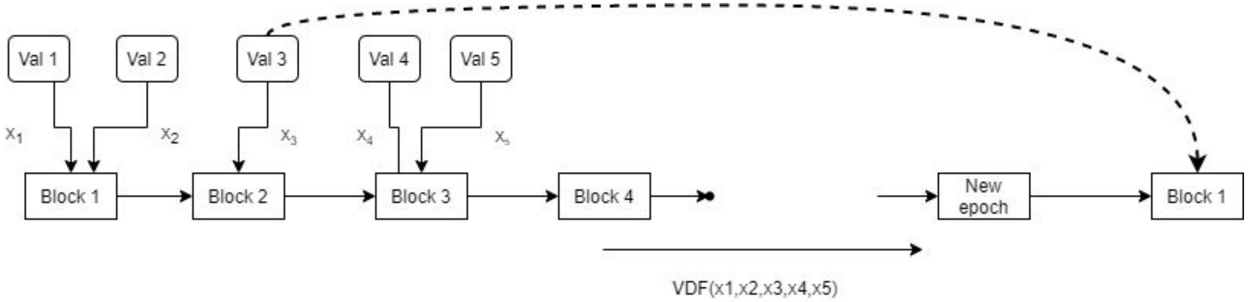


Perspective VDF in Ethereum

1. Validators submit entropy x1,x2,...xn
2. Evaluator E computes a VDF (Proof P, value V= H(x1,x2,...,xn)) for 3 days.
3. Everyone can check P and select validator X based on V.
4. Blocks for the next epoch (a few hours) are selected by X.

Requirements:

1) Tight latency -- no shortcuts
2) Minimum hardware for E
3) Succinct proofs
4) Quantum upgrade
5) Delay granularity
6) Non-malleability



Existing VDF constructions: RSA

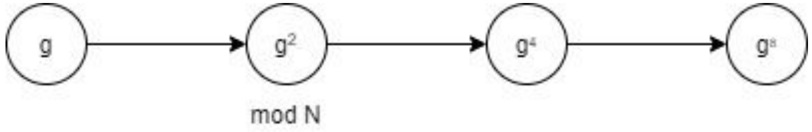
RSA VDF Setup: generate module N=pq so that *no one* knows p or q.

RSA VDF Run: select input I, make T squarings mod N, output O

RSA VDF Proof: certain intermediate values. Concretely,

1) L is random, given to Prover
2) Prover provides $Z=g^{(2^T) \text{ div } L}$
3) Verifier checks if $O=Z^L g^{(2^T) \% L}$

Needs trusted setup!

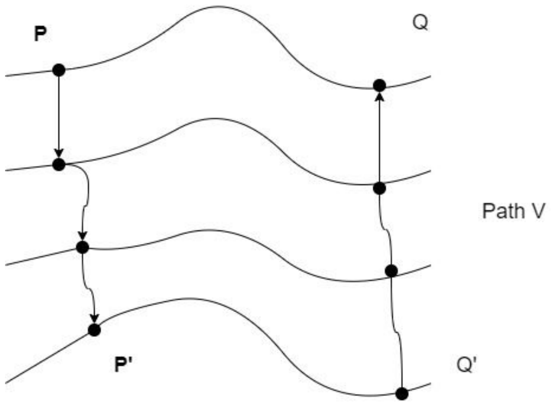


Existing VDF construction: Isogenies

1. Generate elliptic curve E and select point P.
2. Make T transformations of E to another curve, finally E’. Point P becomes P’.
3. Prover selects Q’ on E’ and has to revert the same transform (V) to get Q.
4. It must hold that e(P’,Q’) = e(P,Q)

Problem:

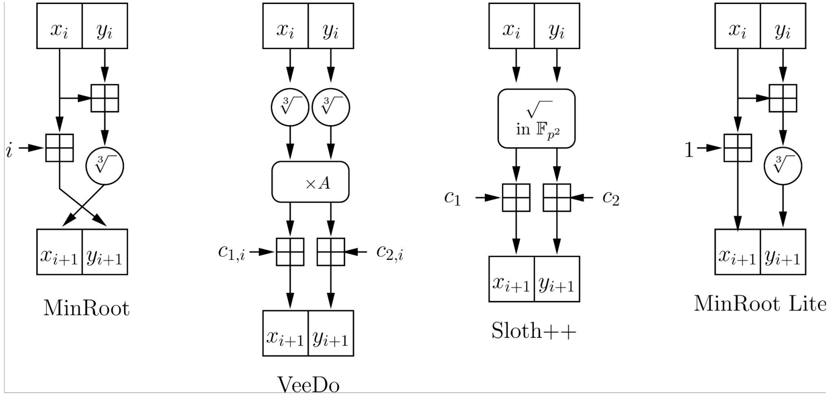
1. The path V requires a lot of storage (GiB/seconds)
2. Can be alleviated if V is pseudo-random but only for one run.

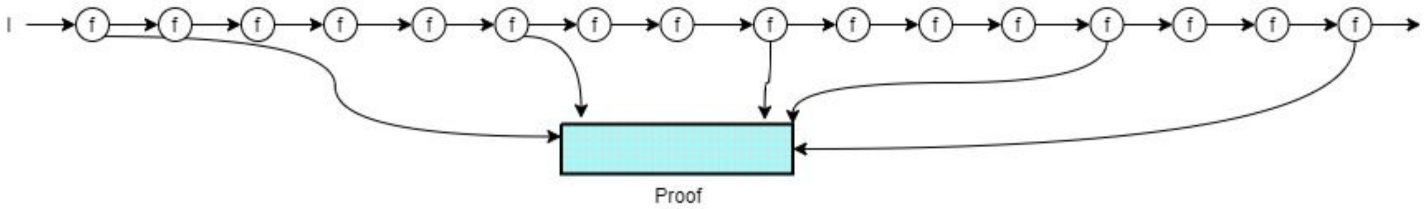


IVC-based VDF: Sloth, MinRoot, VeeDo

1. Select some iterative transformation F.
2. Apply F to input I 2^T times and produce O.
3. Prove that I->O.

F is chosen to be SNARK-friendly and hardware-minimal.



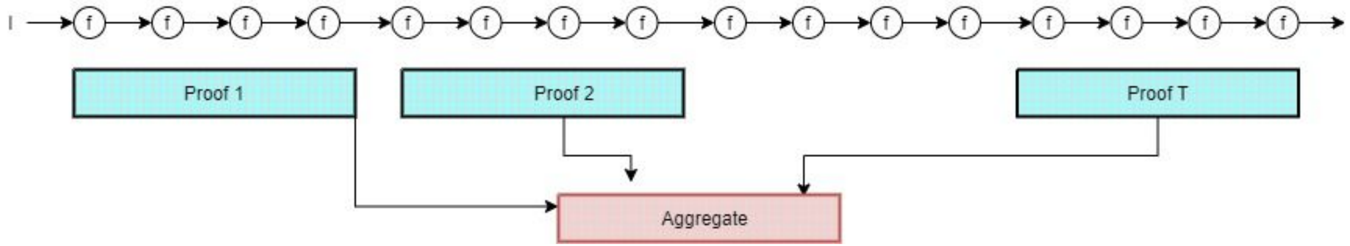


IVC-based VDF: Sloth, MinRoot, VeeDo

SnarkPack:

1. Split VDF in X parts
2. Prove each part when it is ready.
3. Aggregate when done.
4. Only 1% overhead.

Fastest provers are not post-quantum but can be upgraded



VDF and Delay Encryption

Delay Encryption works as follows:

1) Protector calls VDF F to derive secret key K=F(x).
2) Protector encrypts data D on K: C= E_K(D) along with proof of correctness.
3) Protector sends D and x to contract S.
4) Someone recomputes K and everyone can decrypt D.

Open Problems

1. Post-quantum VDF without trusted setup and low overhead
2. Isogeny-based VDF with constant storage
3. Simple and secure trusted setup for RSA-based VDF.
4. Latency upper and lower bounds for modular multiplications and exponentiations.
5. Quantum precomputation attacks on VDF