Bonus Lecture #6: Secret Leader Selection and Verifiable Random Functions

COMS 4995-001: The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Bonus Lecture #6

1. Verifiable random functions (VRFs).
   - how validators can obtain private and verifiable (pseudo)randomness
2. Using VRFs for leader selection in proof-of-stake Tendermint.
   - issues to address include unpredictable number of leaders selected
3. Pseudorandomness beacons.
   - goal: derive unmanipulatable pseudorandomness from blockchain state
4. Randomly sampled committees.
   - scaling Tendermint-style protocols up to 1000s of validators

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution:

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Good news: relatively simple, no fancy cryptography.

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Good news: relatively simple, no fancy cryptography.

Bad news: leaders of future views known well in advance.

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

Good news: relatively simple, no fancy cryptography.

Bad news: leaders of future views known well in advance.

- ➔ risk of bribery, coercion, DoS attacks
- also has its benefits (e.g., for tx dissemination)

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's.

Solution: use epoch of length N views each (N large).

Good news: relatively simple, no fancy cryptography.

Bad news: leaders of future views known well in advance.

Question: secret (and verifiable) leader selection?

- a la Nakamoto consensus, but with proof-of-stake sybil-resistance

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

- for each view v, produces an independent and uniformly random output r_v (e.g., in $\{0,1\}^{256}$). [known to all validators without any communication]

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

- for each view v, produces an independent and uniformly random output r_v (e.g., in $\{0,1\}^{256}$). [known to all validators without any communication]

Question: why not use r_v to directly sample leader for view v (HW9)?

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

- for each view v, produces an independent and uniformly random output r_v (e.g., in $\{0,1\}^{256}$). [known to all validators without any communication]

Question: why not use r_v to directly sample leader for view v (HW9)?

- answer: not secret (leader known to all as soon as r_v drops)
  - even if it's "secret enough," can also use VRFs for other purposes

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

- for each view v, produces an independent and uniformly random output r_v

Idea:

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

– for each view v, produces an independent and uniformly random output r_v

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader of v ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA)

Secretly Selecting a Leader

Updated goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to q_i's, s.t. result is secret (but verifiable once reported).

Assume for now: existence of a "randomness beacon."

- for each view v, produces an independent and uniformly random output r_v

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader of v $\Leftrightarrow$ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA)
  - for sybil-proofness, threshold must be increasing in q_i

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader $\Leftrightarrow$ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader $\Leftrightarrow$ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader $\Leftrightarrow$ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

- by assumption, can't manipulate r_v (will revisit assumption later)

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

- by assumption, can't manipulate r_v (will revisit assumption later)
- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - i.e., look for a value of sk that makes $sig_{sk}(r_v)$ as small as possible

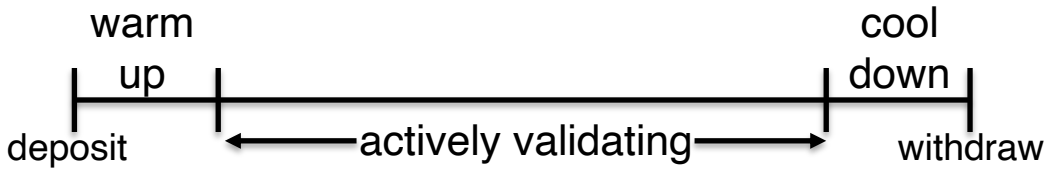
Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

- by assumption, can't manipulate r_v (will revisit assumption later)
- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - i.e., look for a value of sk that makes $sig_{sk}(r_v)$ as small as possible
  - fix: make warm-up period long enough that i commits to sk_i before seeing r_v



Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

  - by assumption, can't manipulate r_v (will revisit assumption later)
- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - fix: make warm-up period long enough that i commits to sk_i before seeing r_v
- issue #2:

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

  – by assumption, can't manipulate r_v (will revisit assumption later)

- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  – fix: make warm-up period long enough that i commits to sk_i before seeing r_v
- issue #2: given r_v and sk_i, could try multiple ("grind") $sig_{sk_i}(r_v)$'s
  – signatures not generally unique (e.g. nonces in Schnorr/ECDSA signatures)

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

  - by assumption, can't manipulate r_v (will revisit assumption later)
- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - fix: make warm-up period long enough that i commits to sk_i before seeing r_v
- issue #2: given r_v and sk_i, could try multiple ("grind") $sig_{sk_i}(r_v)$'s
  - signatures not generally unique (e.g. nonces in Schnorr/ECDSA signatures)
  - fix: use signature scheme with unique signatures (e.g., BLS)

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

  - by assumption, can't manipulate r_v (will revisit assumption later)
- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - fix: make warm-up period long enough that i commits to sk_i before seeing r_v
- issue #2: given r_v and sk_i, could try multiple ("grind") $sig_{sk_i}(r_v)$'s
  - fix: use signature scheme with unique signatures (e.g., BLS)
- issue #3: $sig_{sk_i}(r_v)$ may not be uniformly random (even if r_v is)

Preventing Grinding Over Signatures

Idea: each view v, each validator i computes $sig_{sk_i}(r_v)$.

- i is leader ⇔ $sig_{sk_i}(r_v)$ is "sufficiently small" (TBA, depends on q_i)

Question: can a validator influence its selection probability?

  - by assumption, can't manipulate r_v (will revisit assumption later)

- issue #1: given r_v, could try multiple (a.k.a. "grind") pk/sk pairs
  - fix: make warm-up period long enough that i commits to sk_i before seeing r_v
- issue #2: given r_v and sk_i, could try multiple ("grind") $sig_{sk_i}(r_v)$'s
  - fix: use signature scheme with unique signatures (e.g., BLS)
- issue #3: $sig_{sk_i}(r_v)$ may not be uniformly random (even if r_v is)
  - fix: use $sig_{sk_i}(r_v)$ as input to a cryptographic hash function

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Terminology: $h(sig_{sk_i}(r_v))$ an example of a *verifiable random function*.

- will write as $VRF_{sk_i}(\cdot)$

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Terminology: $h(sig_{sk_i}(r_v))$ an example of a *verifiable random function*.

- will write as $VRF_{sk_i}(\cdot)$

1. given sk_i, easy to compute $VRF_{sk_i}(\cdot)$

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Terminology: $h(sig_{sk_i}(r_v))$ an example of a *verifiable random function*.

- will write as $VRF_{sk_i}(\cdot)$

1. given sk_i, easy to compute $VRF_{sk_i}(\cdot)$
2. knowing only pk_i and not sk_i, hard to compute $VRF_{sk_i}(\cdot)$

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Terminology: $h(sig_{sk_i}(r_v))$ an example of a *verifiable random function*.

- will write as $VRF_{sk_i}(\cdot)$

1. given sk_i, easy to compute $VRF_{sk_i}(\cdot)$
2. knowing only pk_i and not sk_i, hard to compute $VRF_{sk_i}(\cdot)$
3. given pk_i and alleged VRF output on some input, easy to check

Verifiable Random Functions (VRFs)

Idea (revised): each view v, each validator i computes $h(sig_{sk_i}(r_v))$.

- i is leader ⇔ $h(sig_{sk_i}(r_v))$ is "sufficiently small" (TBA, depends on q_i)
- h(.) = CHF, sig(.) = signature scheme w/unique signatures

Terminology: $h(sig_{sk_i}(r_v))$ an example of a *verifiable random function*.

- will write as $VRF_{sk_i}(\cdot)$

1. given sk_i, easy to compute $VRF_{sk_i}(\cdot)$
2. knowing only pk_i and not sk_i, hard to compute $VRF_{sk_i}(\cdot)$
3. given pk_i and alleged VRF output on some input, easy to check
4. output indistinguishable from random

Setting the Thresholds

Idea (revised): each view v, each validator i computes $VRF_{sk_i}(r_v)$.

– i is leader ⇔ $VRF_{sk_i}(r_v)$ is "sufficiently small" (at most some threshold τ_i)

Issue #5:

Setting the Thresholds

Idea (revised): each view v, each validator i computes $VRF_{sk_i}(r_v)$.

– i is leader ⇔ $VRF_{sk_i}(r_v)$ is "sufficiently small" (at most some threshold τ_i)

Issue #5: how to set the thresholds (the τ_i's)?

– e.g., to guarantee one leader in expectation, but also sybil-proof

Setting the Thresholds

Idea (revised): each view v, each validator i computes $VRF_{sk_i}(r_v)$.

– i is leader ⇔ $VRF_{sk_i}(r_v)$ is "sufficiently small" (at most some threshold τ_i)

Issue #5: how to set the thresholds (the τ_i's)?

– e.g., to guarantee one leader in expectation, but also sybil-proof

Example: suppose q_i=1 for all i=1,2,…,n. [easiest case]

Setting the Thresholds

Idea (revised): each view v, each validator i computes $VRF_{sk_i}(r_v)$.

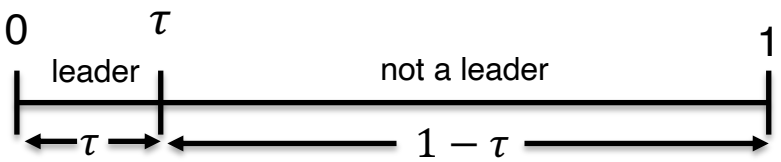
– i is leader ⇔ $VRF_{sk_i}(r_v)$ is "sufficiently small" (at most some threshold τ_i)

Issue #5: how to set the thresholds (the τ_i's)?

– e.g., to guarantee one leader in expectation, but also sybil-proof

Example: suppose q_i=1 for all i=1,2,…,n. [easiest case]

- if set (common) threshold of τ, Pr[i a leader]= τ for all i=1,2,..,n
  – since VRF output acts like a uniformly random number from [0,1]



Setting the Thresholds

Idea (revised): each view v, each validator i computes $VRF_{sk_i}(r_v)$.

– i is leader ⇔ $VRF_{sk_i}(r_v)$ is "sufficiently small" (at most some threshold τ_i)

Issue #5: how to set the thresholds (the τ_i's)?

– e.g., to guarantee one leader in expectation, but also sybil-proof

Example: suppose q_i=1 for all i=1,2,…,n. [easiest case]

- if set (common) threshold of τ, Pr[i a leader]= τ for all i=1,2,..,n
  - expected number of leaders = $\tau \cdot n$, set $\tau = 1/n$ for one leader in expectation

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ $VRF_{sk_i}(r_v) < \tau_i$.

- how to set the τ_i's, e.g., for one leader in expectation, but also sybil-proof?
  - all q_i's = 1 ➔ set $\tau = 1/n$ for one leader in expectation

Intuition: if $q_i > 1$, …

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ $VRF_{sk_i}(r_v) < \tau_i$.

- how to set the τ_i's, e.g., for one leader in expectation, but also sybil-proof?
  - all q_i's = 1 ➔ set $\tau = 1/n$ for one leader in expectation

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- in effect, treat i *as if* it was using q_i sybils with one coin each

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ $VRF_{sk_i}(r_v) < \tau_i$.

- how to set the τ_i's, e.g., for one leader in expectation, but also sybil-proof?
  - all q_i's = 1 ➔ set $\tau = 1/n$ for one leader in expectation

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- in effect, treat i *as if* it was using q_i sybils with one coin each

- first guess: if $q_i > 1$, use $\frac{1}{q_i} VRF_{sk_i}(r_v)$ instead of $VRF_{sk_i}(r_v)$
  - compare versus the threshold $\tau = 1/Q$, where $Q = \sum_{i=1}^{n} q_i$

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ $VRF_{sk_i}(r_v) < \tau_i$.

- how to set the τ_i's, e.g., for one leader in expectation, but also sybil-proof?
  - all q_i's = 1 ➔ set $\tau = 1/n$ for one leader in expectation

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- in effect, treat i *as if* it was using q_i sybils with one coin each

- first guess: if $q_i > 1$, use $\frac{1}{q_i} VRF_{sk_i}(r_v)$ instead of $VRF_{sk_i}(r_v)$
  - compare versus the threshold $\tau = 1/Q$, where $Q = \sum_{i=1}^{n} q_i$
- you check: not quite right

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- rewrite RHS of (*) for $q_i = 1$ as $\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- rewrite RHS of (*) for $q_i = 1$ as $\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$
- $q_i > 1$ ➔ i leader ⇔ $\frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- rewrite RHS of (*) for $q_i = 1$ as $\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$
- $q_i > 1$ ➔ i leader ⇔ $\underbrace{\frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)})}_{\text{"crediential" of i in view v}} < \ln(\frac{1}{1-\tau})$

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- rewrite RHS of (*) for $q_i = 1$ as $\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$
- $q_i > 1$ ➔ i leader ⇔ $\underbrace{\frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)})}_{\text{"crediential" of i in view v}} < \ln(\frac{1}{1-\tau})$
  - for one leader in expectation, use $\tau = 1/Q$, where $Q = \sum_{i=1}^{n} q_i$

Setting the Thresholds (con'd)

Idea: i is leader of view v ⇔ "adjusted $VRF_{sk_i}(r_v)$" $< \tau$. (*)

Intuition: if $q_i > 1$, adjust $VRF_{sk_i}(r_v)$ s.t. it has same distribution as the minimum of q_i i.i.d. uniform samples from [0,1]. (for sybil-proofness)

- rewrite RHS of (*) for $q_i = 1$ as $\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$
- $q_i > 1$ ➔ i leader ⇔ $\underbrace{\frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)})}_{\text{"crediential" of i in view v}} < \ln(\frac{1}{1-\tau})$
  - for one leader in expectation, use $\tau = 1/Q$, where $Q = \sum_{i=1}^{n} q_i$
  - you check: sybil-proof (# of pks doesn't change selection probability)

From VRFs to Consensus Protocols

Final version of idea: i is leader of view v ⇔

$$\underbrace{\frac{1}{q_i}\ln\left(\frac{1}{1-VRF_{sk_i}(r_v)}\right)}_{\text{“credential” of i in view v}} < \ln\left(\frac{1}{1-\tau}\right)$$

to target one leader, use $\tau = 1/(\text{total stake})$

To use in a blockchain protocol, need:

From VRFs to Consensus Protocols

Final version of idea: i is leader of view v ⇔

$$\underbrace{\frac{1}{q_i}\ln\left(\frac{1}{1-VRF_{sk_i}(r_v)}\right)}_{\text{“credential” of i in view v}} < \ln\left(\frac{1}{1-\tau}\right)$$

to target one leader, use $\tau = 1/(\text{total stake})$

To use in a blockchain protocol, need:

1. Concrete implementation of randomness beacon.
   - i.e., what are the r_v's, exactly?

From VRFs to Consensus Protocols

Final version of idea: i is leader of view v ⇔

$$\underbrace{\frac{1}{q_i}\ln\left(\frac{1}{1-VRF_{sk_i}(r_v)}\right)}_{\text{“credential” of i in view v}} < \ln\left(\frac{1}{1-\tau}\right)$$

to target one leader, use $\tau = 1/(\text{total stake})$

To use in a blockchain protocol, need:

1. Concrete implementation of randomness beacon.
   - i.e., what are the r_v's, exactly?
2. Modifications to consensus protocol to handle bad cases #1 + 2.
   - if no leader or multiple leaders selected in a view

Proof-of-Stake Tendermint Revisited

Last time: modified Tendermint so that:

1. uses epoch-based weighted round-robin leader selection
   – validators only allowed to join/leave at epoch boundaries
2. redefine quorum certificate = signatures by distinct public keys that collectively represent more than 2/3rds of the overall stake
3. add logic to update validator set at epoch boundaries

Result: consistent and (eventually) live in partial synchrony (assuming < 33% Byzantine stake), in the quasi-permissionless setting (i.e., honest validators always online).

Proof-of-Stake Tendermint Revisited

Last time: modified Tendermint so that:

1. uses epoch-based weighted round-robin leader selection
2. redefine quorum certificate = signatures by distinct public keys that collectively represent more than 2/3rds of the overall stake
3. add logic to update validator set at epoch boundaries

Here: keep (2) and (3), but replace (1) with VRF-based sampling:

i is leader of view v $\Leftrightarrow \underbrace{\frac{1}{q_i}\ln\left(\frac{1}{1-VRF_{sk_i}(r_v)}\right)}_{\text{“crediential” of i in view v}} < \ln\left(\frac{1}{1-\tau}\right)$.

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

- e.g., to target one leader, use $\tau = 1/(\text{total stake})$

Question:

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

- e.g., to target one leader, use $\tau = 1/(\text{total stake})$

Question: what if a view has no leaders?

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

- e.g., to target one leader, use $\tau = 1/(\text{total stake})$

Question: what if a view has no leaders?

- answer: nothing happens, as if leader crashed

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what if a view has no leaders?

- answer: nothing happens, as if leader crashed

Question: what if a view has multiple leaders?

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what if a view has no leaders?

- answer: nothing happens, as if leader crashed

Question: what if a view has multiple leaders?

- answer: honest validators vote, among all block proposals seen, for the one accompanied by the smallest leader credential

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what if a view has no leaders?

- answer: nothing happens, as if leader crashed

Question: what if a view has multiple leaders?

- answer: honest validators vote, among all block proposals seen, for the one accompanied by the smallest leader credential
  - you check: Tendermint retains its consistency and liveness guarantees

Handling the Two Bad Cases

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what if a view has no leaders?

- answer: nothing happens, as if leader crashed

Question: what if a view has multiple leaders?

- answer: honest validators vote, among all block proposals seen, for the one accompanied by the smallest leader credential
  - you check: Tendermint retains its consistency and liveness guarantees
  - you check: Pr[i has smallest credential] = $q_i/(\sum_{j=1}^{n} q_j)$

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i} \ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- idea: output of a cryptographic hash function h

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i} \ln\left(\frac{1}{1-VRF_{sk_i}(r_v)}\right) < \ln\left(\frac{1}{1-\tau}\right)$.

Question: what is r_v?

- idea: output of a cryptographic hash function h
  - question: what's the input?

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- idea: output of a cryptographic hash function h
  - question: what's the input?
- idea: some function f(.) of blockchain state σ_v at start of view v

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- idea: output of a cryptographic hash function h
  - question: what's the input?
- idea: some function f(.) of blockchain state σ_v at start of view v

Minor issue: blockchain state may not change between views.

- e.g., if leader of last view crashed

The (Pseudo)Randomness Beacon

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- idea: output of a cryptographic hash function h
  - question: what's the input?
- idea: some function f(.) of blockchain state σ_v at start of view v

Minor issue: blockchain state may not change between views.

- solution: use $r_v = h(f(\sigma_v)||v)$ rather than $r_v = h(f(\sigma_v))$

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = cryptographic hash function, f = some function of blockchain state σ_v

Major issue: validators can manipulate σ_v (and therefore r_v).

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = cryptographic hash function, f = some function of blockchain state σ_v

Major issue: validators can manipulate σ_v (and therefore r_v).

- solution: choose function f to minimize manipulability

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = cryptographic hash function, f = some function of blockchain state σ_v

Major issue: validators can manipulate σ_v (and therefore r_v).

- solution: choose function f to minimize manipulability
  - bad choice: $f(\sigma_v)$ = most recently finalized block

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = cryptographic hash function, f = some function of blockchain state σ_v

Major issue: validators can manipulate σ_v (and therefore r_v).

- solution: choose function f to minimize manipulability
  - bad choice: $f(\sigma_v)$ = most recently finalized block
    - incentivizes leader of view v-1 to grind over blocks to minimize its view-v credential (turns PoS into PoW!)

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = cryptographic hash function, f = some function of blockchain state σ_v

Major issue: validators can manipulate σ_v (and therefore r_v).

- solution: choose function f to minimize manipulability
- example: $f(\sigma_v)$ = leader credential in most recently finalized block
  - assuming warm-up period, no grinding over sk_i's credential possible

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = CHF, $f(\sigma_v)$ = leader credential in most recently finalized block

Issue: validators can still (to some extent) manipulate the r_v's.

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = CHF, $f(\sigma_v)$ = leader credential in most recently finalized block

Issue: validators can still (to some extent) manipulate the r_v's.

- suppose validator i controls pk_1 + pk_2, leaders of view v = {pk_1,pk_2}

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = CHF, $f(\sigma_v)$ = leader credential in most recently finalized block

Issue: validators can still (to some extent) manipulate the r_v's.

- suppose validator i controls pk_1 + pk_2, leaders of view v = {pk_1,pk_2}
- i incentivized to propose block with whichever of pk_1,pk_2 results in smaller view-(v+1) credentials for it (as opposed to smallest view-v credential)

The (Pseudo)Randomness Beacon (con'd)

Now: i is leader of view v $\Leftrightarrow \frac{1}{q_i}\ln(\frac{1}{1-VRF_{sk_i}(r_v)}) < \ln(\frac{1}{1-\tau})$.

Question: what is r_v?

- proposal: $r_v = h(f(\sigma_v)||v)$
  - h = CHF, $f(\sigma_v)$ = leader credential in most recently finalized block

Issue: validators can still (to some extent) manipulate the r_v's.

- suppose validator i controls pk_1 + pk_2, leaders of view v = {pk_1,pk_2}
- i incentivized to propose block with whichever of pk_1,pk_2 results in smaller view-(v+1) credentials for it (as opposed to smallest view-v credential)
- in practice, generally willing to live with this

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled “committees” of validators

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled "committees" of validators
  - tricky but solvable: sample committee in sybil-proof way

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled "committees" of validators
  - tricky but solvable: sample committee in sybil-proof way
  - bigger issue: unlucky sample ➔ committee could be > 33% Byzantine

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled "committees" of validators
  - tricky but solvable: sample committee in sybil-proof way
  - bigger issue: unlucky sample ➔ committee could be > 33% Byzantine
    - fix #1: make stronger assumption about fraction of honest stake

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled "committees" of validators
  - tricky but solvable: sample committee in sybil-proof way
  - bigger issue: unlucky sample ➔ committee could be > 33% Byzantine
    - fix #1: make stronger assumption about fraction of honest stake
    - fix #2: make committee sizes sufficiently big

Randomly Sampled Committees

Issue: too many validators ➔ bad performance in Tendermint.

- solution #1: cap number of validators, explicitly or implicitly
- solution #2: combine Tendermint with some aspects of longest-chain consensus (which scales well with # of validators)
- solution #3: use randomly sampled "committees" of validators
  - tricky but solvable: sample committee in sybil-proof way
  - bigger issue: unlucky sample ➔ committee could be > 33% Byzantine
    - fix #1: make stronger assumption about fraction of honest stake
    - fix #2: make committee sizes sufficiently big
    - fix #3: emergency procedure in case of forks by bad committees