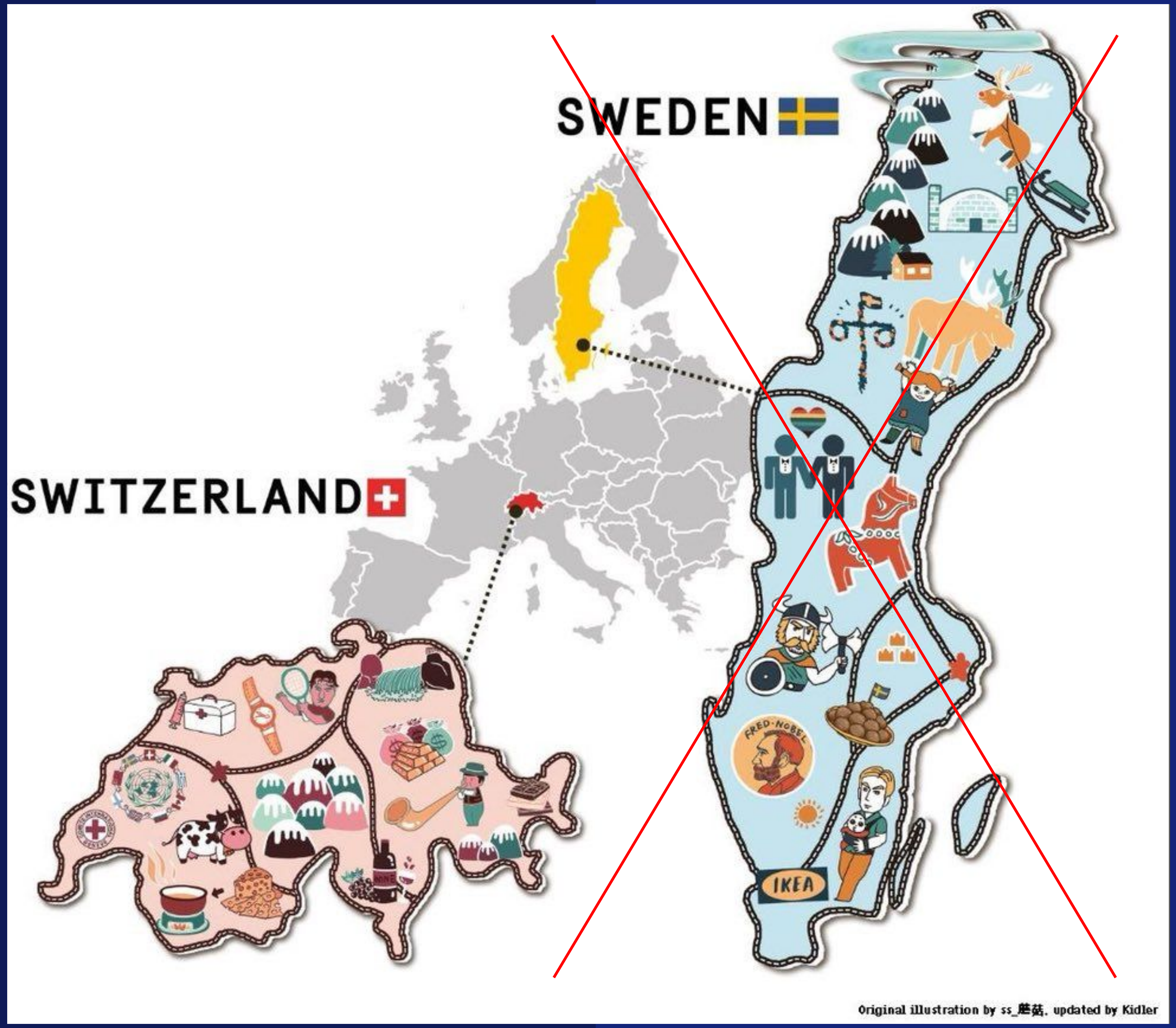
Public, verifiable, and unbiasable randomness wassat?

or *Randomness 101*

Yolan Romailler (@anomalroil)



Who am I?

→ SWE / applied cryptographer @ Protocol Labs

→ CTF player (mostly crypto, forensic & misc)

→ Board games amateur

→ Speaker at Black Hat, DEF CON, GopherConEU, and it's my 2nd time at North Sec!

→ Maths background, but don't worry!

Protocol Labs

Email
yolan@protocol.ai

Twitter
https://twitter.com/anomalroil

Agenda

- What is randomness and its flavours?
- Why do we need it?
- Why are there problems with it?
- In practice
- Future works

What is randomness?

What is randomness?

According to the Cambridge dictionary, randomness is:

- “the quality of being random”

What is randomness?

According to the Cambridge dictionary, randomness is:

- “the quality of being random”

Granted, they refine it a bit:

- “the quality of being random (= happening, done, or chosen by chance rather than according to a plan)”

What is randomness?

According to the Cambridge dictionary, randomness is:

- “the quality of being random”

Granted, they refine it a bit:

- “the quality of being random (= happening, done, or chosen by chance rather than according to a plan)”

But I still prefer the Oxford Languages definition:

- “the quality or state of lacking a pattern or principle of organization; unpredictability”

Is that random?

```
0b1111111111111111111111111111111111111
0b0100111101011110110011010110000000101
0b1000000000000000000011111111111111111
0b1001000110100010101100111100010101011
0b1010101010101010101010101010101010101
```

Is that random?

~~0b1111111111111111111111111111111111111~~

0b0100111101011110110011010110000000101

0b1000000000000000001111111111111111111

0b1001000110100010101100111100010101011

0b1010101010101010101010101010101010101

Is that random?

~~0b1111111111111111111111111111111111111~~

0b0100111101011110110011010110000000101 → 0x09ebd9ac05

0b1000000000000000001111111111111111111

0b1001000110100010101100111100010101011

0b1010101010101010101010101010101010101

Is that random?

```
~~0b1111111111111111111111111111111111111~~
0b0100111101011110110011010110000000101 → 0x09ebd9ac05
~~0b1000000000000000000111111111111111111~~
0b1001000110100010101100111100010101011
0b1010101010101010101010101010101010101
```

Is that random?

~~0b1111111111111111111111111111111111111~~

0b0100111101011110110011010110000000101 → 0x09ebd9ac05

~~0b1000000000000000000111111111111111111~~

~~0b1001000110100010101100111100010101011~~ → 0x12345678ab

0b1010101010101010101010101010101010101

Is that random?

~~0b1111111111111111111111111111111111111~~

0b0100111101011110110011010110000000101 → 0x09ebd9ac05

~~0b1000000000000000000111111111111111111~~

~~0b1001000110100010101100111100010101011~~ → 0x12345678ab

~~0b1010101010101010101010101010101010101~~

Is that random?

~~0b1111111111111111111111111111111111111~~

0b0100111101011110110011010110000000101 → 0x09ebd9ac05

~~0b1000000000000000000111111111111111111~~

~~0b1001000110100010101100111100010101011~~ → 0x12345678ab

~~0b1010101010101010101010101010101010101~~

And yet all of them have the same probability to occur in a random draw!

The notion of randomness

- So, we have some kind of intuition of “what is random”, but it still can be fooled.
- A more formal treatment of randomness can be done using “the Kolmogorov complexity” which can also help us understand our intuition.
- The Kolmogorov complexity of something is the length of a shortest program (in a given language) that produces that thing as output:

```
print(‘0b’+37*’1’)                                  → 19 chars
print(‘0b1111111111111111111111111111111111111’) → 48 chars
```

What is *public* randomness?

- Public randomness is simply a **random value that is meant to be *public***

- Its goal is often to increase the trust we have in a random “draw” (think of lotteries, tombola, jury election, etc.)

- We want public randomness typically for:
  - avoiding risks of manipulation
  - “be off the hook” in case of issues

vs “*secret*” randomness

We often rely on “secret” randomness:

- to generate keys, both for public key cryptography and symmetric cryptography
- for ephemeral keys / IV / nonces (most protocols need some kind to ensure forward secrecy)

WARNING:

DO NOT USE PUBLIC RANDOMNESS TO GENERATE CRYPTOGRAPHIC MATERIAL

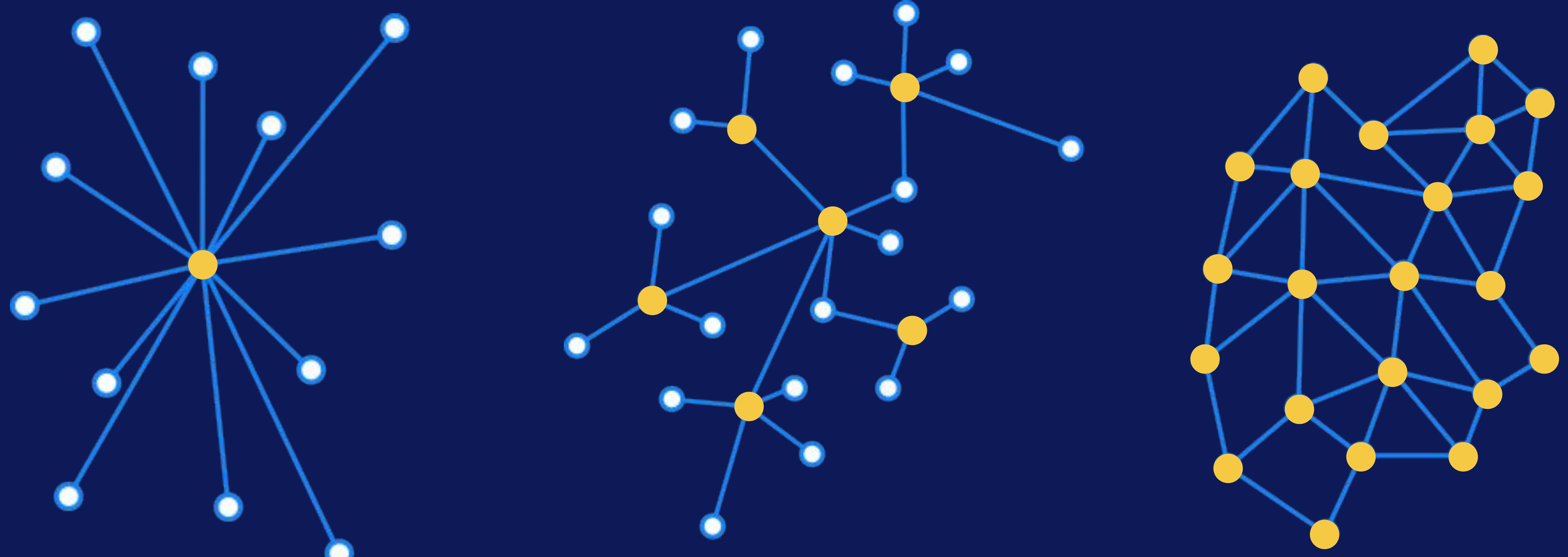
What about *verifiable* randomness?

- Public randomness is cool, but we usually want it to allow for “public auditability” of the resulting randomness

- Verifiable randomness means we can easily verify that it was properly issued and not manipulated

- To achieve such auditability we typically use:
  - signatures
  - secure hardware & “remote attestations”
  - in general: complex cryptography

What is *distributed* randomness?



The notion of distributed randomness hides two aspects:

- decentralisation of trust, no single point of failure
- achieving consensus on a random value is hard

Failing at producing proper randomness can be very dangerous for any distributed system, especially nowadays for blockchains

Why do we need randomness?

Why do we need randomness ?

- Lotteries, jury selection, election event, audits…
- Protocols & Cryptography:
  - **Protocols**: leader election in Proof of Stake blockchains, Tor (path selection), sharding
  - **Gossiping**: randomly choosing peers in the network to disseminate information
  - **Parameters**: Nonces & IV for symmetric encryptions, composite or prime numbers for selecting a field for RSA, or even ECC
  - **Schemes**: Diffie Hellman exchange, Schnorr signatures, more generally for zero knowledge proofs…
- Statistics: sampling, reducing bias in controlled trials in medicine
- Software: fuzzing, chaos monkey, etc
- Even for *cleromancy* and *divination* … !

Why are there problems with randomness?

“Randomness is hard”

This is something you’ll often hear whenever you talk to an applied cryptographer who did some code assessment in their life.

In general it’s very important to have “proper” randomness, that is:

- **Unpredictable**: impossible to predict the next numbers
- **Bias-resistant**: the final output cannot be biased in any way

Why unpredictable?

- If you can predict the random value, you can “cheat” (gambling, games, etc.)
- If you can predict who’s going to be selected, fairness isn’t guaranteed anymore (think of leader election, sharding, jury election, …)
- If you can predict “a secret key”, then the security of the system is compromised

Why unbiased?

- The (EC)DSA signature scheme requires **uniformly distributed nonces** for its “k” value, and attacks exploiting biased ephemeral keys are known since 1999, and used in practice (the PS3 hack, Biased Nonce Sense, etc.)
- the ElGamal signature schemes in general (caused a vuln in GPG)
- the Schnorr signature schemes in general (just like the others)

Notice that a nonce can be biased on **less than a bit** and still lead to key recovery attacks!

It’s easy to have biased randomness

Get a value x between 0 and 255:

```
    import os
    x = ord(os.urandom(1))
```

Get a value x between 0 and 106

```
    import os
    x = ord(os.urandom(1)) % 107
```

It’s easy to have biased randomness

Get a value x between 0 and 255:

```
import os
x = ord(os.urandom(1))
```

~~Get a value x betwe[illegible]n 0 and [illegible]06~~

```
import os
x = ord(os[illegible]m(1)) % 107
```

This is a Modulo Bias

Out of the 256 possible values of a random bytes, from 0 to 255, if we reduce modulo 107, then the lowest 42 first values are more likely to occur because 255 % 107 = 42

How to avoid bias?

Use a “crypto library” to get your random values:

- Python’s secrets `secrets.randbelow(107)`
- Go’s `crypto/rand`: `rand.Int(rand.Reader, big.NewInt(107))`
- Rust’s `rand`: `thread_rng().gen_range(0..107);`

Read more about modulo bias, and “rejection sampling”:

[The definitive guide to “modulo bias and how to avoid it”]()

In practice

History: Beacons

M. Rabin first proposed the usage of “random beacons” in 1993 to secure transactions [1]:

“These transactions are provably impossible without a trusted intermediary. However, they can be implemented with just a small probability of a participant cheating his partner, by use of a beacon emitting random integers”

[1] https://doi.org/10.1016/0022-0000(83)90042-9

History: Prior art

Public randomness services exist since a long time:

“RANDOM.ORG offers true random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs. People use RANDOM.ORG for holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music. The service has existed since 1998”

History: The NIST Beacons

- The idea of running a public, verifiable “trusted” randomness beacon was first proposed by NIST in 2011
- Their NIST Beacon v1 was launched on 2013-09-05
- Their NIST Beacon v2 was launched in 2019: https://doi.org/10.6028/NIST.IR.8213-draft

Previous attempts to generate public randomness

Some examples:

- NIST Randomness beacon[1] based on quantum entanglement:
  - Unpredictability, autonomy, consistency
  - We still need to trust NIST and its “secure hardware”… (remember the DUAL_EC_DRBG fiasco)

- Bitcoin[2]: Using blockchain as a source of random value
  - Promising, but slow, relies on PoW which is inefficient and leads to centralization

- Randhound[3]: the **jackpot**!
  - Scalable, bias-resistant, unpredictable, publicly verifiable, decentralized
  - Relies on solid cryptographic assumptions, uses ECC
  - But offers probabilistics *guarantees*, has complex setup, large transcript to verify, multiple RTT, 6s generation…

Can we do *simpler* & *faster* ?

1. https://www.nist.gov/programs-projects/nist-randomness-beacon
2. https://eprint.iacr.org/2015/1015
3. https://eprint.iacr.org/2016/1067.pdf

The Internet needed a randomness service, just like it has:

- **DNS**: Highly available source of naming information
- **NTP**: Highly available source of timing information
- **PKIs**: Trusted network delivering certificates
- **Certificate transparency**: Certificate authenticity information

➡️ **Drand**: Highly available, decentralized, and publicly verifiable source of randomness

Drand is meant to be a foundational Internet protocol for randomness

Drand properties

- Drand is a software ran by a set of independent nodes that collectively produce randomness
- Drand is **open source**[1], coded in Go
  - Originally from DEDIS@EPFL, moved to its own organization
  - Now supported by Protocol Labs
- **Decentralized randomness service** using
  - (t-n) Distributed Key Generation: t = n/2
  - Verifiable secret sharing, threshold cryptography
  - Key is defined on G2 of the **BLS12-381 pairing curve**, to achieve 128 bits of security
- Binds together **independent entropy sources** into a publicly verifiable one
- Tested, **audited**, and deployed (more on that later)

1. https://github.com/drand/drand

Drand properties

Decentralized: a threshold of nodes operated by different parties is needed to generate randomness; there is **no central point of failure.**

Unpredictable: no party learns anything about the output of the round **until a sufficient number of drand nodes** reveals their contributions thanks to **threshold cryptography**.

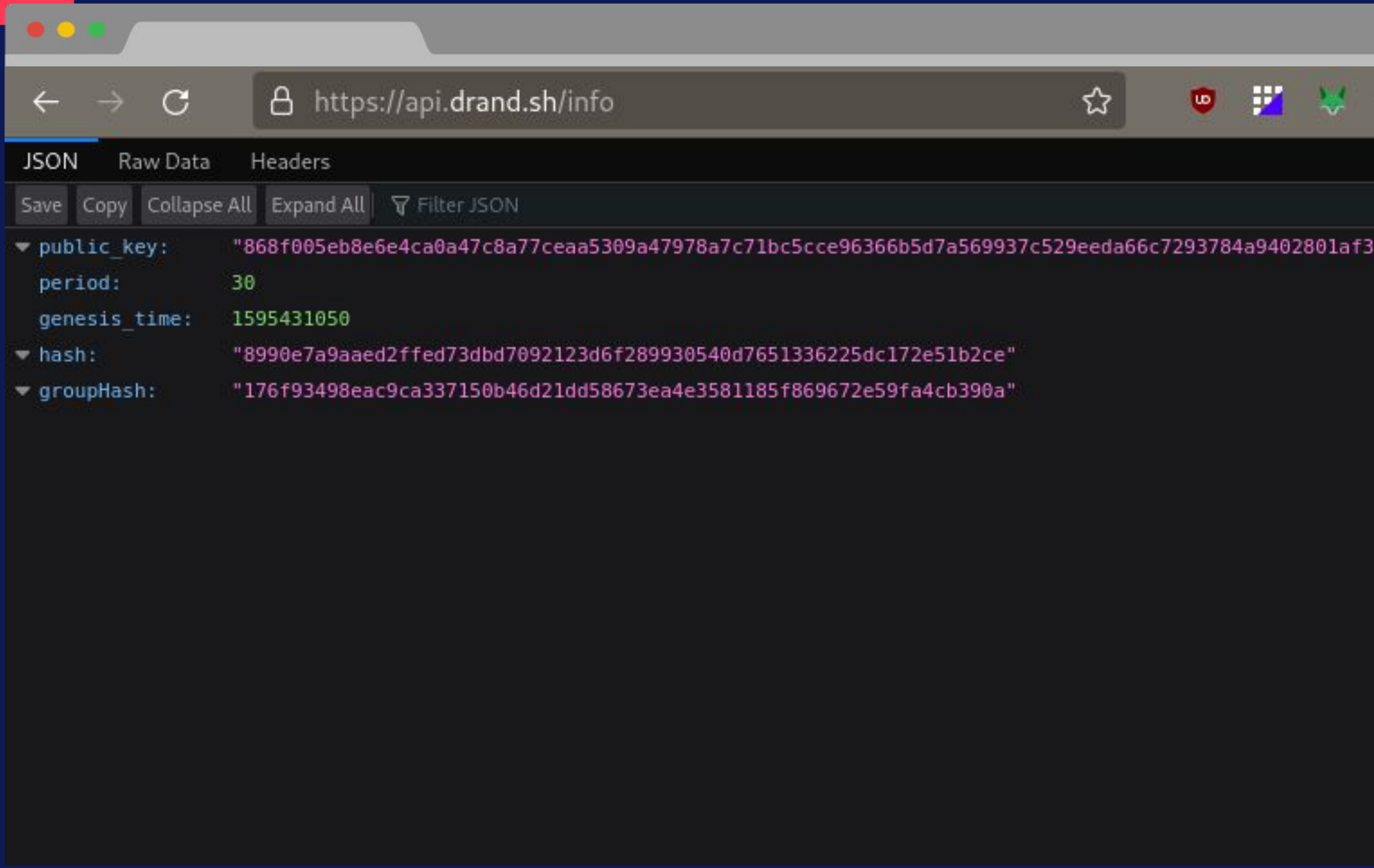
Bias Resistant: the output represents an **unbiased, uniformly random value.**

Verifiable: the random output is **third-party verifiable** by verifying the **aggregate BLS signatures** against the collective public key computed during setup.

More about the security model can be found online: https://drand.love/docs/security-model/

Public API: web endpoints

Firefox even supports a nice JSON viewer



```
▾ public_key:    "868f005eb8e6e4ca0a47c8a77ceaa5309a47978a7c71bc5cce96366b5d7a569937c529eeda66c7293784a9402801af3
  period:        30
  genesis_time:  1595431050
▾ hash:          "8990e7a9aaed2ffed73dbd7092123d6f289930540d7651336225dc172e51b2ce"
▾ groupHash:     "176f93498eac9ca337150b46d21dd58673ea4e3581185f869672e59fa4cb390a"
```

```
curl https://api.drand.sh/public/latest
```

The League of Entropy

The League is a global *drand network* composed of multiple *independent, diversified* organizations

- Created in June 2019[1] with initially 10 members.
- It is now composed of 16 members, 23 nodes and a threshold of 12.

And since it's open-source, anybody can run such a network!

1. https://www.cloudflare.com/leagueofentropy/

LEAGUE OF ENTROPY
CLOUDFLARE
c·Labs
The Initiative for CryptoCurrencies and Contracts
TIERION
Universidad de Chile
KUDELSKI SECURITY
ChainSafe
UCL
Protocol Labs
EPFL
ptisp
C4DT

drand network: the League of Entropy

The “entropy”

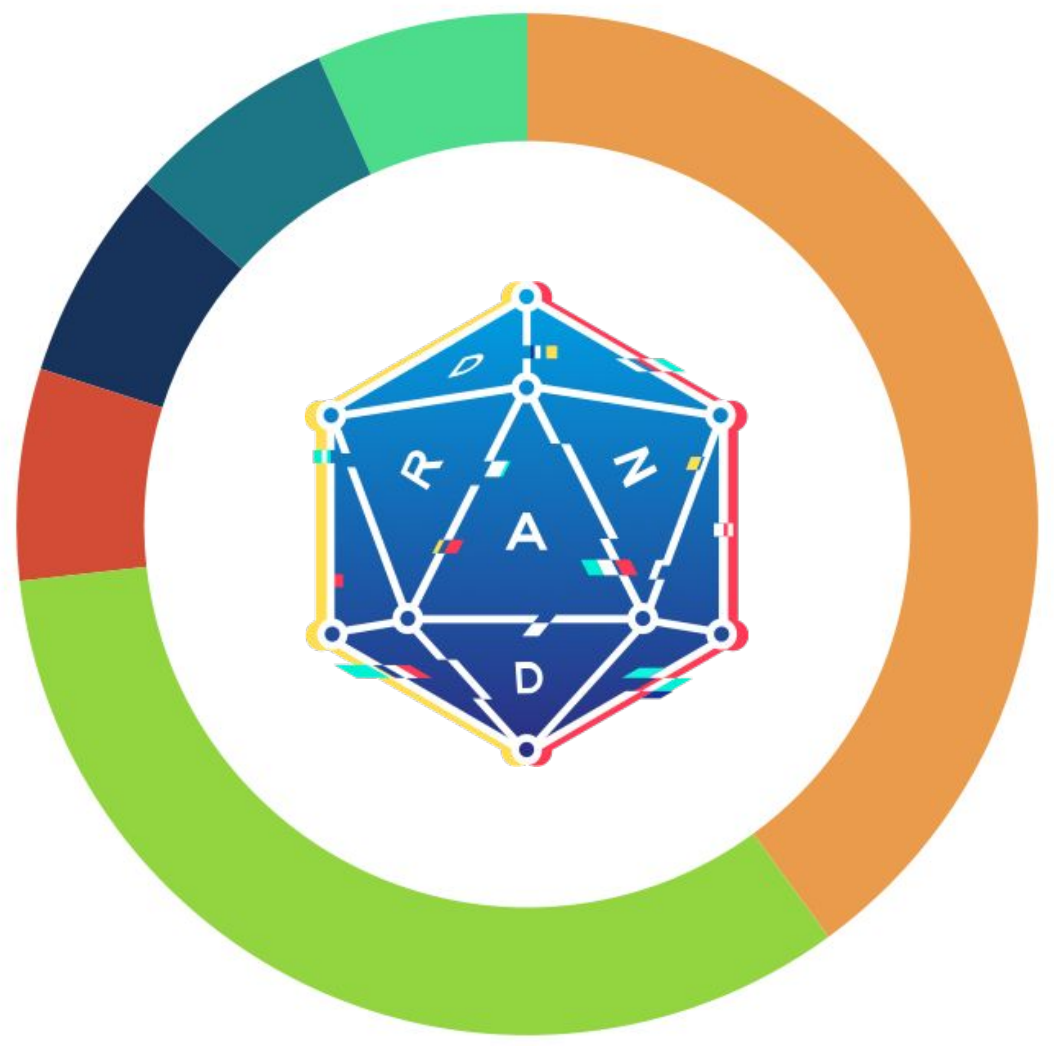
The only moment where fresh entropy is required is during the Distributed Key Generation.

Some partners are getting their entropy sources from so-called “TRNG”, based on physical properties known to be unpredictable.

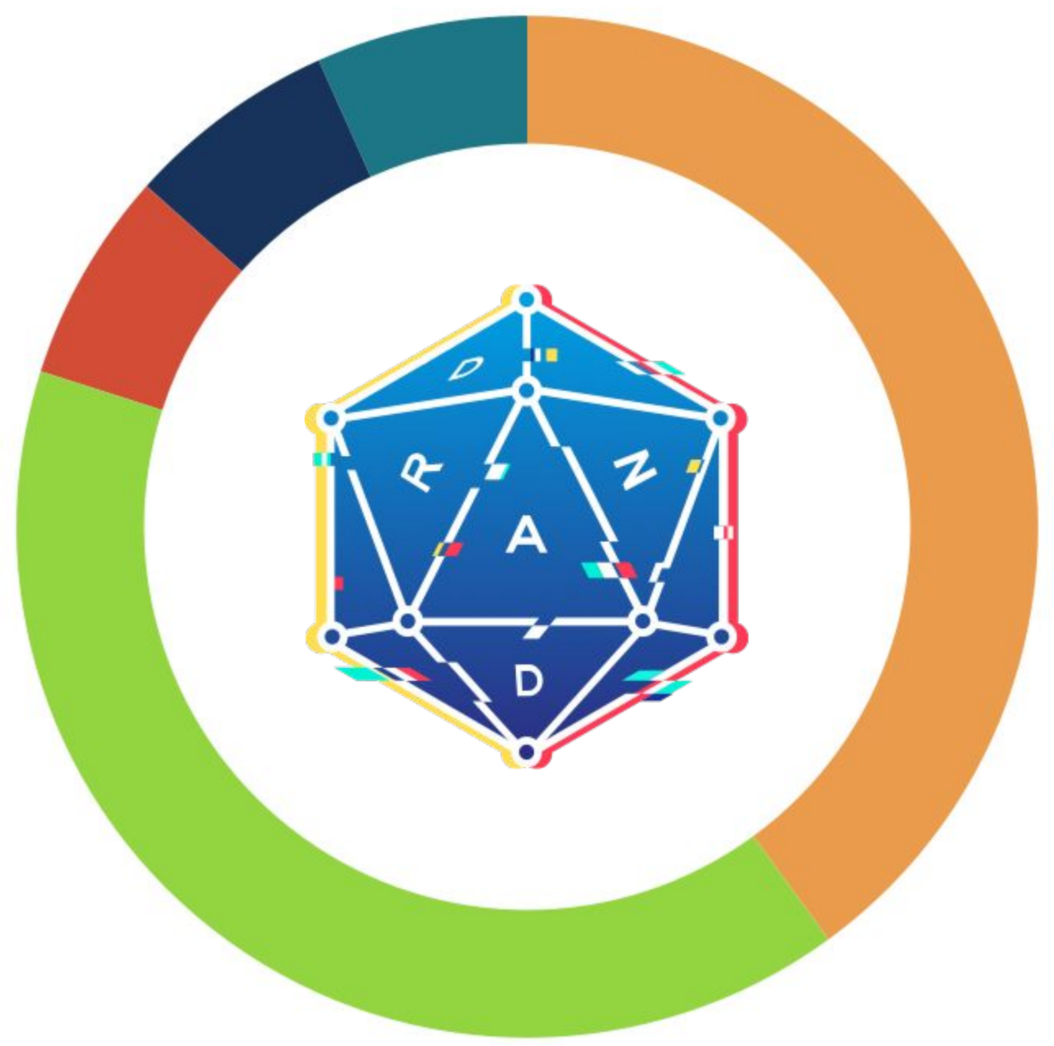
Lava lamps in the Cloudflare lobby. Courtesy of @mahtin drand network: the League of Entropy

Facts & Figures

Jurisdiction
Partner Diversity



🟧 USA
🟩 Switzerland
🟥 Chile
⬛ Portugal
🟦 Great Britain
🟩 Israel



🟧 on-prem
🟩 AWS
🟥 Cloudflare
⬛ Azure
🟦 Exoscale

16 organizations

1.9M+ rounds

23 nodes

1+ year

4 endpoints

0 disruptions since 2020-08-10

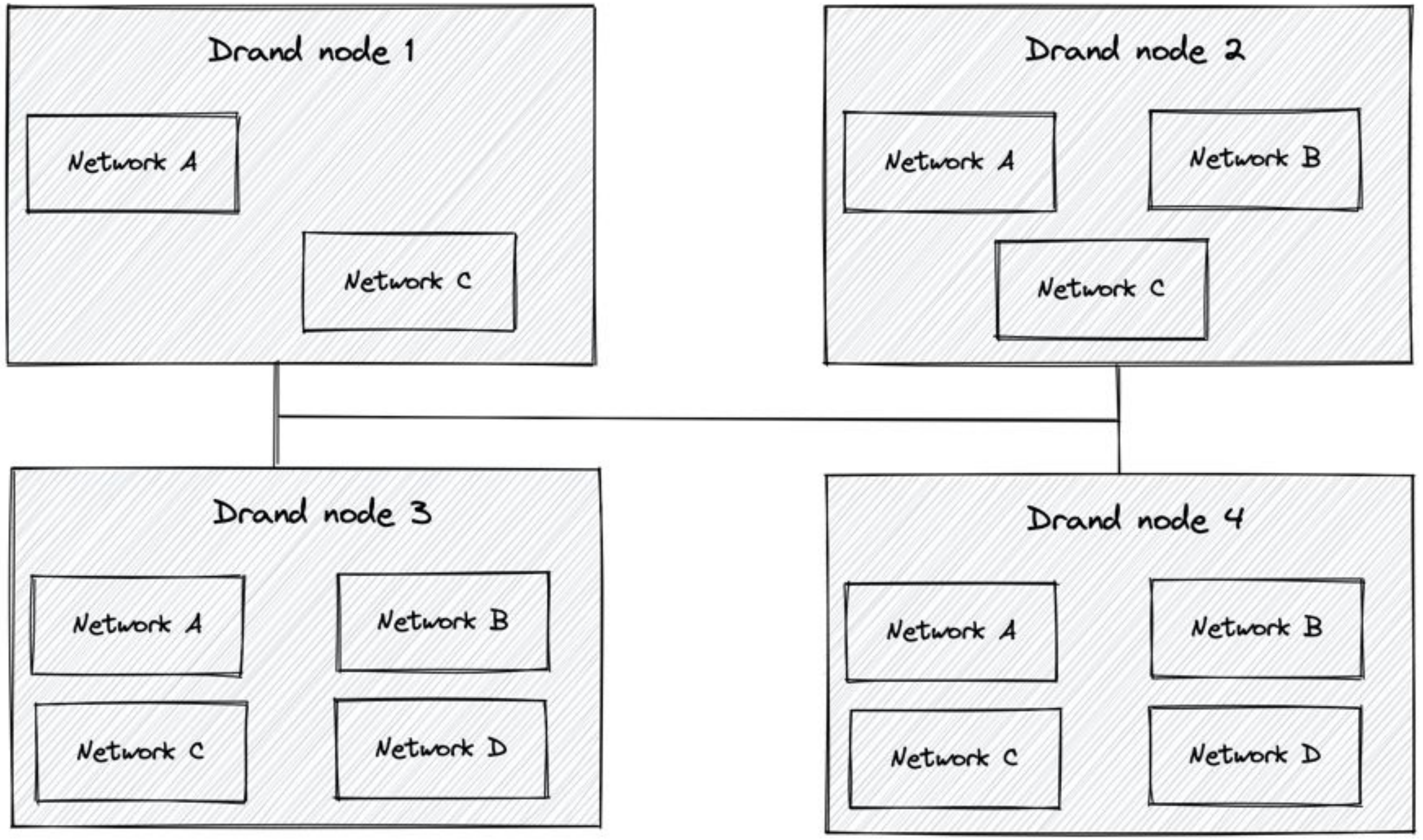
0 single point of failure

Multi Protocol Support

We can now have different protocols for different use cases in parallel!

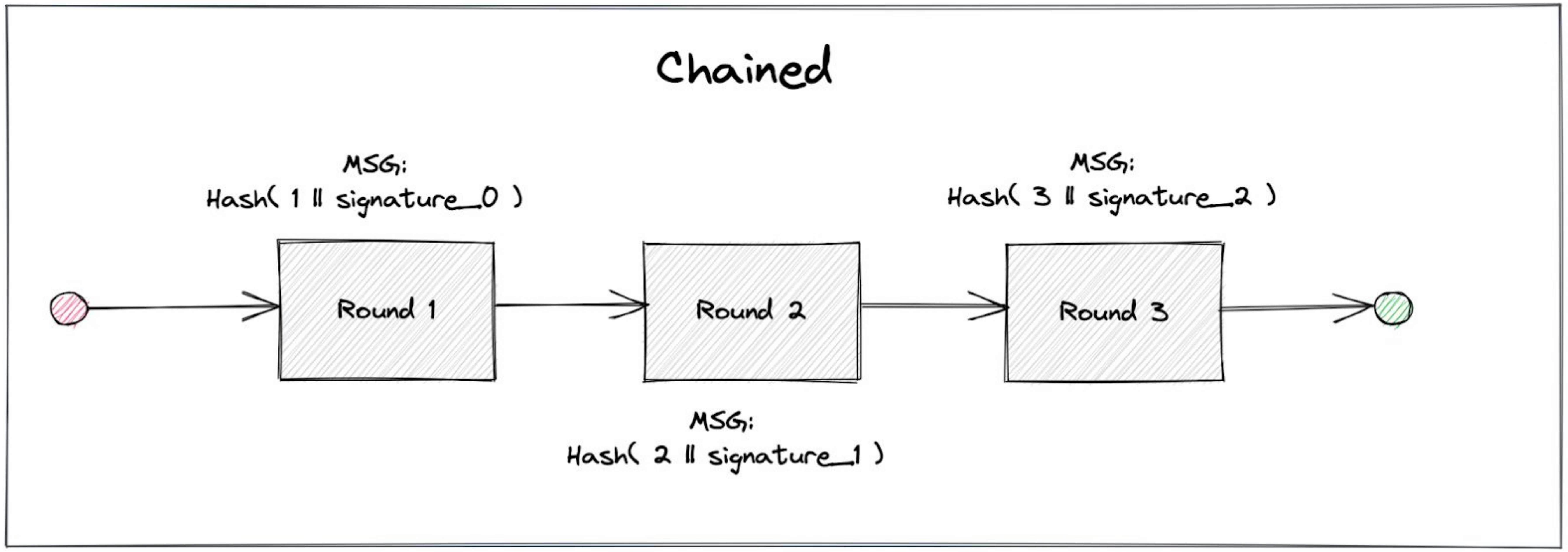
Current target: have a higher frequency network

This was just launched on our testnet!



Chained Randomness

Current randomness is **chained**:

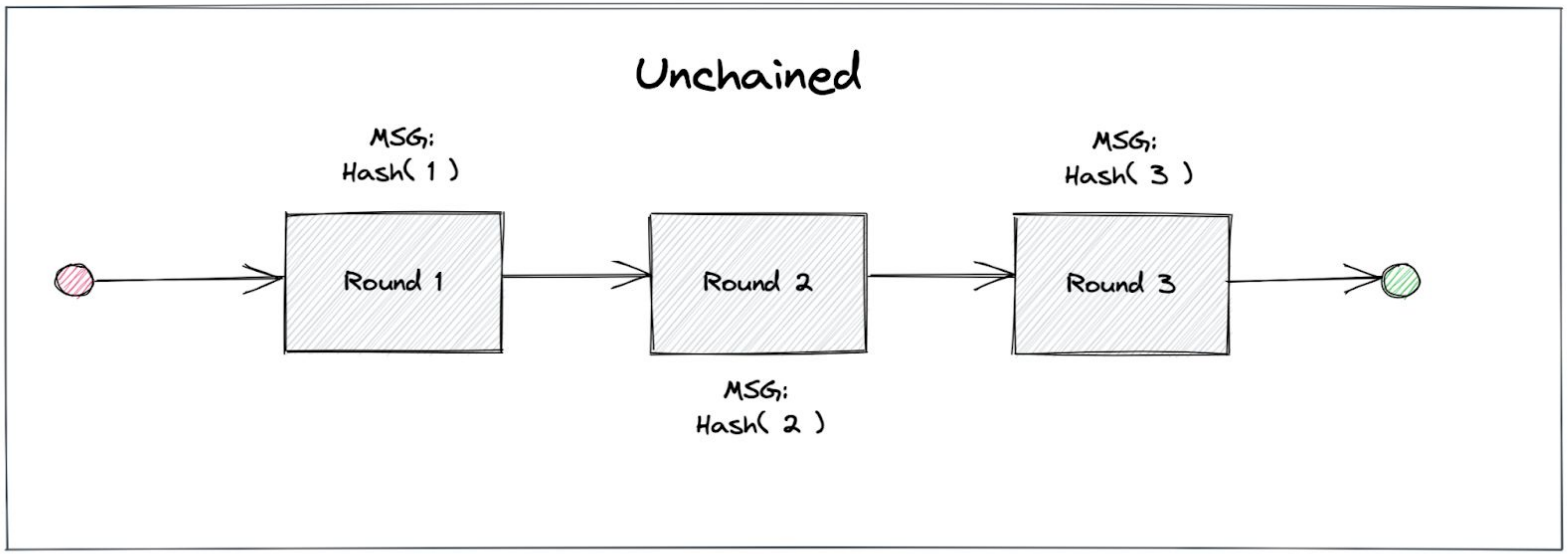


Consequences:

- No one knows the message before **one round** in advance
  - Hash(3 || sig_2) can only be known at round 2
- Need the full chain verification in applications

Unchained Randomness

New **unchained randomness**:



Consequences:

- Anybody can predict the message ahead of time
- Verification is simpler and less stateful

Future works

Timelock encryption!

- Being able to encrypt *toward the future*
- Idea initially submitted by **Tim May** in 1993 on the Cypherpunks mailing list that introduced the idea of relying on a **pool of trusted third parties** to release the sealed decryption key at the proper time.
- Using **paring-based** and **identity-based crypto** we can achieve this without any single trusted third party! → same trust assumptions as for the League
- Can significantly reduce frontrunning, and help mitigate MEV issues!
- Relying on drand's new unchained randomness beacons, we'll release open-source libraries and clients to do timelock encryption later this year!

Find more users!

- drand is used by Filecoin for Leader election!
- drand is currently integrated on-chain on some blockchain systems, to help produce proper randomness for smart contracts. Notice that in such cases extra care must be taken to properly handle front-running.
- It was used to increase the entropy of a RNG certified by Gaming Labs International for several lottery games

https://drand.love/

Grow the League!

- Join the League of Entropy!

join the effort to provide a randomness service as a foundational component of the Internet protocol stack

- We are looking for partners that can run multiple drand core or relay nodes.
- Infrastructure and operational requirements are minimal: Estimated commitment: 2-3 hours/month

https://drand.love/partner-with-us/

Thank you !

For more information and/or if you want to reach out, go to:

https://drand.love

https://leagueofentropy.com

https://github.com/drand/drand

Email
yolan@protocol.ai

Twitter
https://twitter.com/anomalroil