Distributed Randomness

Prof. Bryan Ford
Decentralized and Distributed Systems (DEDIS)
School of Information and Communications (IC)
dedis@epfl.ch – dedis.epfl.ch

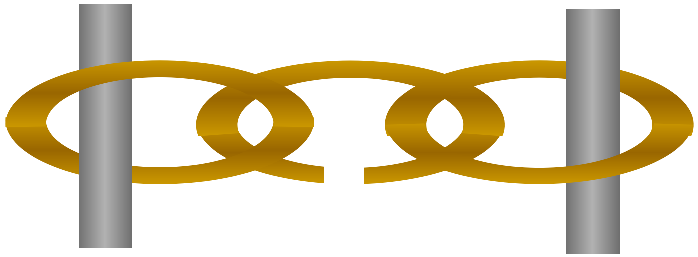
Nicolas Gailly
Protocol Labs

IC3 Blockchain Camp – July 30, 2020

A Fundamental Challenge

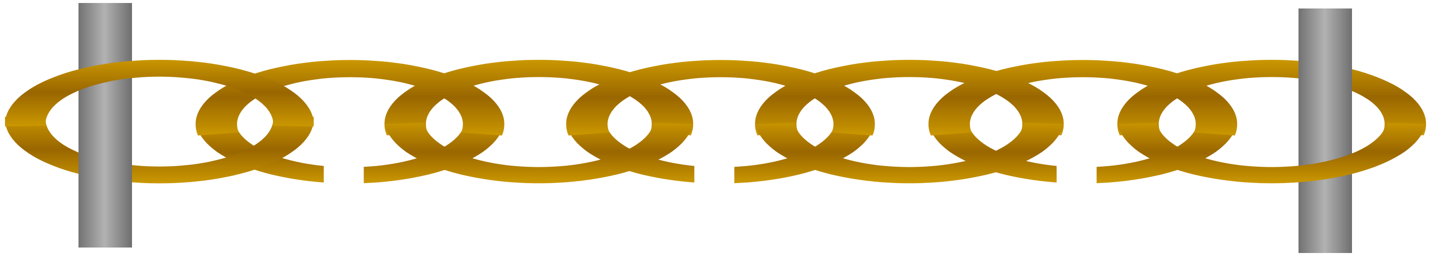
In today's IT systems, security is an afterthought

- Designs embody "weakest-link" security



Scaling to bigger systems → weaker security

- Greater chance of any "weak link" breaking

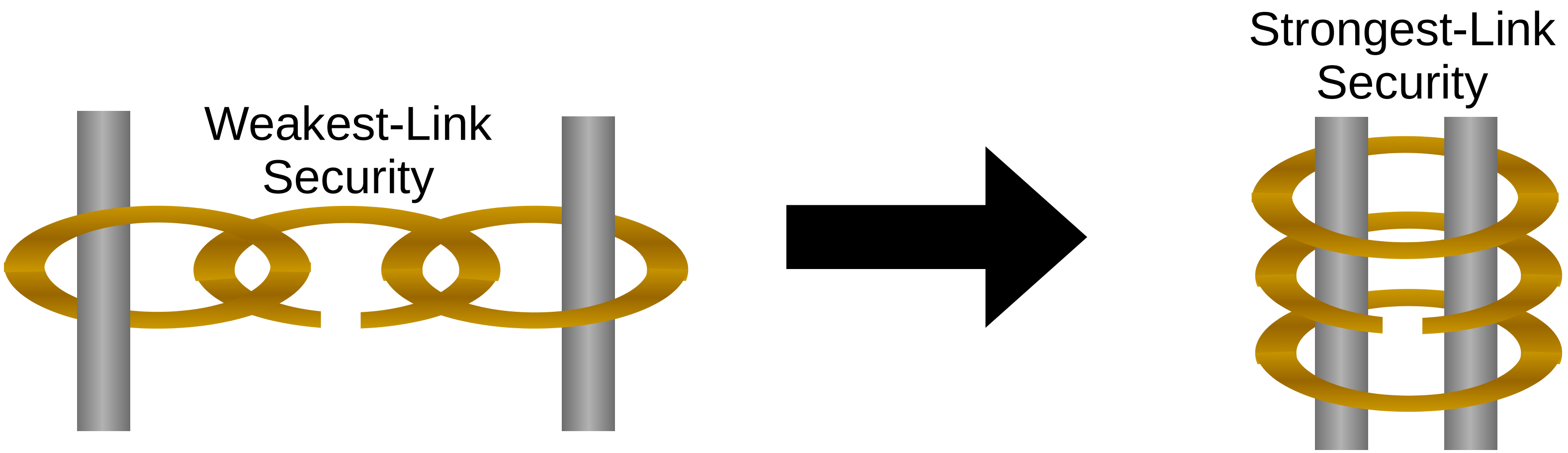


The DEDIS lab at EPFL: Mission

Design, build, and deploy secure privacy-preserving
Decentralized and Distributed Systems (DEDIS)

- **Distributed:** spread widely across the Internet & world
- **Decentralized:** independent participants, no central authority, *no single points of failure or compromise*

Overarching theme: building decentralized systems
that **distribute trust** widely with **strongest-link security**



Turning Around the Security Game

Design IT systems so that making them bigger makes their security *increase* instead of *decrease*

Weakest-link security

Strongest-link security

Scalable Strongest-link security

DEDIS Laboratory Members

Bryan Ford
Associate Professor

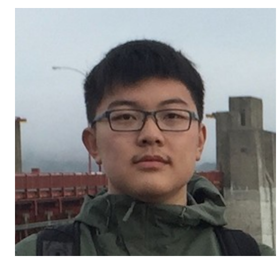
Jeff R. Allen
Laboratory Manager

David Lazar
Postdoctoral Scholar

Kirill Nikitin
Ph.D. Student

Cristina Basescu
Ph.D. Student

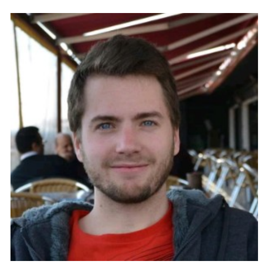
Enis Ceyhun Alp
Ph.D. Student



Haoqian Zhang
Ph.D. Student

Pasindu Tennage
Ph.D. Student

Gaylor Bosson
Software Engineer



Noémien Kocher
Software Engineer

Gaurav Narula
Software Engineer

Talk Outline

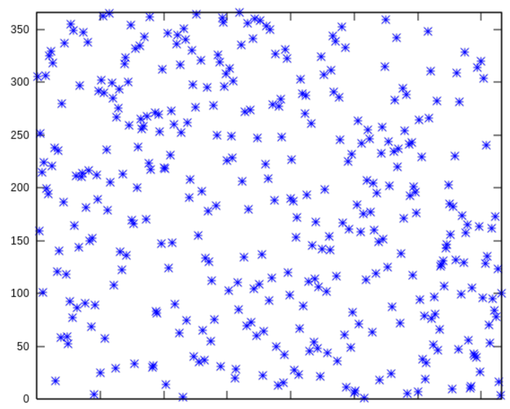
- Public Randomness: Introduction
  - Challenges: Quality, Trustworthiness, Bias
  - General Approaches Known
- Background: Shamir Secret Sharing
- Research protocols: RandHound, RandHerd
- Deployment: The League of Entropy (drand)

Talk Outline

- **Public Randomness: Introduction**
  - Challenges: Quality, Trustworthiness, Bias
  - General Approaches Known
- Background: Shamir Secret Sharing
- Research protocols: RandHound, RandHerd
- Deployment: The League of Entropy (drand)

Problem: secure public randomness

Vietnam War Lotteries (1969)

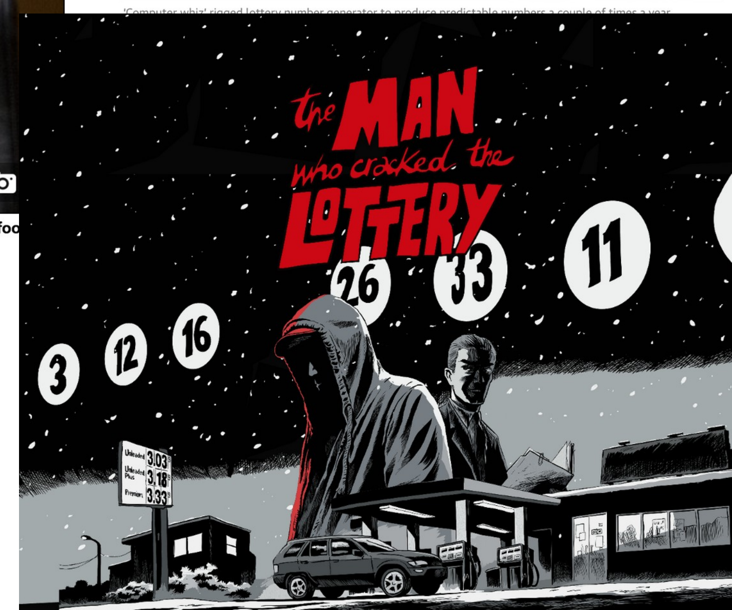


'European draws have been rigged': Ex-FIFA president Sepp Blatter claims to have seen hot and cold balls used to aid cheats

Former FIFA president Sepp Blatter said he had witnessed rigged draws for European foo competitions

Man hacked random-number generator to rig lotteries, investigators say

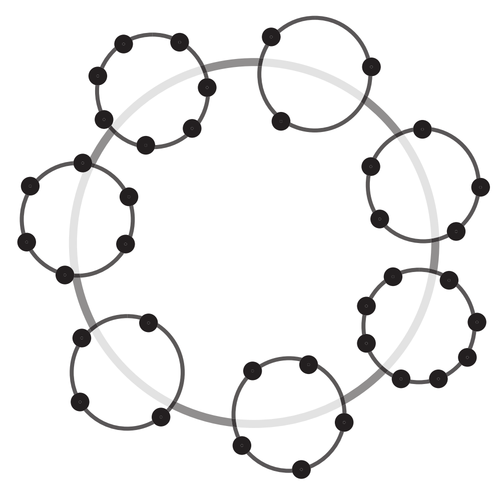
New evidence shows lottery machines were rigged to produce predictable jackpot numbers on specific days of the year netting millions in winnings



Some uses of public randomness

We need **fair** and **unbiased** “coins” for many purposes

- Choose a lottery winner fairly and transparently
- Fair sampling: e.g., risk-limiting audits of elections
- Pick representative quorums from large pools
  - e.g., for secure blockchain sharding (e.g., OmniLedger)
- Divide large user network into smaller random anonymity sets
  - e.g., Herbivore [Goel/Sirir '04]
- Proof-of-Stake blockchains
- …



Randomness: what can go wrong?

Some of the common failure modes:

- Mixing up **public** with **private** randomness
- Low-quality or low-entropy generators
- Trustworthiness: do you trust who flips coins?
- Bias: even if it’s random, is it **uniform**?

[Credit: Mike Izbicki]

Random Related Randomness

Some existing approaches:

- Random oracles: Cachin et al, PODC 2000
- Quorum-building: King et al, ICDCN 2011
- Slow hashes: Lenstra/Wesolowski, 2015
- Via PoW blockchains: Bonneau et al, …

This talk's focus: protocols that
preserve simple **t-of-n threshold model**

- Permissioned systems (e.g., drand/LoE)
- PoW/PoS with elected groups (e.g., ByzCoin)

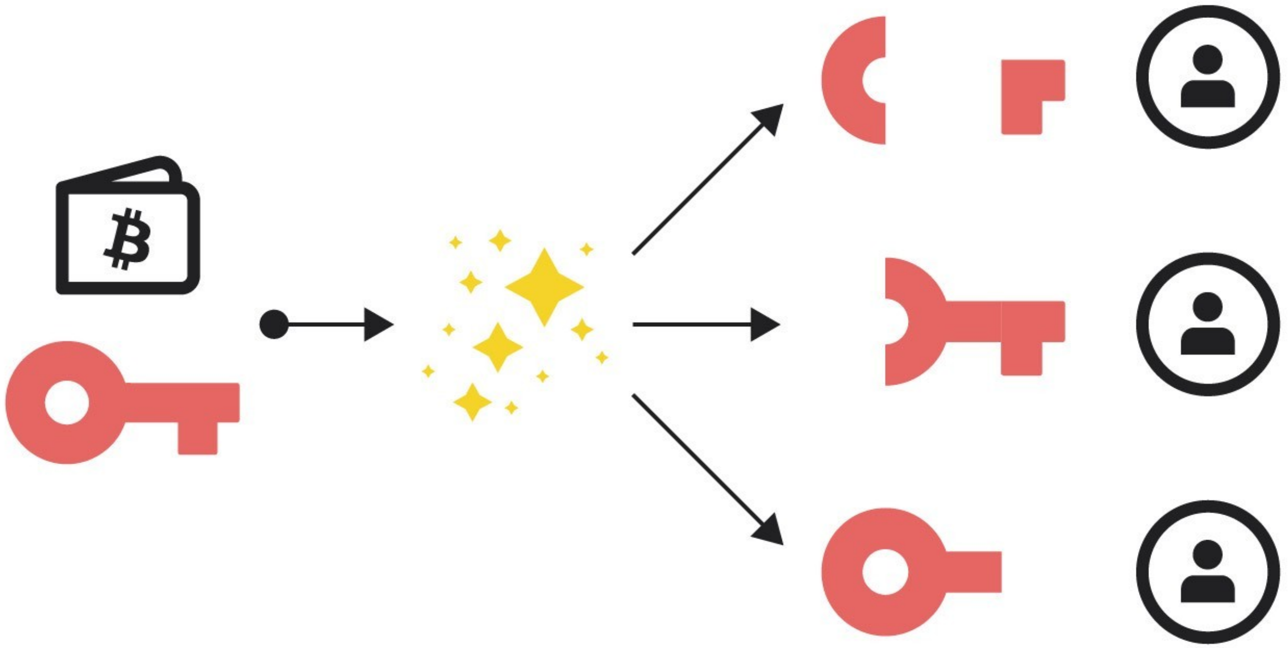
Talk Outline

- Public Randomness: Introduction
  - Challenges: Quality, Trustworthiness, Bias
  - General Approaches Known
- **Background: Shamir Secret Sharing**
- Research protocols: RandHound, RandHerd
- Deployment: The League of Entropy (drand)

Shamir Secret Sharing

The foundation of much **threshold cryptography**

- Threshold encryption, threshold signing, MPC
- Decades old, but little-used in blockchains



Shamir Secret Sharing

Basics: “deal” a secret to a threshold ***t*** of ***n*** parties

- Any ***t*** parties can cooperatively recover or use it
- If **<*t*** parties compromised, leaks *no* information!

Secret Sharing: Illustration

Suppose you're a pirate & bury your treasure…

Keeping the Location Secret

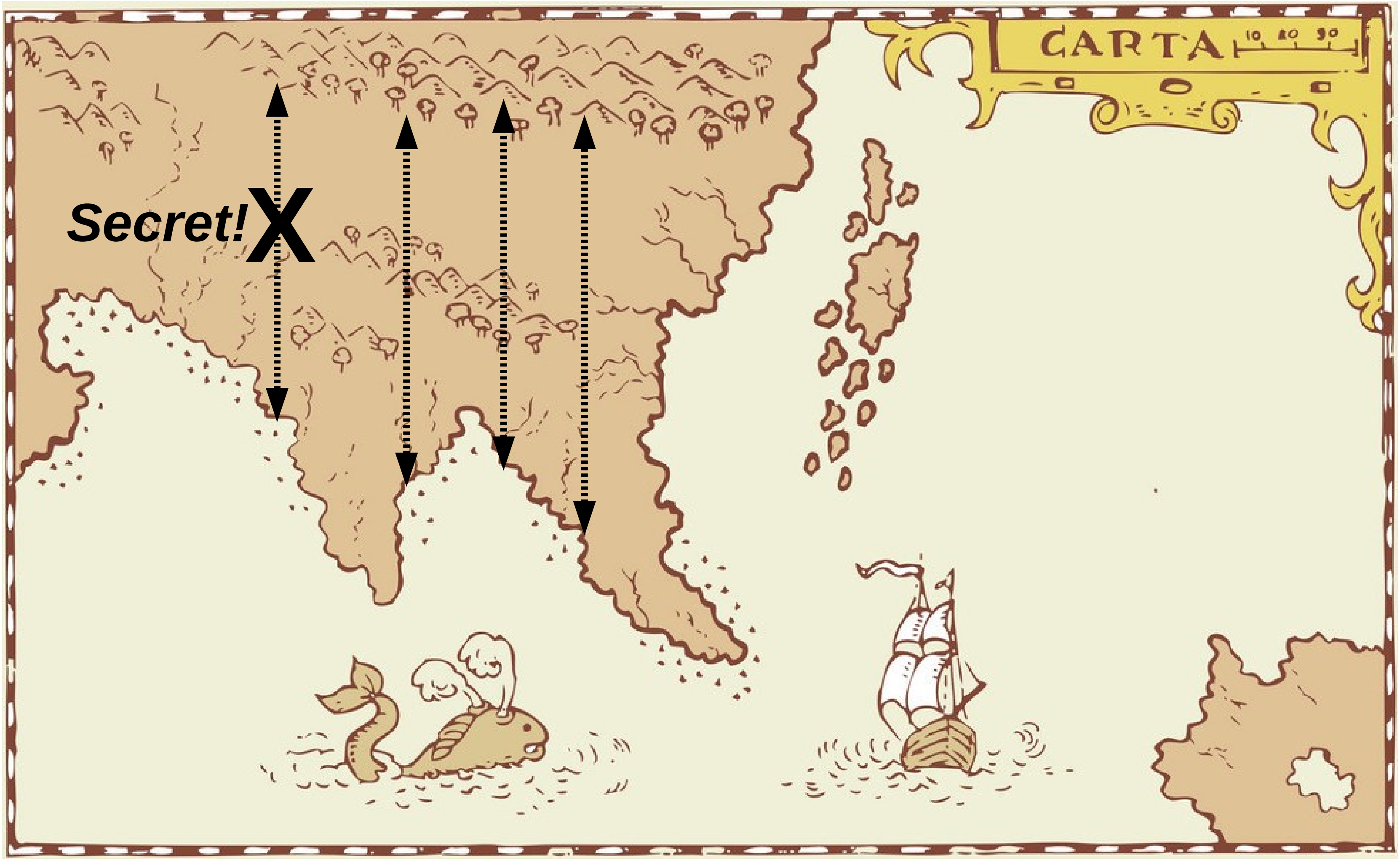
You have 3 henchmen who you want to send back for it later, but you don't trust *any one* completely

Secret Sharing: Illustration

You mark the spot between two reference points

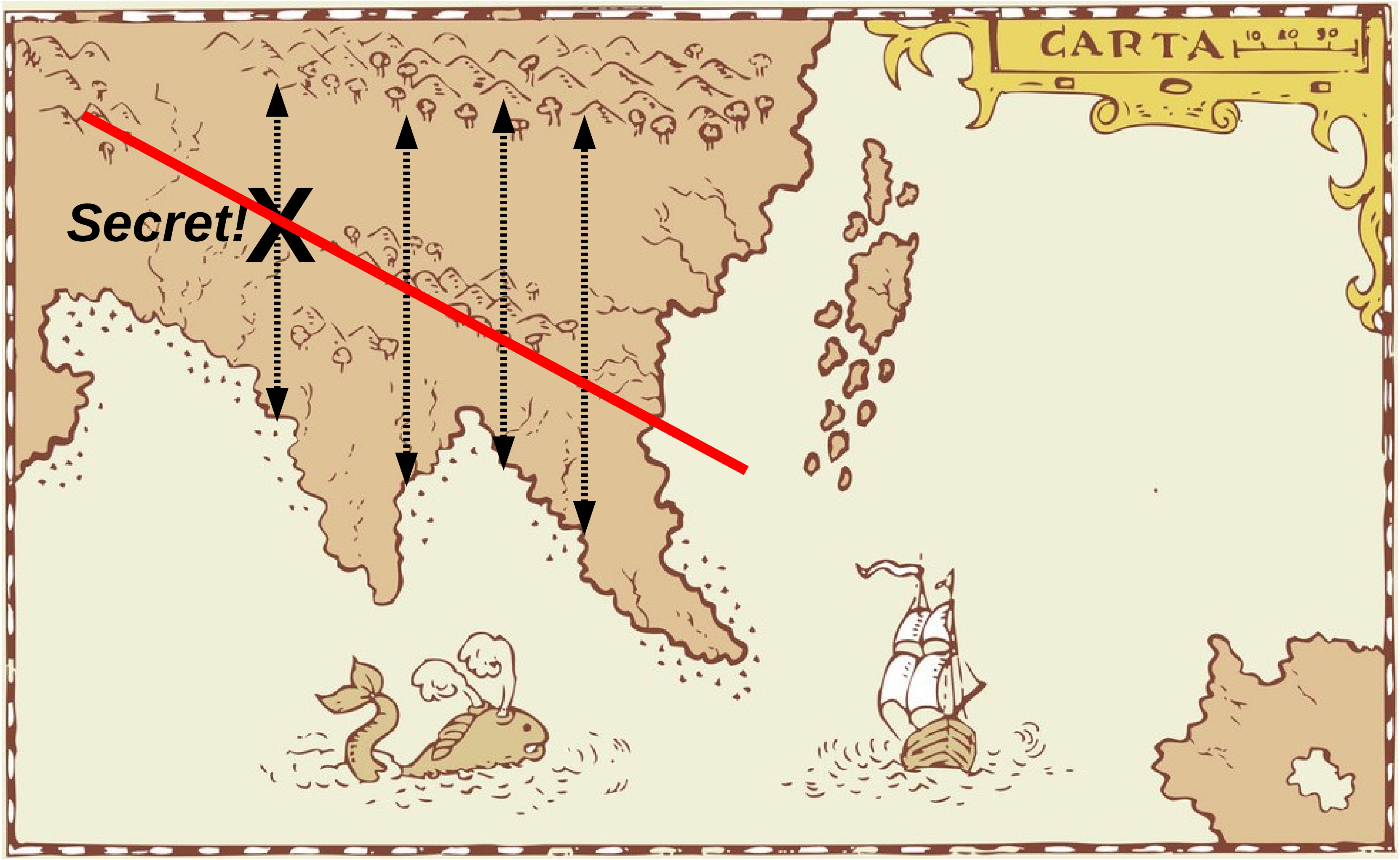
Secret Sharing: Illustration

Then draw three parallel reference lines…



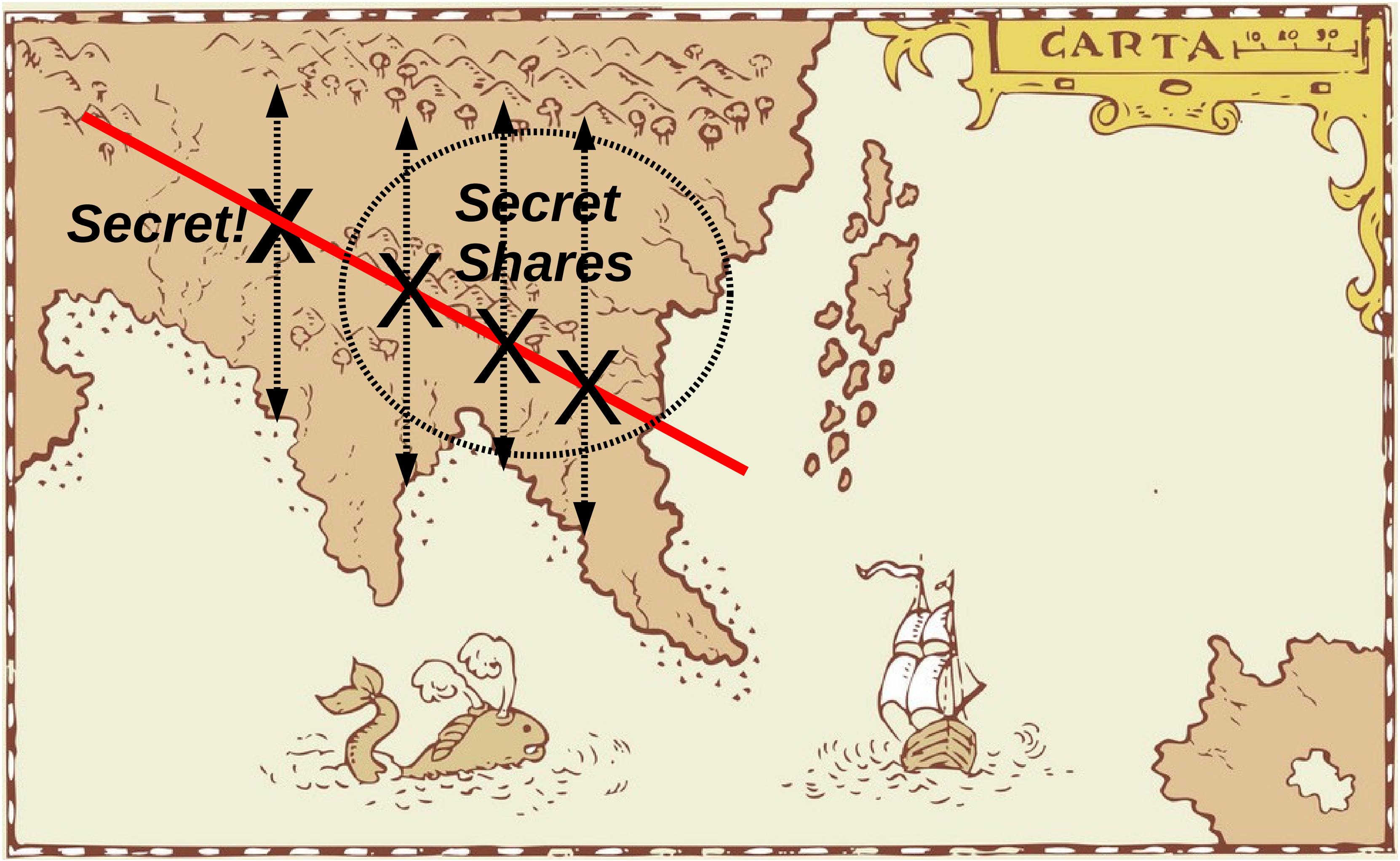
Secret Sharing: Illustration

…and another line intersecting all four…



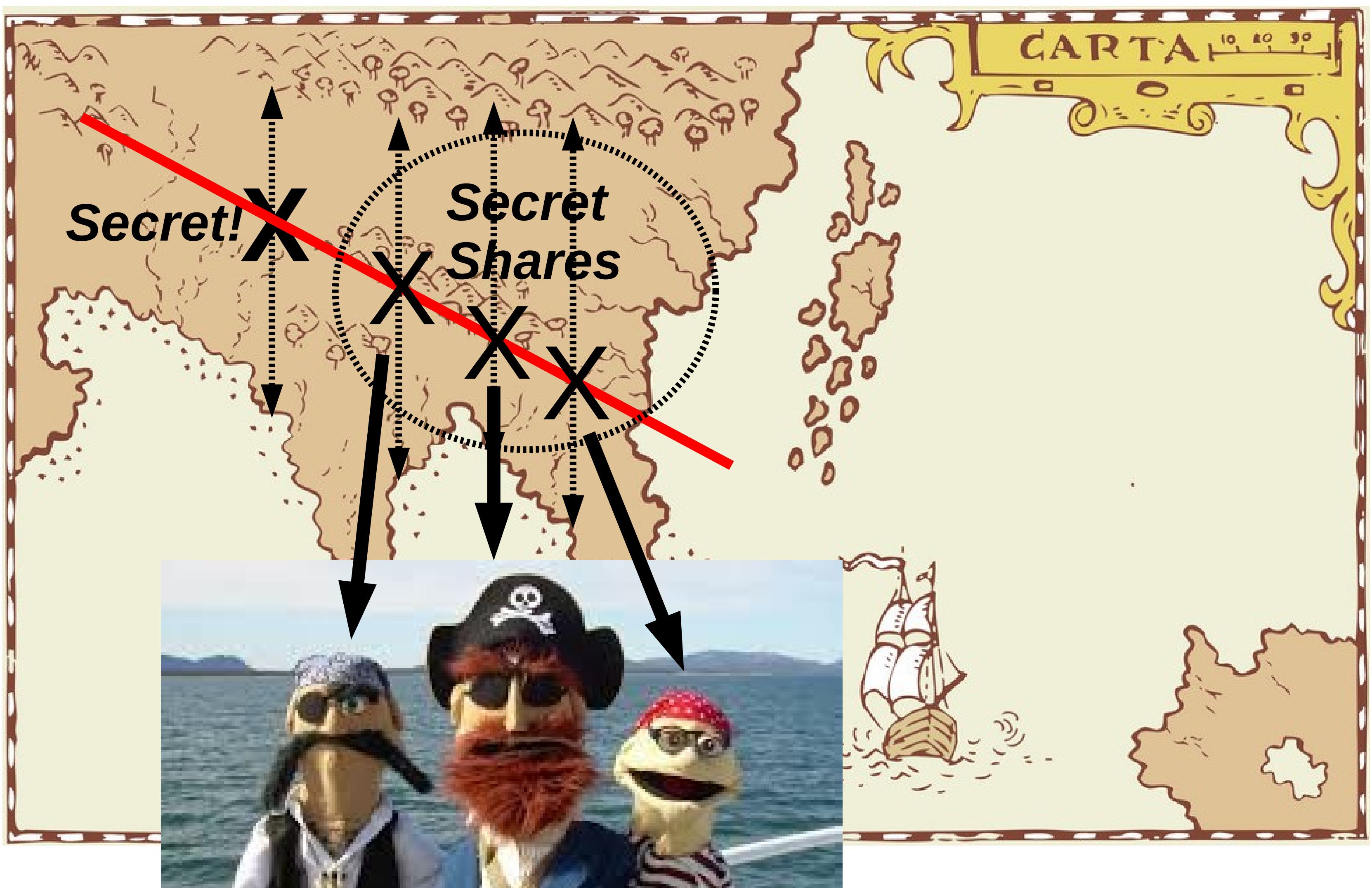
Secret Sharing: Illustration

The intersection points are the *secret shares...*



Secret Sharing: Illustration

You give *one* of these shares to *each* henchman



Threshold Secret Sharing

Now suppose your henchmen come back later to recover the treasure…

- Any **one** henchman won't know how to find it
- Any **two** henchmen together will be able to!

You get both **threshold privacy** of the secret…

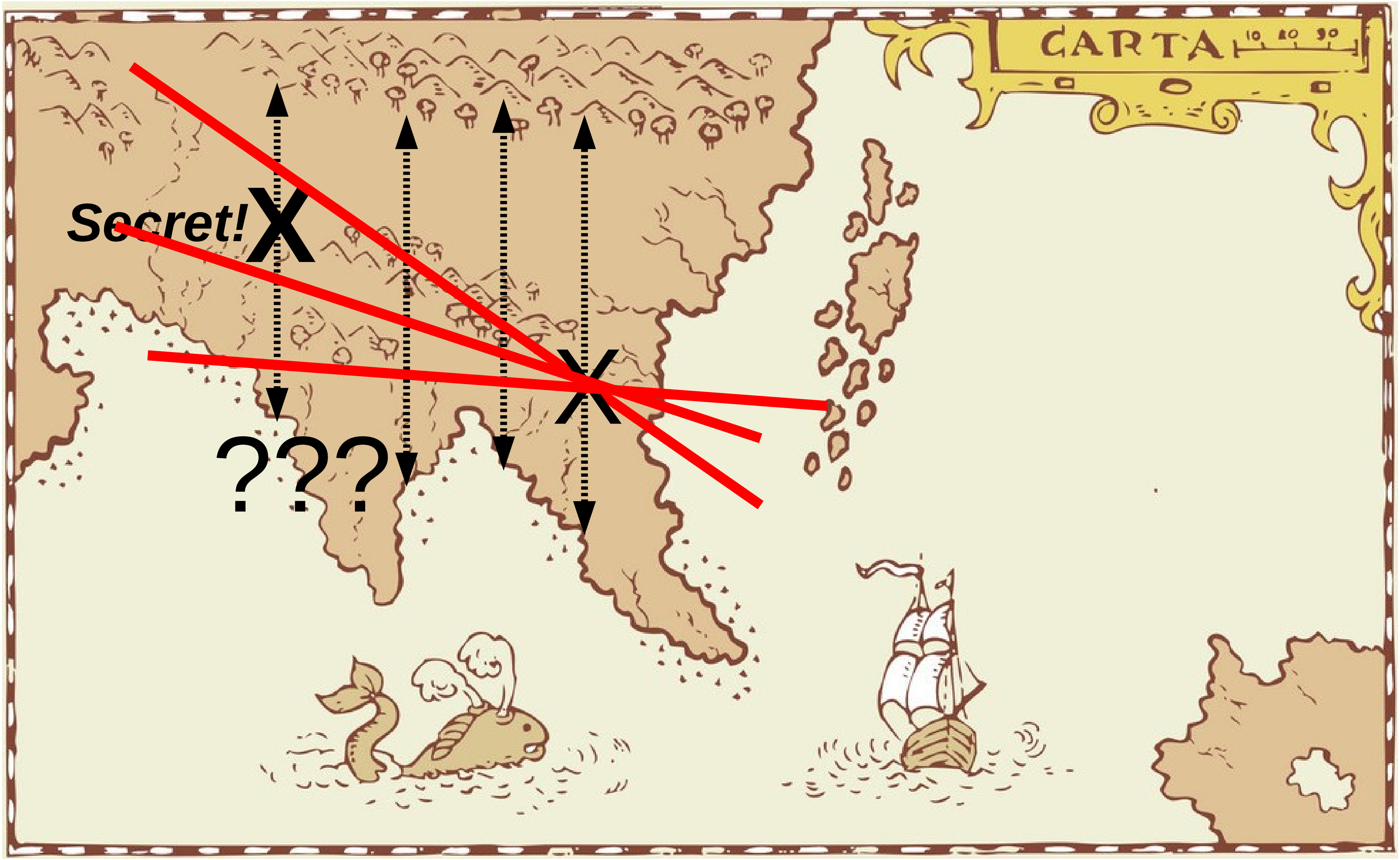
- No single compromised party can recover it

You also get **threshold availability** of the secret

- Can still recover if one henchman goes missing

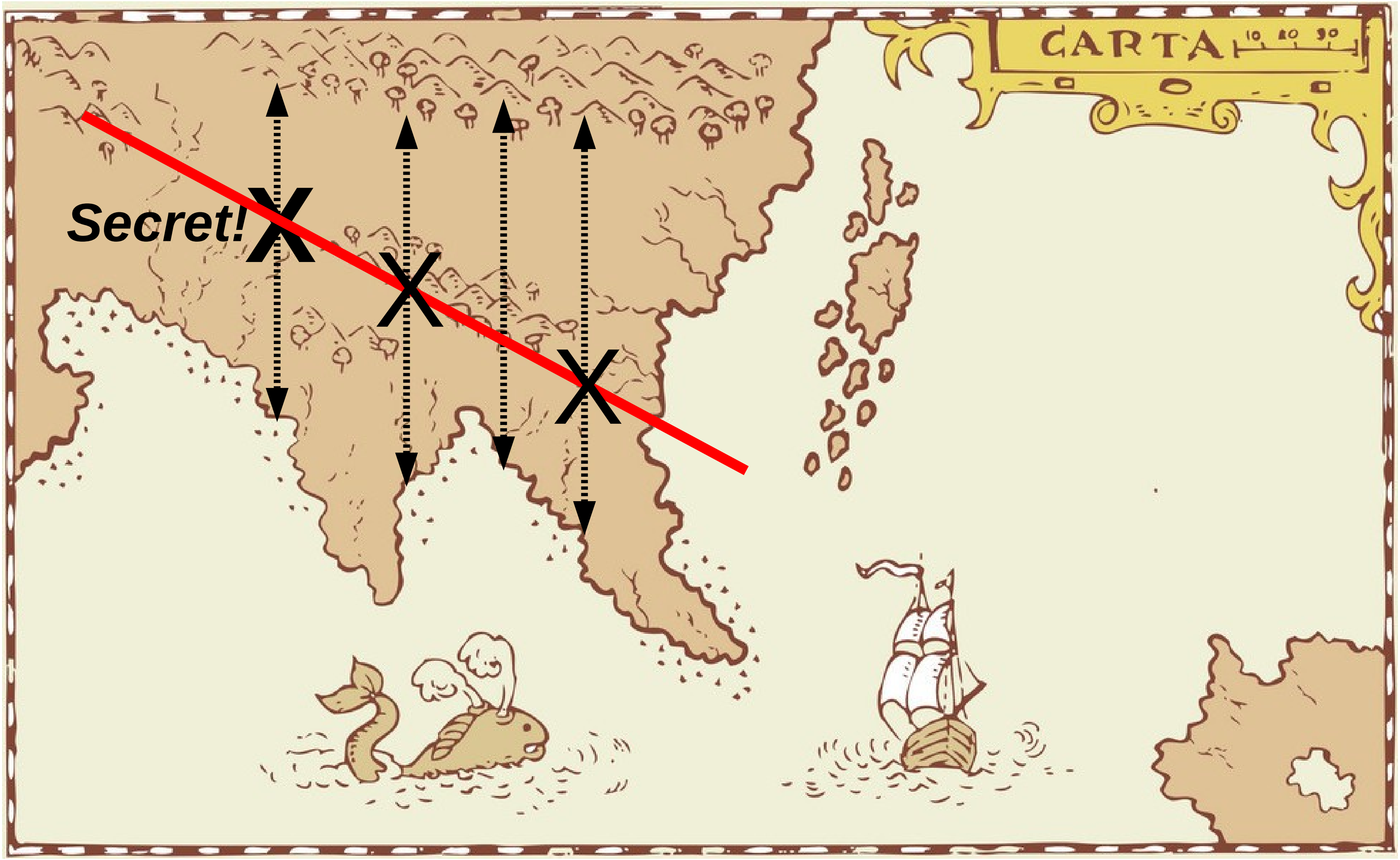
Secret Sharing: Illustration

One henchman alone can't recover secret



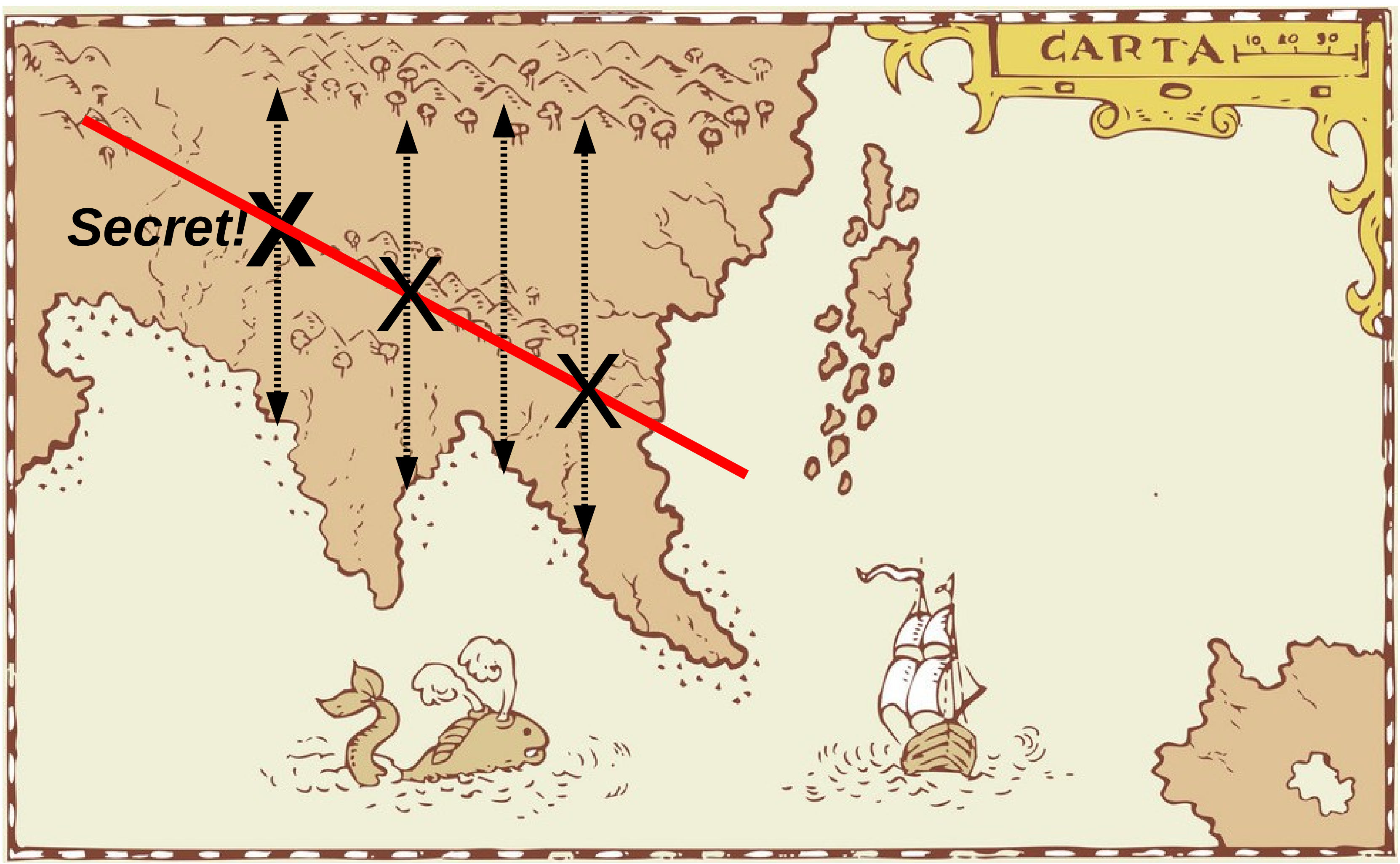
Secret Sharing: Illustration

…but *any two* working together can!



Secret Sharing: Illustration

…but *any two* working together can!

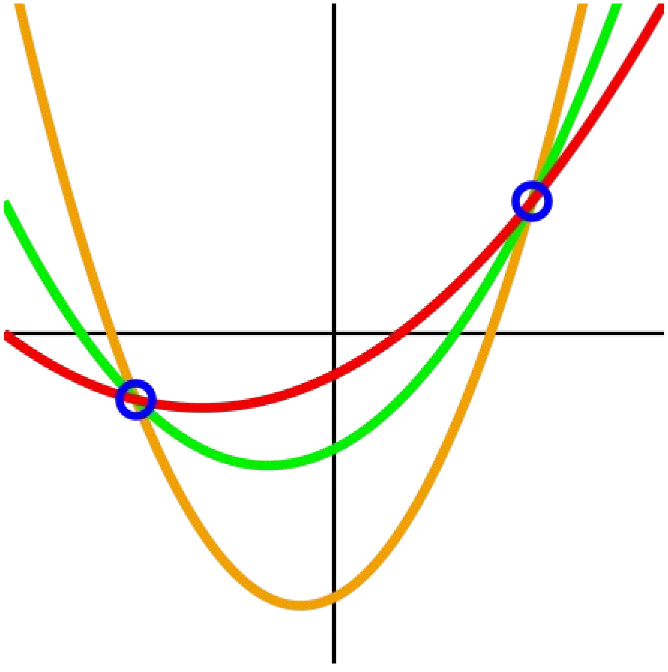


Supporting arbitrary thresholds

Just use higher-degree random polynomials

- Degree ***d*** polynomial yields threshold ***d*+1**

Example:
degree 2 (quadratic) polynomial →



Requires 3 points to reconstruct

Talk Outline

- Public Randomness: Introduction
  - Challenges: Quality, Trustworthiness, Bias
  - General Approaches Known
- Background: Shamir Secret Sharing
- **Research protocols: RandHound, RandHerd**
- Deployment: The League of Entropy (drand)

Threshold Randomness Protocols

Not at all new in the research community

- Cachin, “Random Oracles in Constantinople”

But secret sharing can be expensive, unscalable

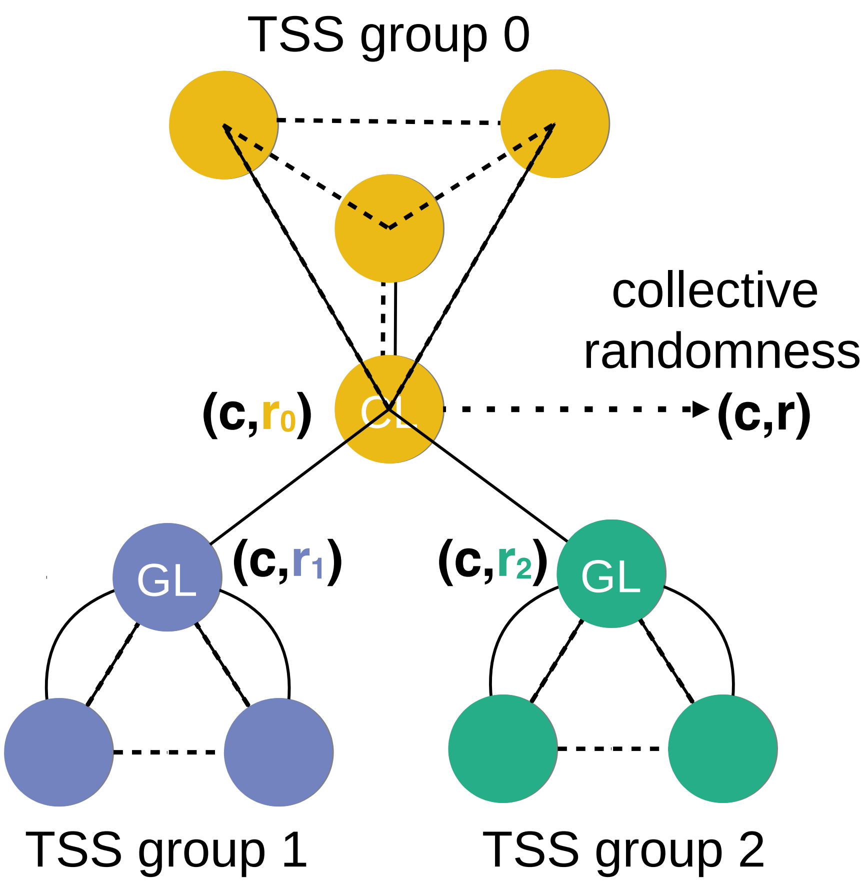
- Distributed key generation (DKG) setup needed
- Typical protocols **$O(n^2)$** communication cost, **$O(n^3)$** computation cost even *after* setup

DEDIS protocols *make public randomness scale*

RandHound/RandHerd

"Scalable Bias-Resistant Distributed Randomness" [IEEE Security & Privacy '17]

- Standard ***t***-of-***n*** threshold model
- Efficient, scales to thousands of parties
- Compatible with ByzCoin group election, OmniLedger sharding



Strawman 1: Commit-and-Reveal

1. Each of n nodes pick a random secret s_i, broadcast a commit to secret, e.g., $C_i = H(s_i)$
2. "Everyone" reveals their secrets s_i, combines to form final output, e.g., $s = \Sigma_\mathrm{i}(s_i)$

Problem: vulnerable to either DoS or bias attacks

- Require *everyone* to reveal → DoS attacks
- Tolerate up to f missing secrets → attacker can choose favorite of 2^f outcomes!

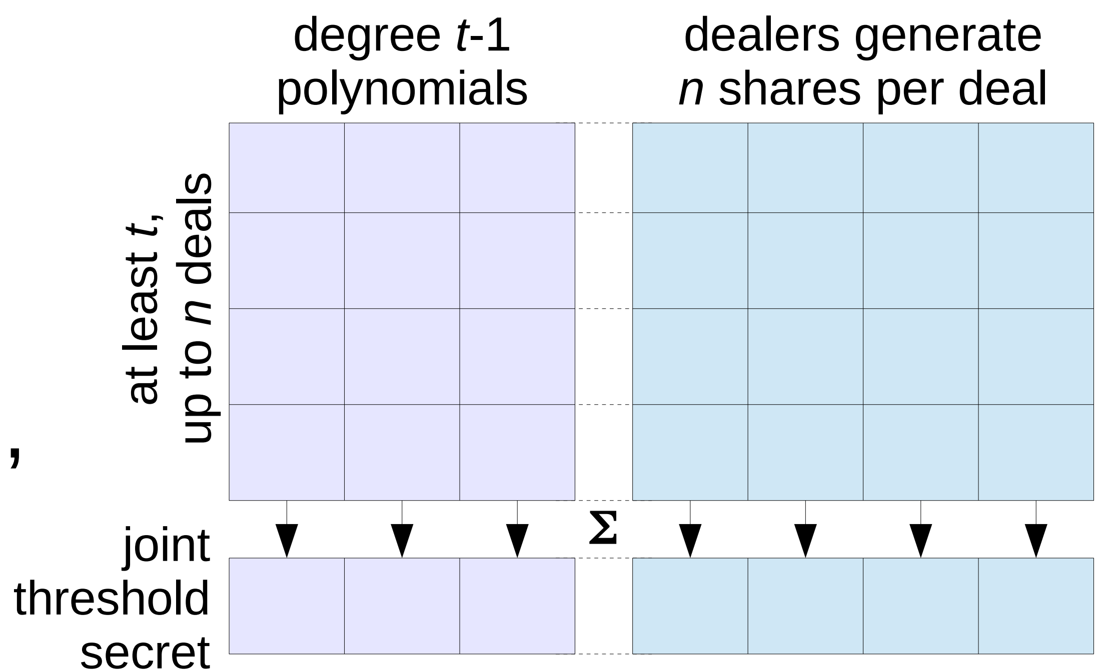
Strawman 2: Shamir Secret Sharing

- Each of n nodes "deals" secret s_i all n nodes via t-of-n publicly verifiable secret sharing (PVSS)
- Agree (BFT) on at least t of these secret deals
- Homomorphically sum polynomials and reveal

Works, secure! ☺

- [Cachin et al, …]

O(n^2) communication,
O(n^3) compute ☹



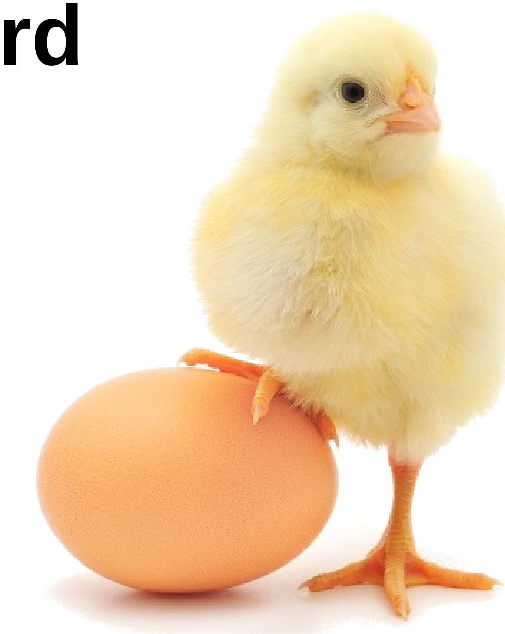
The Chicken-and-Egg Problem

More scalable if we could use *smaller groups*…
but need randomness to *sample* them securely!

- Sharding needs randomness needs sharding

Addressed by **RandHound**, **RandHerd**

- **RandHound**: bootstrap protocol, O(*n* log *n*) efficiency
- **RandHerd**: repeating beacon, O(log *n*) cost/node/round



RandHound: Key Intuition

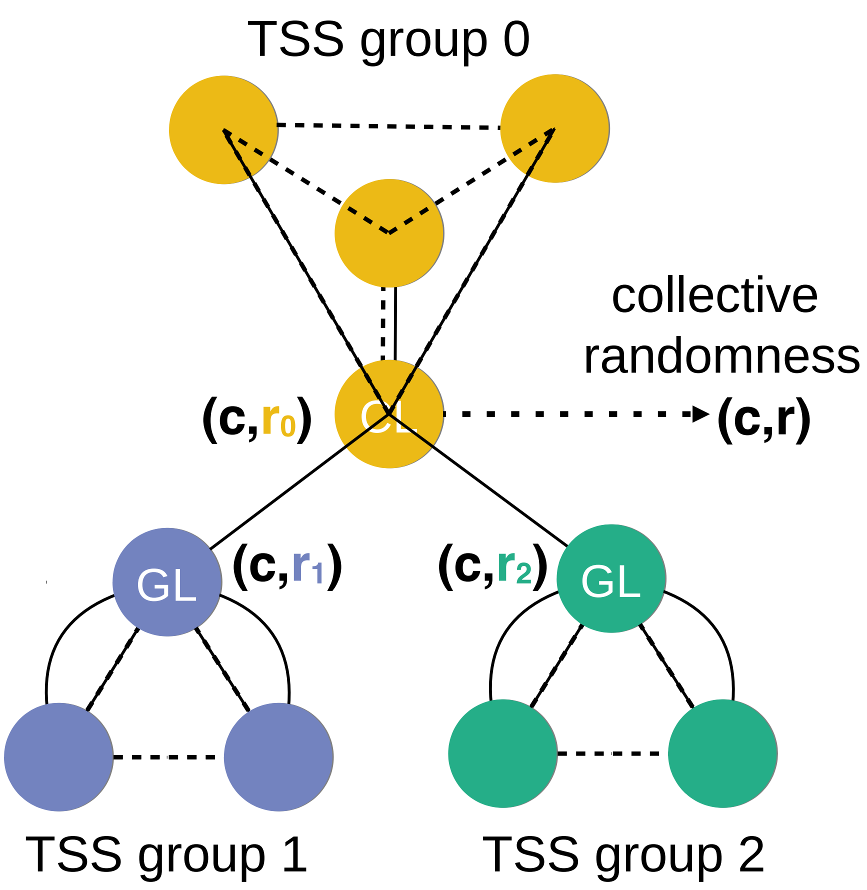
A RandHound client initiates a “scavenging” run

- Assumes initiator wants trusted randomness
  → “trusted” for **liveness** but not for **integrity**
  - Initiator gets *only one try* to produce an output
- Initiator picks subgroups, randomly if honest
  - Subgroup size is a security parameter, O(log *n*)
  - Each subgroup runs PVSS commit-and-reveal
  - Threshold of **each** subgroup contributes to output
  - Pigeonhole principle → *at least one* subgroup good
  - Bad initiator can’t compromise output, only self-DoS

RandHerd: Key Intuition

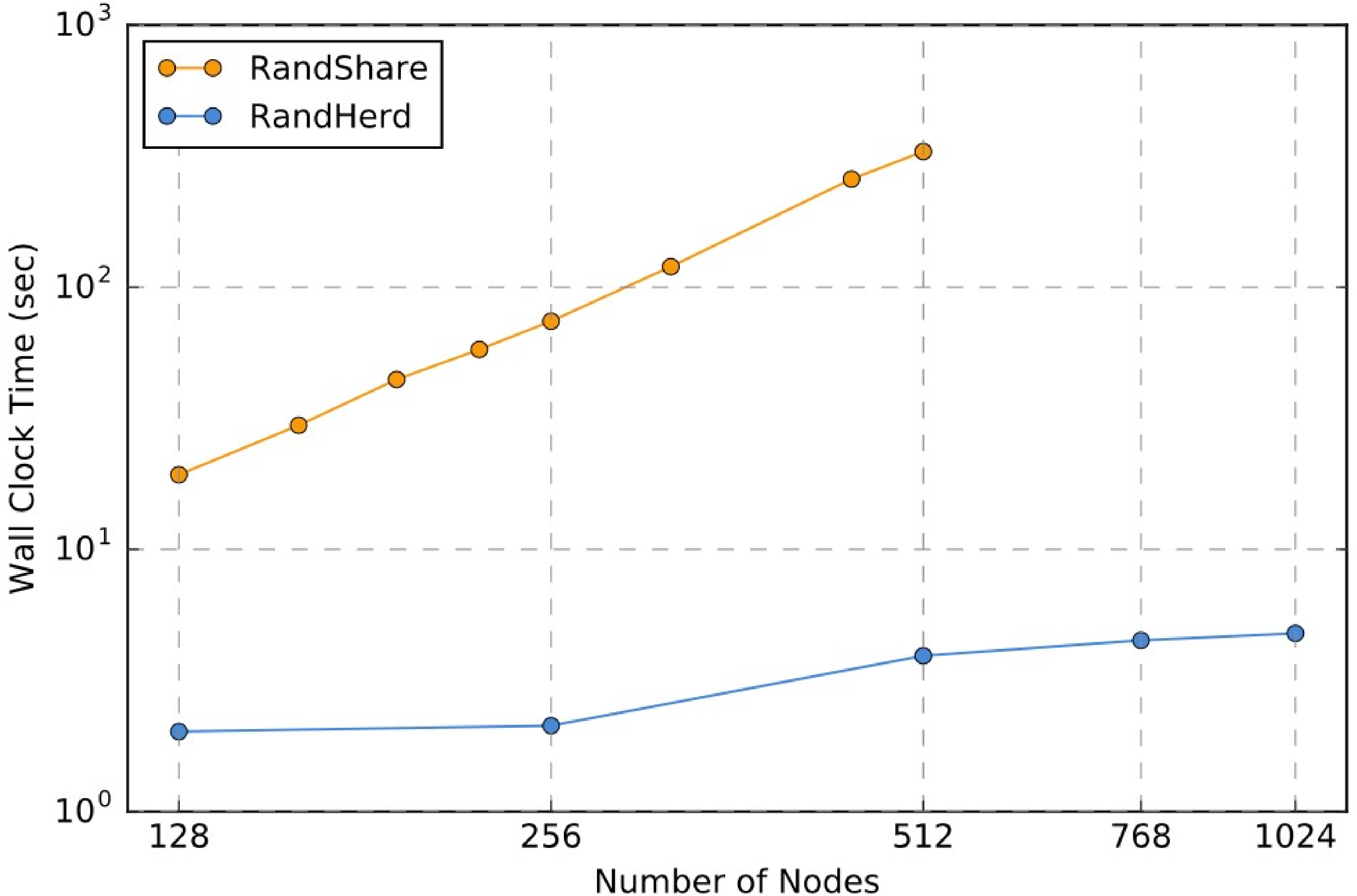
Efficient decentralized **randomness beacon**: stable group producing new output every few secs

- Uses PBFT-style leader election, view changes
- Leader uses **RandHound** to bootstrap a new view
  - Success → good random output forms subgroups
  - Failure → view change
- Then fast/cheap rounds



RandHerd Performance, Scalability

Compared with baseline PVSS “Strawman 2”



Talk Outline

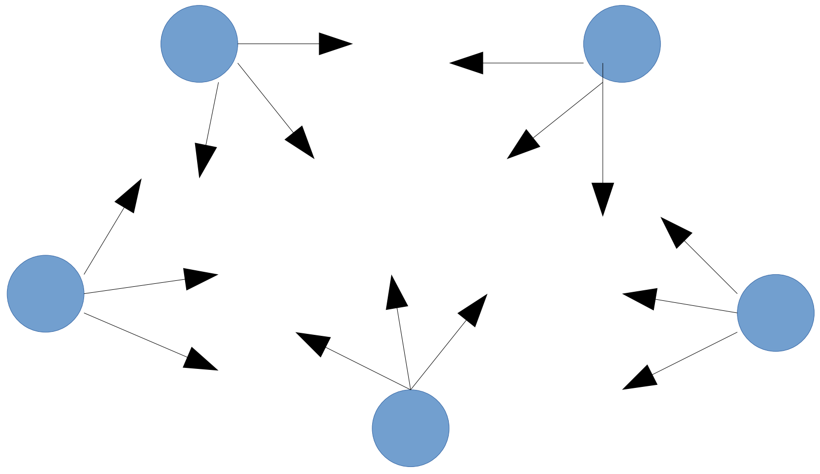
- Public Randomness: Introduction
  - Challenges: Quality, Trustworthiness, Bias
  - General Approaches Known
- Background: Shamir Secret Sharing
- Research protocols: RandHound, RandHerd
- **Deployment: The League of Entropy (drand)**

Challenges to practical system

- Distributed systems are hard
- Randherd: View change protocol is hard to implement in practice
- CoSi tree is difficult to maintain in practice in case of failures
- Delay to reconstruct private key via VSS

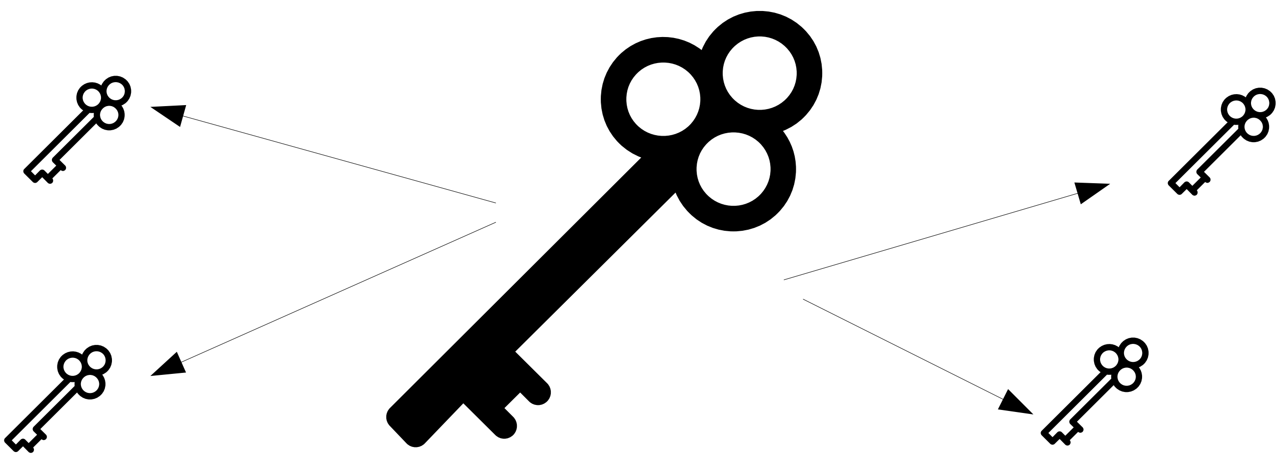
Drand: simplified randomness generation

- **Distributed key generation** *similar* to randherd as an initial ceremony between nodes
- Then periodical "one round" randomness generation protocol via ***pairings & BLS signatures – no leader, no tree***
  - Each node simply **broadcasts** a partial signature



Distributed Key Generation

- Allows to distributively generate **x** such that
  - No nodes know **x, no leader**
  - A subset of nodes is needed to reconstruct **x**
- **Simple DKG:** each node does a VSS with the rest then nodes adds all its shares it received.
  - Share can be used to create partial BLS signature

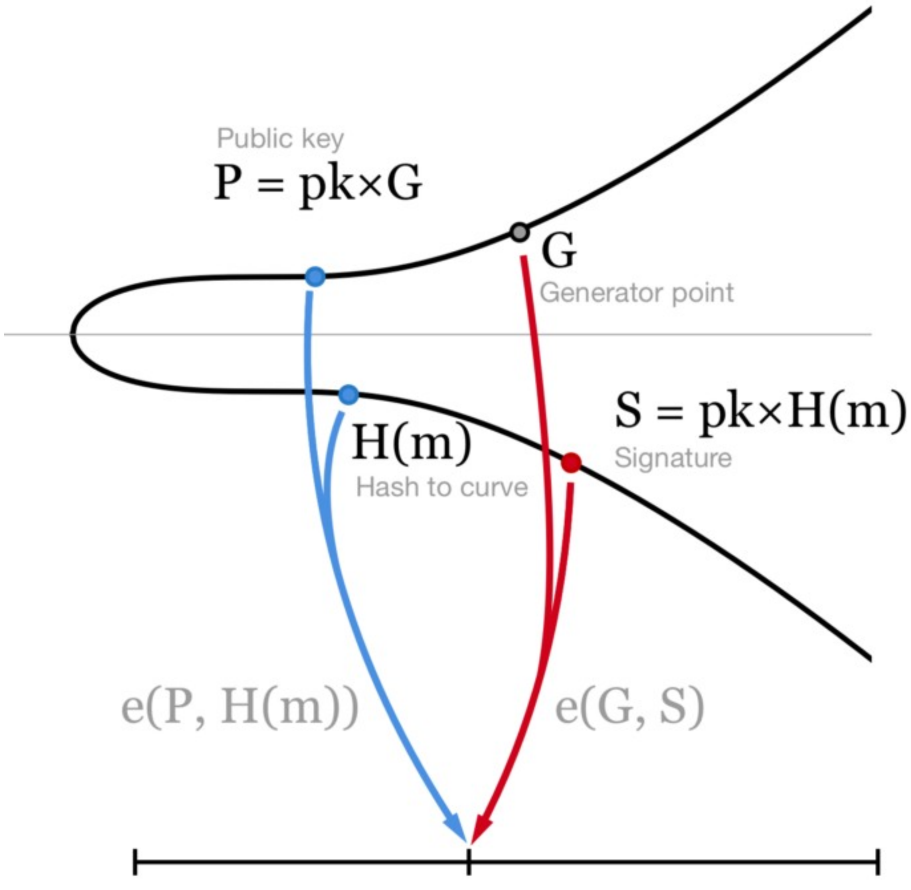


Randomness: Pairing curves

- Special structure of curve called “pairing friendly curves”
- The “pairing” mapping is
  - e: G1 x G2 → GT
  - **Key property:** e(g1^a, g2^b) = e(g1,g2)^ab
  - Allows to solve *Decisional Diffie Hellman* problem !

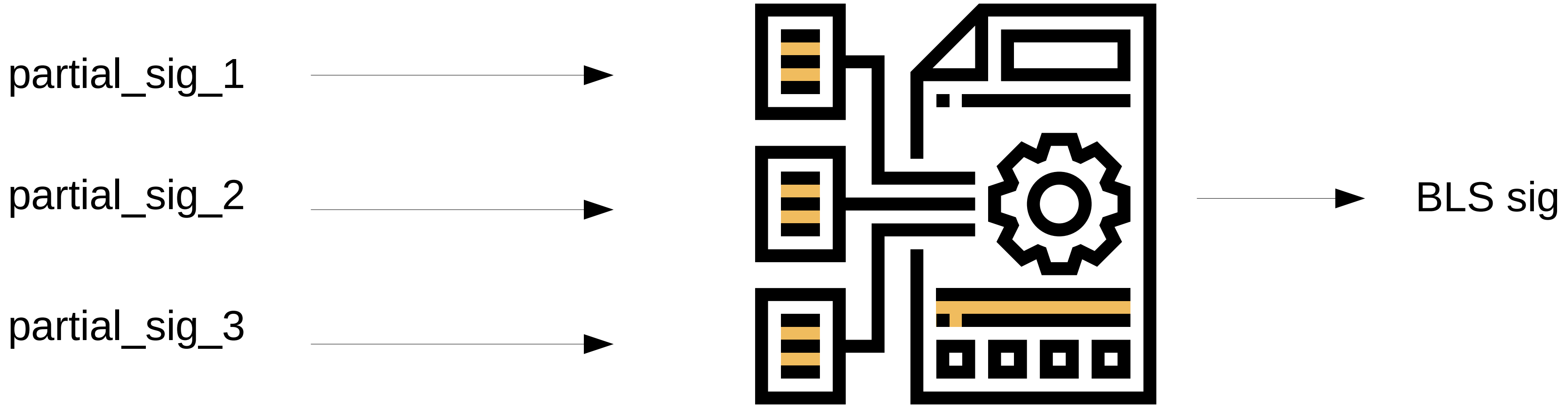
Randomness: BLS signature

- BLS Signature [Boneh et al. '04] is a *unique* signature scheme
- Private key: **x**, public key: **X,** generator: $\mathbf{G_1}$,$\mathbf{G_2}$
- **Sign**: sig = H(m)^**x**
- **Verify**:
  - e(H(m), **X**) =?= e(sig,$\mathbf{G_2}$)



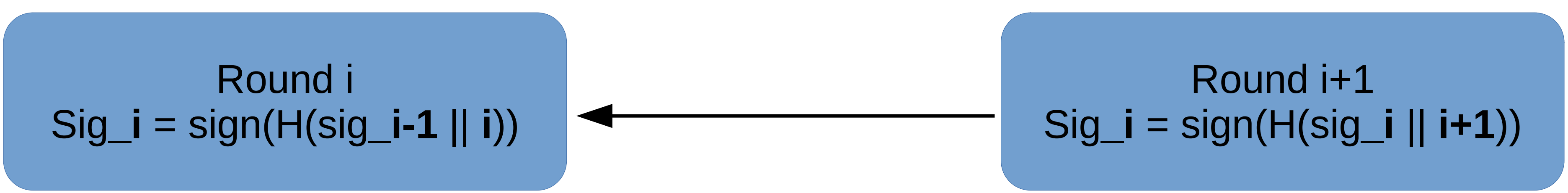
Randomness: Threshold BLS

- A BLS private key can be secret shared !
- Creation of a BLS signature with partial signatures
  - **PartialSign**: H(m)^$\mathbf{s_i}$ where $\mathbf{s_i}$ is the share of node i
- Node aggregate locally the final signature
  - "Aggregation" is Lagrange interpolation



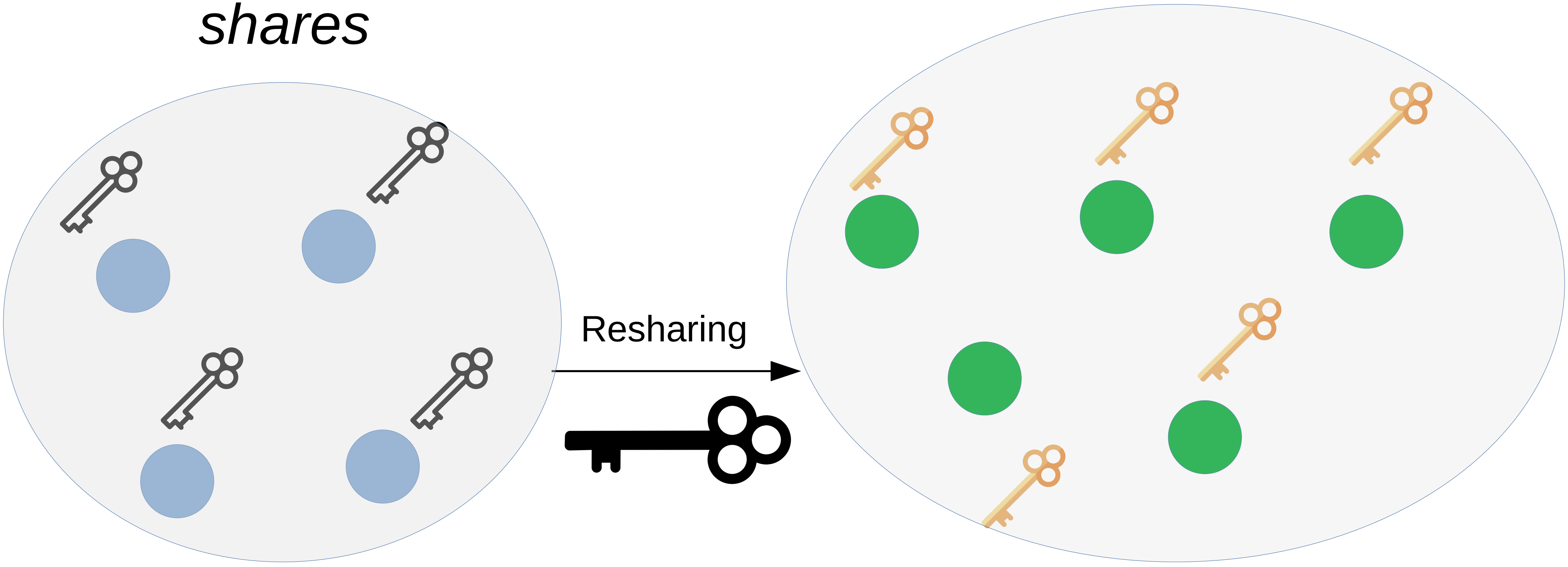
Drand: chain of randomness

- Nodes form a ***unbiasable*** *chain of randomness*
  - Deterministic mapping round ↔ timestamp
  - At round i, broadcast
    - $sig_{i,j}$ = PartialSign(H(sig_{i-1} || i-1))
  - Randomness is H[sig_i = LAG(any t $sig_{i,j}$)]



Drand: dynamic resharing

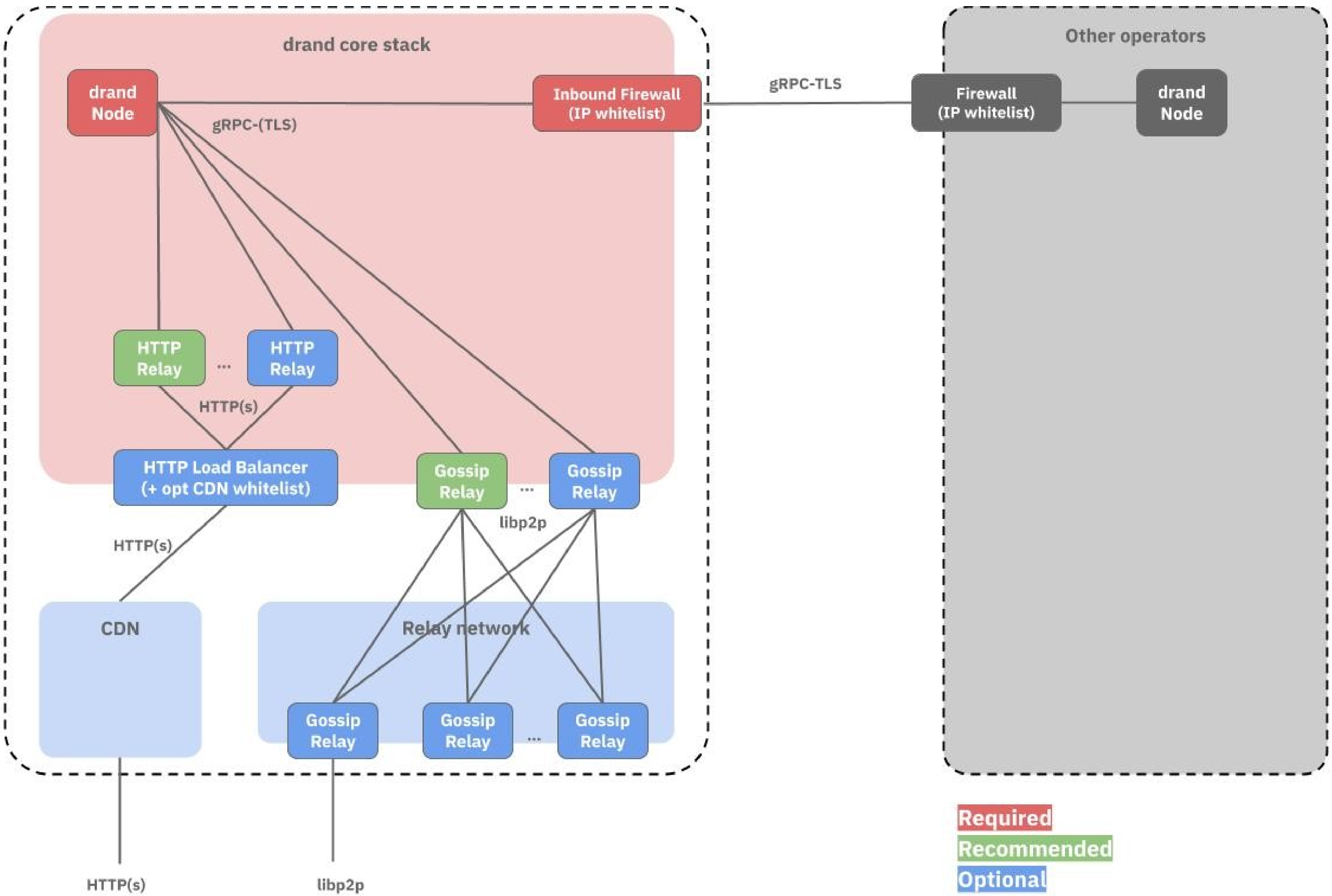
- Need to adapt the group of nodes over time
- Use a variation of DKG to ***reshare*** to a new set of nodes
  - The ***same*** *distributed key* is used but with **new** *shares*



Drand: the project

- Project started at DEDIS
  - Based on collaboration with DFINITY’s BLS based beacon
- Only do one job, free, open source & no blockchain !
- Now moved to github.com/drand/
- Team working at Protocol Labs
- Software written in Go
- Curve is BLS12-381

Drand: production network



The League of Entropy

Public network of multiple individual organizations running drand.

* https://leagueofentropy.com

More news coming soon.

“Randomness as a Service” network:

- Similar to NTP servers, CT servers, etc

League of Entropy: Members

- Initially launched by EFPL-DEDIS, the network has grown since !

League of Entropy: try!

DEMO:

```
homer:~:% curl -s https://drand.cloudflare.com/public/latest | jq
{
  "round": 22624,
  "randomness": "1d0f5d64cc83707344ffdde1f7aa76c26574babe2655b8ac83428
a2af4826042",
  "signature": "a5264e0a1929ff826722f29ee01771b7c8bdb1856d69521e0f6324
053cbef68686a0862183da18dfa0344931ea05d4cf1790ec96525f7aa1460ffe74a06b
ca5cecfa06cd4864c267207a6f5fc28893956ade27e234a18e80d904c08f2ab46d9e",
  "previous_signature": "a23318523749b484218b001b91c6d847d9b2f3e5aedda
6ee10997fc536de72838893ece563f6859823e917c1c7047d040cb3e4c8fa40458fb82
5bb68aaef4f02a8f19cc88ebd5764de536bf1fb7aba144758fabaec3f9375371c199f2
95edd51"
}
```

Conclusion

Threshold public randomness provides secure, unpredictable, unbiased, uniform coin flips

- Provided fewer than ***t*** of ***n*** parties compromised

RandHound/RandHerd show that this can **scale**

League of Entropy (drand) makes it practical **now**!