Timelock Encryption based on drand

Protocol Labs

Yolan Romailler @anomalroil yolan@protocol.ai

Intro

Digression: What is “randomness”?

According to the Cambridge dictionary, “randomness” is:

Digression: What is “randomness”?

According to the Cambridge dictionary, “randomness” is:

“the quality of being random”

The notion of randomness

- We have some kind of intuition of "what is random", usually.
- **Randomness is hard**, and we're not using "true random generators", but instead have "pseudo random generators".
- Also, randomness has different *flavours*: secret, public, verifiable, distributed. Picking the right one for your needs is important!

What are *public* & *secret* randomness?

- Public randomness is simply a **random value that is meant to be *public***, e.g. for a lottery, or a jury election.

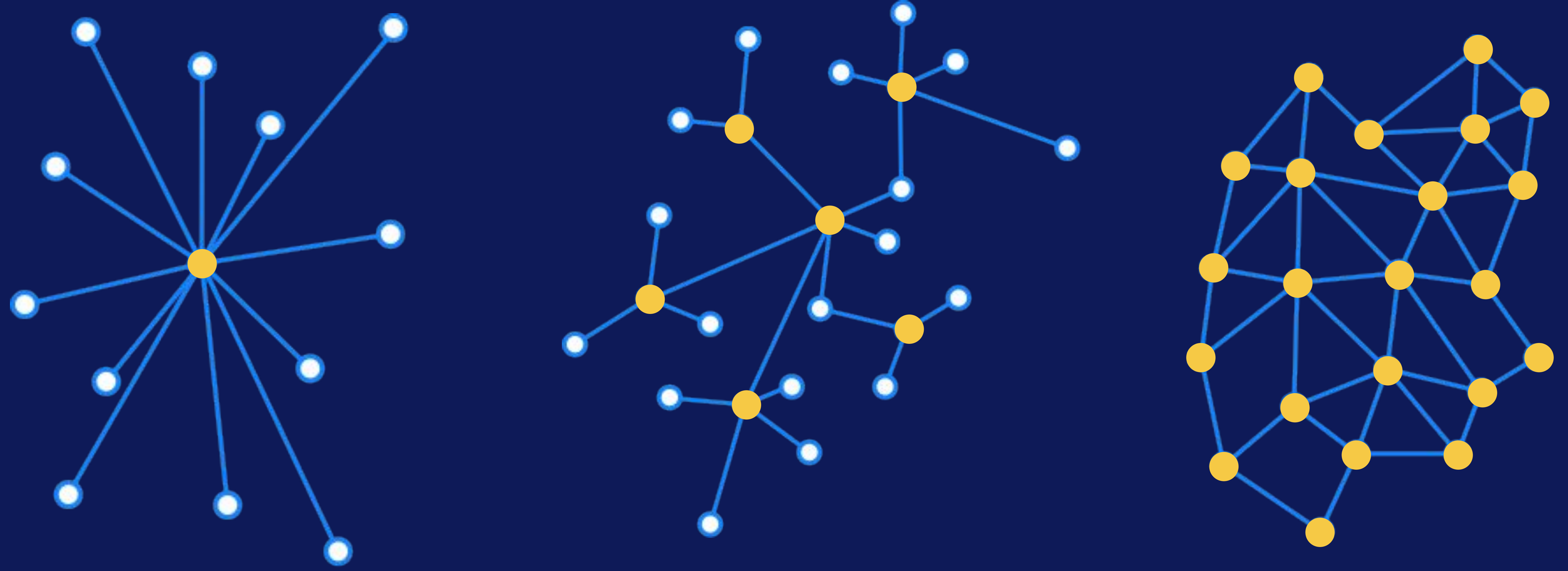
- This is the contrary of the randomness we're most used to, which is the **"secret" randomness**, meant to be used as secret keys, as nonce, as IV, etc.

How about *public & 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 actually **verify** that it was properly issued and not manipulated after it’s revealed.

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

Since 2019: drand is a public randomness service

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

➡️ **how about public randomness?**

Drand is meant to be a foundational Internet protocol for randomness:

a highly available, decentralized, and publicly verifiable source of randomness

Drand properties

- Drand is a software ran by a set of independent nodes that collectively produce randomness
- Drand is **open source**[1]
- **Decentralized randomness service** using
  - **Threshold** cryptography based on **pairings**
  - Verifiable secret sharing, Distributed Key Generation, BLS
- Binds together **independent entropy sources** into a publicly verifiable one
- Tested, **audited**, and deployed at scale

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

What is the League of Entropy?

16 organizations

2M+ rounds

23 nodes

2+ years

4 endpoints

0 disruptions since 2020-08-10

0 single point of failure

What is timelock encryption?

Timelock Encryption

Exactly what it sounds like: being able to **encrypt** *toward the future*

Sometimes also referred to as:

- Timelapse encryption
- Timed release encryption

Applications

- Bids in sealed-bid auctions
- Can help prevent MEV issues
- Conditional transfers of wealth
- Could help with Electronic Voting
- Issue documents with a **known embargo period**

Applications: for fun and profit version

- Responsible Ransomwares:

  “Pay now to get the decryption key, or wait 6 months.”
- Escaping emulation:

  “Wait until time X has lapsed to decrypt the payload.”

Prior art: ideation

Timelock Encryption was initially submitted by Tim May in 1993 on the Cypherpunks mailing list.

He introduced the idea of relying on a pool of **trusted third parties** to release a **sealed decryption key** at the proper time.

Prior art: Rivest Time-lock Puzzles

In **1996**, Rivest, Shamir and Wagner proposed a “**proof of work**” way to achieve Time-lock encryption.

“There are 2 natural approaches to implementing timed-release crypto:

- Use “time-lock puzzles”—computational problems that can not be solved without running a computer continuously for at least a certain amount of time.
- Use trusted agents who promise not to reveal certain information until a specified date.”

Prior art: Rivest Time-lock Puzzles

In **1999**, using their 1996 scheme, Ron Rivest created a time capsule, **LCS35**, commemorating the MIT Computer Science and Artificial Intelligence Laboratory; he expected that puzzle to take ~35 years to complete:

https://people.csail.mit.edu/rivest/lcs35-puzzle-description.txt

Prior art: breaking Rivest Puzzle

Rivest Puzzle was actually solved independently by 2 different implementations in **2019**, only 20 years later:

- In April, using a GNU GMP squaring routine for ~3.5 years on a single i7-6700 CPU core (~4Ghz).
- In May, using a FPGA implementation with very low latency, it only required two months for the Cryptophage collaboration between the Ethereum Foundation, Supranational, and Protocol Labs to solve it.

Prior art

- Equivalent to Bitcoin POW, using hashes:
  https://github.com/petertodd/timelock
  https://github.com/dorianj/timelock
- Timed Commitments:
  https://link.springer.com/chapter/10.1007/3-540-44598-6_15
- Provably Secure Timed-Release Public Key Encryption:
  https://dl.acm.org/doi/10.1145/1330332.1330336
- Time-lapse cryptography:
  https://www.eecs.harvard.edu/~cat/tlc.pdf
- "Time-Lock Puzzles from Randomized Encodings":
  https://eprint.iacr.org/2015/514.pdf
- "How to build time-lock encryption" introducing the notion of "Computational reference clocks":
  https://link.springer.com/article/10.1007/s10623-018-0461-x

Problems with all prior art

Most of the proposals are:

- Burning the planet with **proof of work**-like systems (Puzzles, Bitcoin-based, etc.) sensitive to ASICs.
- Cutting-edge cryptography that's not battle-tested
- **Impractical** (obfuscation and homomorphic cryptosystems are too slow nowadays in general)
- **Never actually deployed** in practice, because they require a dedicated service.

In practice

Encrypt towards the future

drand rounds maps to a precise time!

How to?

Using pairing-based and identity-based crypto we can achieve this without any single trusted third party! → same trust assumptions as for the League

Drand is relying on the **BLS pairing-based signature scheme**.

Using **identity-based encryption** we can encrypt towards the (future) signature of a **given message**.

Maths stuff:

How does it work?

Super easily (just kidding, see next slides): $\phi = V \oplus H_2(\hat{e}(U, \sigma))$

$$= \phi \oplus H_2({\mathcal{G}^k}_r) \oplus H_2(\hat{e}(U, \sigma))$$

$$= \phi \oplus H_2(\hat{e}(Pub, Q_r)^k) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi \oplus H_2(\hat{e}(s\mathcal{G}_1, H_1(\mathbb{H}(r)))^k) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

By definition of bilinear pairings, we get

$$= \phi \oplus H_2(\hat{e}(\mathcal{G}_1, H_1(\mathbb{H}(r)))^{ks}) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r)))) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi$$

Maths stuff:

drand primer

Drand generates BLS signatures in a threshold way.

Let P be the public key associated with the network, and s_i the share of this public key that belongs to the node i.

$$P = sG_1 \in \mathbb{G}_1$$

Where s is the free coefficient of the interpolated polynomials of any t shares s_i, i.e. the distributed secret key.

Maths stuff:

drand primer

At each epoch ρ, the drand networks generates a BLS signature over the message e mapped in F_q — More specifically, each nodes generates a partial BLS signature in the following way:

$$\pi_i = s_i * H_1(\rho) \in \mathbb{G}_2$$

while $H_1 : F_q \rightarrow \mathbb{G}_2$ is a secure hash function.

Maths stuff:

drand primer

Then the final signature π is interpolated using the Lagrange basis polynomials $L_i(x)$:

$$\pi = \left(\sum_{i}^{t} \pi_i * L_i(x) \right)(0)$$

for any subsets of partial signatures of size t.

More maths:

Our timelock

Epoch: We map the Q_{id} in the paper to the epoch ρ mapped to G_2 via

$$Q_{id} = H_1(\rho) \in \mathbb{G}_2$$

and we call the associated signature π_ρ.

Pairing's magic

Basically we are using the fact that the pairing operation is bilinear to extract the secret key once from the public key and once from the signature to perform a key agreement:

$$\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

$$\mathrm{e}(G_1, \pi) = \mathrm{e}(G_1, sM) = s\,\mathrm{e}(G_1, M)$$
$$\mathrm{e}(P_g, M) = \mathrm{e}(sG_1, M) = s\,\mathrm{e}(G_1, M)$$

Pairing's magic

To ensure secrecy, we need to add the notion of ephemeral key to the mix:

$$P_e = rG_1, r \in \{0,1\}^\ell$$

$$r\,\mathrm{e}(P_g, M) = r\,\mathrm{e}(sG_1, M) = rs\,\mathrm{e}(G_1, M)$$
$$\mathrm{e}(P_e, \pi) = \mathrm{e}(rG_1, sM) = rs\,\mathrm{e}(G_1, M)$$

More maths:

Our timelock

Encryption: A client that wishes to encrypt a message $M \in \{0,1\}^l$ "towards" the epoch ρ will perform the following:

1. Compute $G_{id} = e(P, Q_{id}) = e(P, H_1(\rho))$, “the round public key”
   - This can be pre-computed per epoch, it is the same for everyone
2. Choose a random $\sigma \in \{0,1\}^l$, “the mask”
3. Set $r = H_3(\sigma, M)$ where $H_3 : \{0,1\}^* \to F_q$ is a secure hash function, “the ephemeral secret key”
4. Output the ciphertext:

$$C = \{U, V, W\} =$$

$$\left\{\begin{array}{lr} U = rG_1, & \text{“the ephemeral public key”} \\ V = \sigma \oplus H_2(rG_{id}), & \text{“the mask commitment”} \\ W = M \oplus H_4(\sigma), & \text{“the one-time-pad”} \end{array}\right\}$$

More maths:

Our timelock

Decryption: Given the ciphertext, **and** the associated signature of epoch $\rho : \pi_\rho$, anybody can decrypt the ciphertext in the following way:

1. Compute $\sigma = V \oplus H_2(e(U, \pi_\rho))$
2. Compute $M = W \oplus H_4(\sigma)$
3. Set $r = H_3(\sigma, M)$
4. Test that $U = rG_1$ → if not, reject.
5. M is the decrypted ciphertext

More maths:

Our timelock

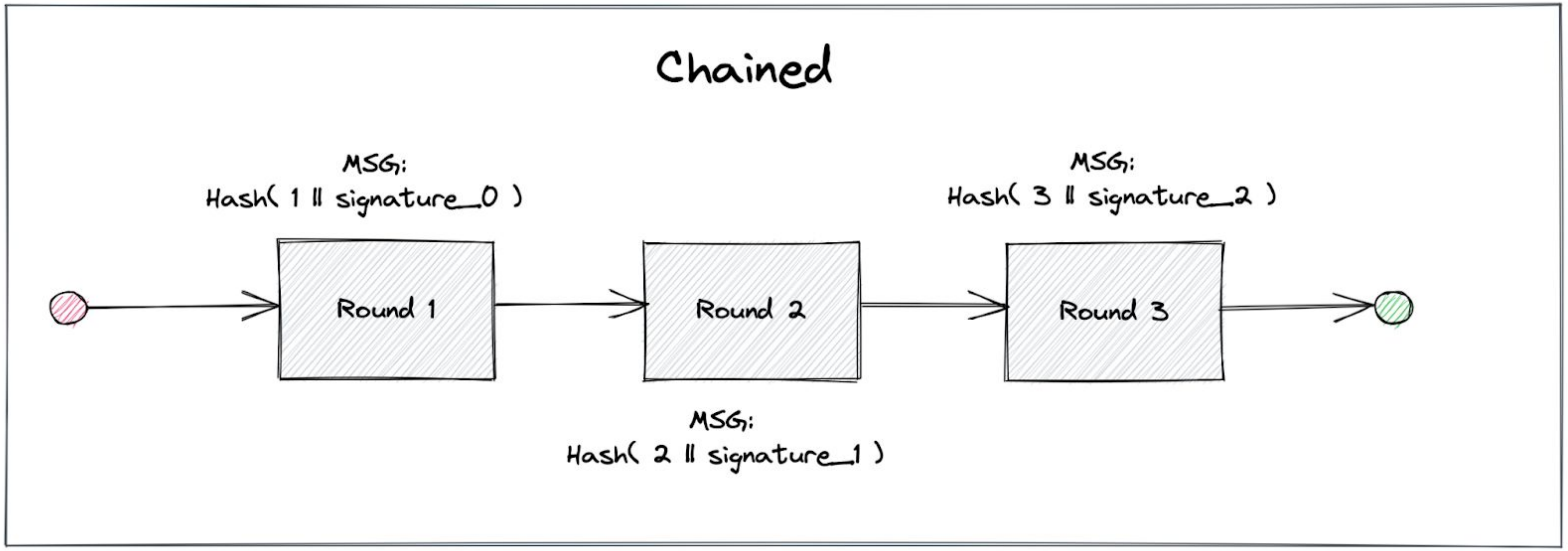
Expect a pre-print in the coming months and a blog post next month going into more details about our scheme, its security, performances, some variations and optimizations.

In the meantime the scheme is publicly available here:

https://protocollabs.notion.site/protocollabs/Timelock-Encryption-drand-f5df65a54a6641dfa77f9b8168c9b90b

Problem: Chained Randomness

Current randomness on the LoE mainnet is **chained**:

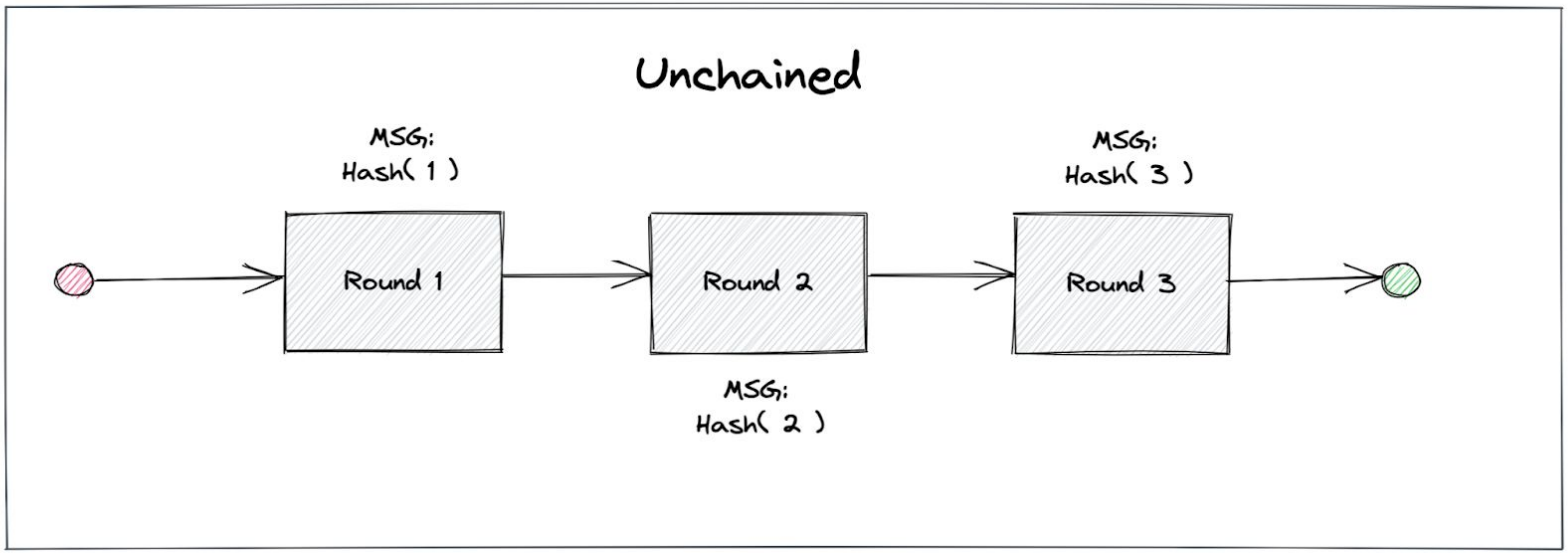


Consequences:

- **No one** knows the round message more than **one round** in advance
  - e.g. Hash(3 || sig_2) can only be known at round 2
- Requires the full chain for proper full verification

Solution: Unchained Randomness

New **unchained randomness** mode introduced in February 2022:



Consequence

- Everybody **knows the future round message** getting signed ahead of time
- Verification is simpler and less stateful, without weakening trust.

Digression: hybrid encryption

We can only encrypt small blocks of data using our timelock encryption scheme, how do we do to encrypt a 10Gb file?

→ using **hybrid encryption**:

we encrypt the data with a random “data encryption key”, and then we only need to time-lock that small data encryption key.

The ~~library~~ libraries

We are glad to open source our work on the topic, by providing two libraries, one in Go and one is JS/TS, which should allow you to start using timelock encryption today in your projects!

- https://github.com/drand/tlock/ (Go)
- https://github.com/drand/tlock-js/ (TS)

Just remember we are relying on the League Of Entropy **testnet** for now, which has a threshold of only 6 versus 13 on mainnet.
Timelock capability is planned on the League's **mainnet** in mid-september.

The CLI tool

We also have create a standalone CLI tool **`tle`** that allows you to encrypt and decrypt data using timelock encryption easily:

```
go install github.com/drand/tlock/cmd/tle@latest
```

Or:

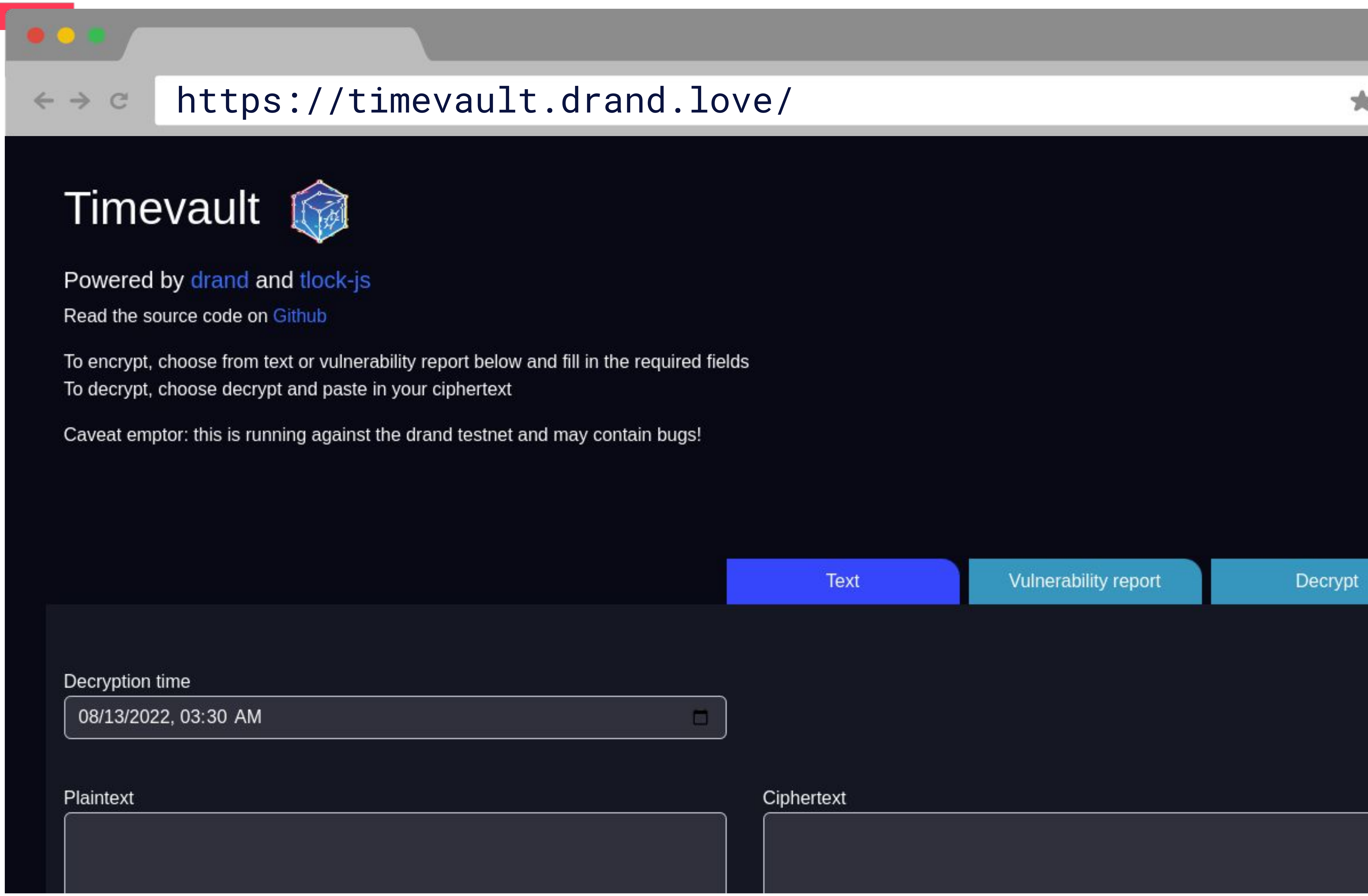
```
git clone https://github.com/drand/tlock
go build cmd/tle/tle.go
```

The CLI tool

```
  tle [--encrypt] (-r round)... [--armor] [-o OUTPUT] [INPUT]
  tle --decrypt [-o OUTPUT] [INPUT]

Options:
  -e, --encrypt  Encrypt the input to the output. Default if omitted.
  -d, --decrypt  Decrypt the input to the output.
  -n, --network  The drand API endpoint to use.
  -c, --chain    The chain to use. Can use either beacon ID name or beacon hash.
                  Use beacon hash in order to ensure public key integrity.
  -r, --round    The specific round to use to encrypt the message.
                  Cannot be used with --duration.
  -D, --duration How long to wait before the message can be decrypted.
                  Defaults to 120d (120 days).
  -o, --output   Write the result to the file at path OUTPUT.
  -a, --armor    Encrypt or Decrypt to a PEM encoded format.
```

Try it live:



timevault.drand.love

Future issues

Long term security is HARD

- New attacks
- Quantum computers
- More computing power

Long term security is HARD

Now, add in the fact that this is a threshold network, it means you need some kind of serious liveness guarantees to encrypt towards the distant future!

Are the nodes still going to run in 10 years? In 20 years?

Long term security is HARD

How about **governance**?

What happens if the League of Entropy members decide to stop the network?

Should they release all the key material to allow to decrypt everything immediately?
Should they destroy key material entirely, making ciphertexts indecipherable?

Credits!

This work is actually a teamwork from the drand team!

- Nicolas Gailly: initial idea & PoC
- Patrick McClurg: JS/TS magic
- Julia Armbrust: web-demo design

And many thanks also to:

- Justin Drake for his insightful comments on the scheme
- Jason Donenfeld for making me realize people loved the idea
- Ardan Labs for collaborating on the Go code

Grow the League!

- Join the League of Entropy! Help secure timelocked content!
- We are looking for partners that can run drand daemon or relay nodes in diverse locations, especially in Asia.
- Infrastructure and operational requirements are minimal:
  Estimated commitment: 2-3 hours/month
  Costs depending on your infrastructure.

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

„We believe the internet has become humanity's most important technology. We build protocols, systems, and tools to improve how it works. ”

–Juan Benet, Protocol Labs

Thank you !

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

https://github.com/drand/tlock
https://github.com/drand/tlock-js
https://drand.love/blog/

Email
yolan@protocol.ai

Twitter
@anomalroil

Details:

References

- http://cypherpunks.venona.com/date/1993/02/msg00129.html
- https://people.csail.mit.edu/rivest/pubs/RSW96.pdf
- https://en.wikipedia.org/wiki/BLS_digital_signature
- https://crypto.stanford.edu/~dabo/papers/bfibe.pdf
- https://en.wikipedia.org/wiki/League_of_entropy
- https://drand.love/blog/2022/02/21/multi-frequency-support-and-timelock-encryption-capabilities/
  (and its FAQ:
  https://docs.google.com/document/d/16QJG3Z-Kr0mN6snQz8cm0NnMXpYBpelKyvCf2oo1Zgc/edit?usp=sharing)

- Upcoming pre-print on eprint, with all the maths and crypto, stay tuned!