tlock: Practical timelock encryption based on threshold BLS

Nicolas Gailly, Kelsey Melissaris, **Yolan Romailler**

Protocol Labs

What we want

Encrypt something to the future

Not new

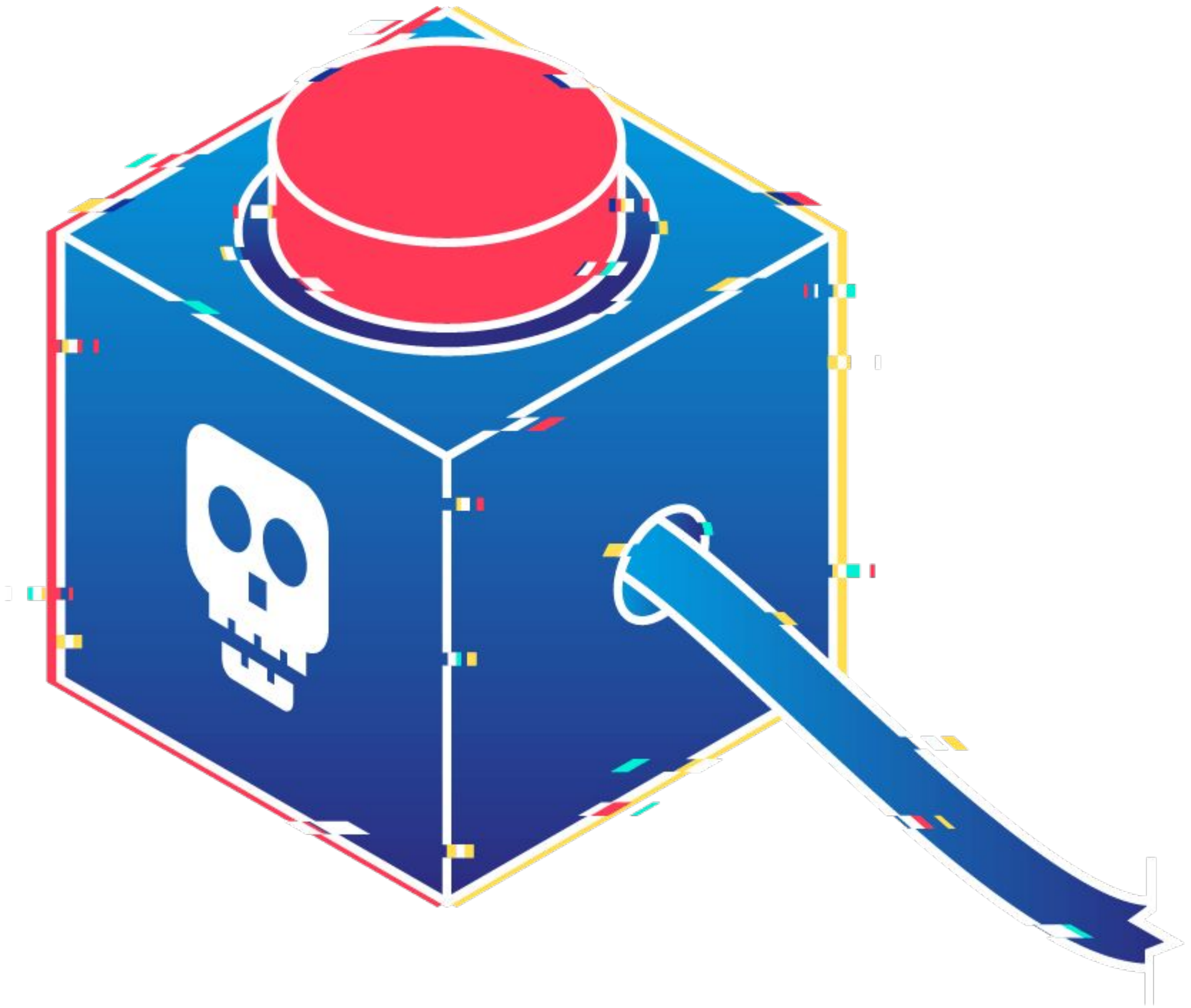
Timelock or Timed-Release Encryption

- Tim May in 1993 on the Cypherpunks mailing list, using trusted third party.
- “Time-lock Puzzles” in 1996 by Rivest, Shamir and Wagner, using PoW.
- “The HP Time Vault Service” in 2002 by HP, using an IBE approach.
- First paper about BLS-based timelock in 2004 by Blake & Chan.
- “Time-lapse cryptography” in 2006 by Rabin and Thorpe, using DKG, verifiable secret sharing and ElGamal encryption because:

  *The notion of “sending a secret message to the future” has been around for over a decade. **Despite this, no solution to this problem is in common use***

Timelock

Applications

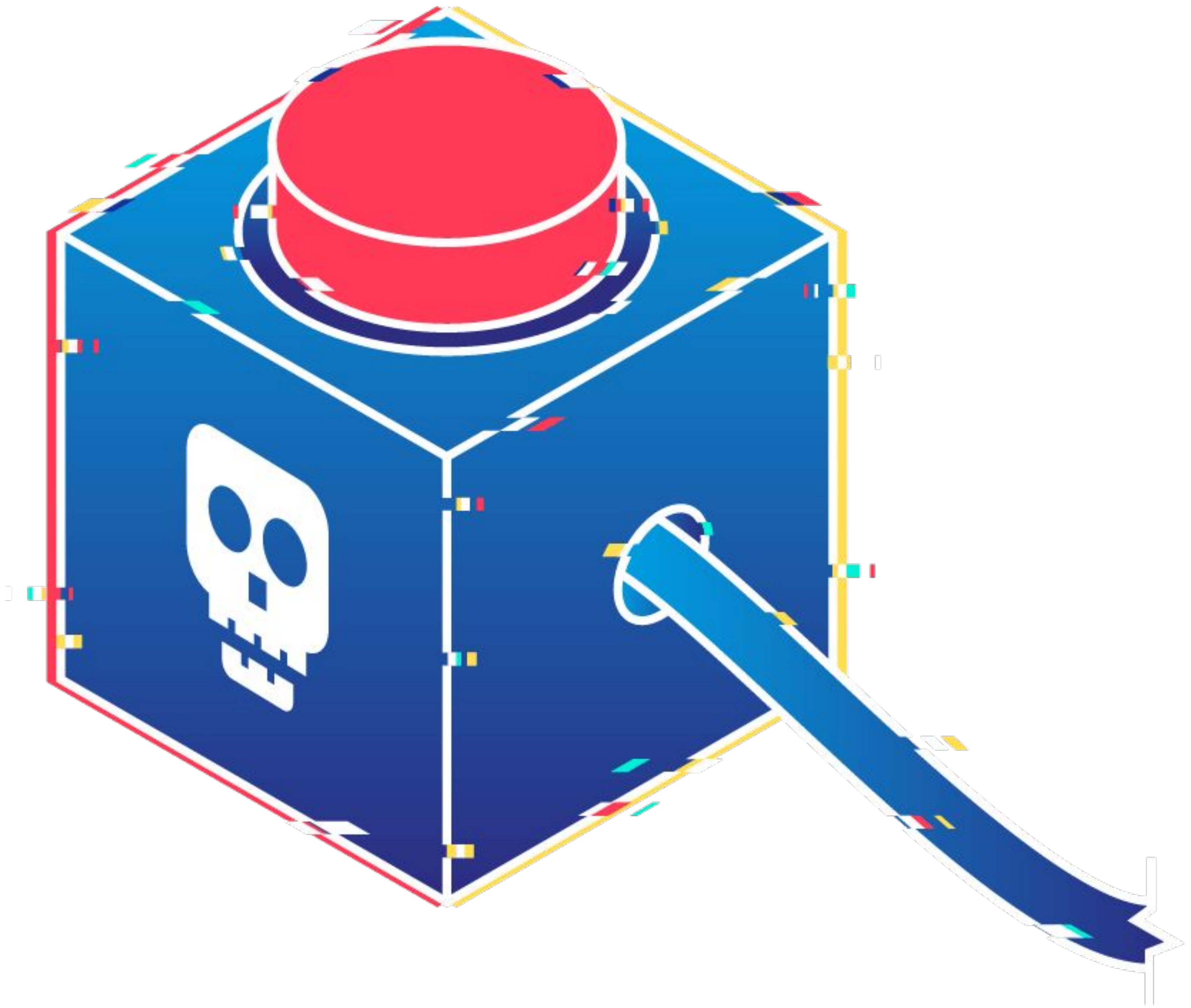


- Bids in **sealed-bid auctions**
- Could help with Electronic Voting
- Can help with **MEV and frontrunning issues**
- Key escrow: “**a dead-man switch for your BTC**”
- Issue documents with a **known embargo period**
- **Responsible Ransomwares**: “Pay now to get the decryption key, or wait.”
- Escaping emulation: “Wait until time X has lapsed to decrypt the payload.”

The well-known risk with responsible disclosure

Timelock

Applications



- Bids in **sealed-bid auctions**
- Could help with Electronic Voting
- Can help with **MEV and frontrunning issues**
- Key escrow: “**a dead-man switch for your BTC**”
- Issue documents with a **known embargo period**
- **Responsible Ransomwares**: “Pay now to get the decryption key, or wait.”
- Escaping emulation: “Wait until time X has lapsed to decrypt the payload.”

What we have
drand

- drand is an **open source** software in Go ran by a set of independent nodes that collectively produce beacons.
- Provides **public, verifiable random beacons** using
    - **Threshold BLS** on the curve **BLS12-381**
    - Pedersen Distributed Key Generation and resharings
- Tested, **audited**, and deployed at scale by the League of Entropy since 2019. Used in production since 2020.

What we have

The League of Entropy

18 organizations

3M+ rounds

21 nodes

3+ years

5 endpoints

0 disruptions since launch in 2020-08-10

0 single point of failure

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

What we have

drand beacons map to a precise time!

What we have

How to?

A **footnote** in the original IBE paper in 2001 mentions that identity decryption keys can be used as *signatures*, BLS does that.
=> **BLS signatures can be seen as decryption keys** for a specific identity.

We have a live production network issuing random beacons signed using **threshold BLS signatures** at a fixed frequency with DKG and all.

Intuition

BLS reminder

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:

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

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

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

Intuition

Use it for encryption

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

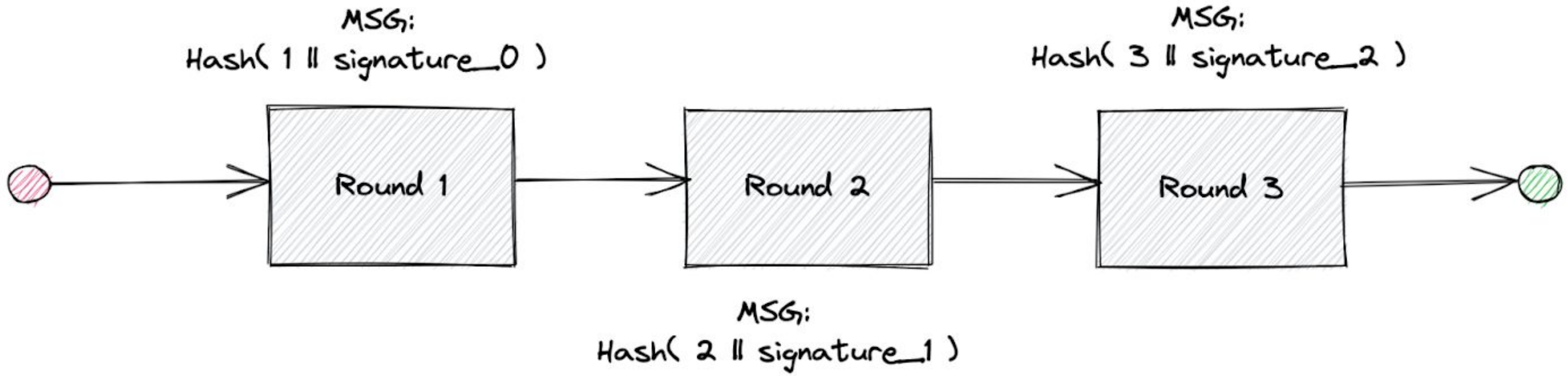
$$P_e = rG_1, r \in \{0,1\}^\ell$$

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

In practice

Problem: chained randomness

The beacons on the LoE `default` mainnet are Chained



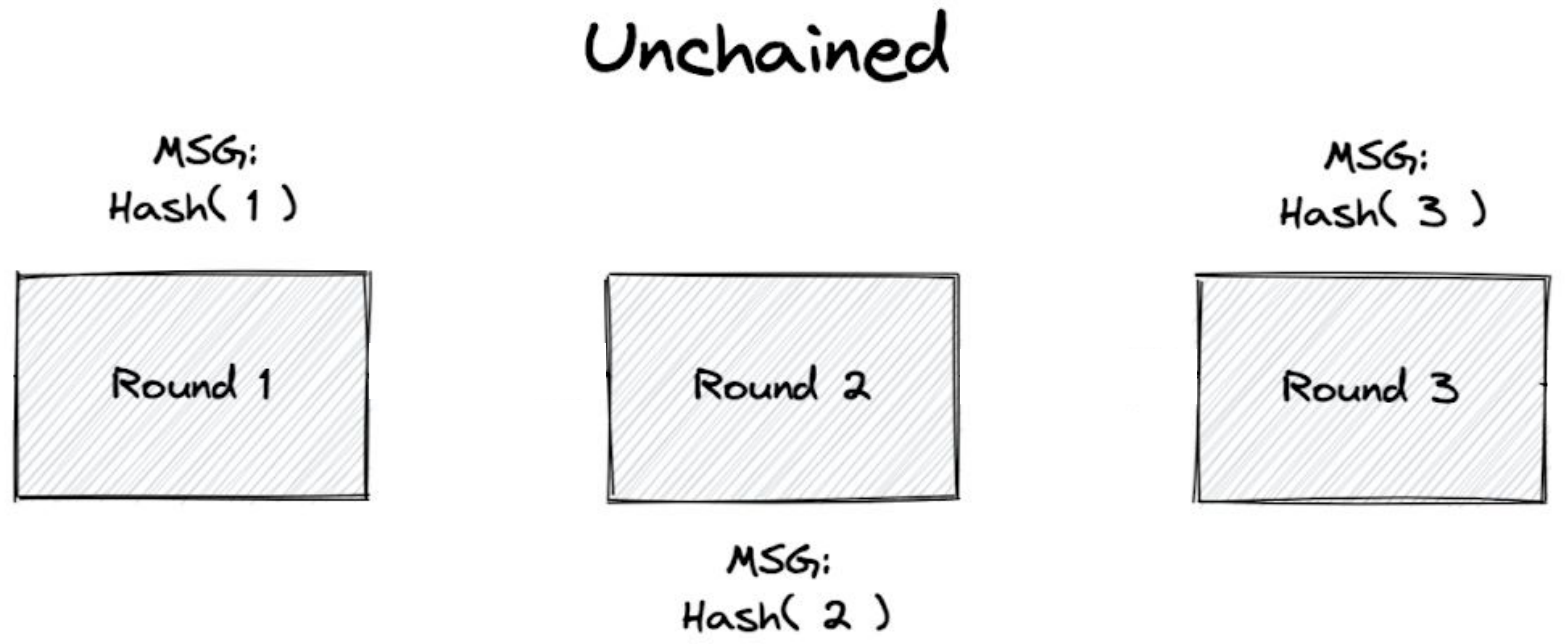
Consequences:

- **No one** knows the round message more than **one round** in advance
    - e.g. Hash(3 || signature_2) can only be known at round 2
- Requires the full chain for proper full verification
- Not compatible with IBE-based Timelock

In practice

Solution: Unchained Randomness

New **unchained randomness** mode introduced in February 2022, launched on Testnet in May and achieved general availability on Mainnet on March 1st, 2023!



Consequences:

- Messages are mapped to a given time: Hash(10) happens at time T_10
- Everybody **knows the future round message** getting signed ahead of time.
- Verification is much simpler and stateless, without impacting trust/security.

In practice

Problem: performance/size on-chain

BLS signatures on BLS12-381 done on $\mathbb{G}_2$ are ~96 bytes in compressed form.

Furthermore we need to map the message M to the group $\mathbb{G}_2$, which is at least 10x more costly than doing so on $\mathbb{G}_1$.

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

BLS signature

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

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

BLS public key

In practice

Solution: swap G1 and G2

New **swapped group scheme** launched in February 2023.

$$e(\pi, G_2) = e(\underbrace{\boxed{sM}}_{\text{BLS signature}}, G_2) = s\, e(M, G_2)$$

$$e(M, P) = e(M, \underbrace{\boxed{sG_2}}_{\text{BLS public key}}) = s\, e(M, G_2)$$

Storage benefit: signatures are now 50% smaller at 48 bytes vs 96 bytes!

In practice

Digression: hybrid encryption

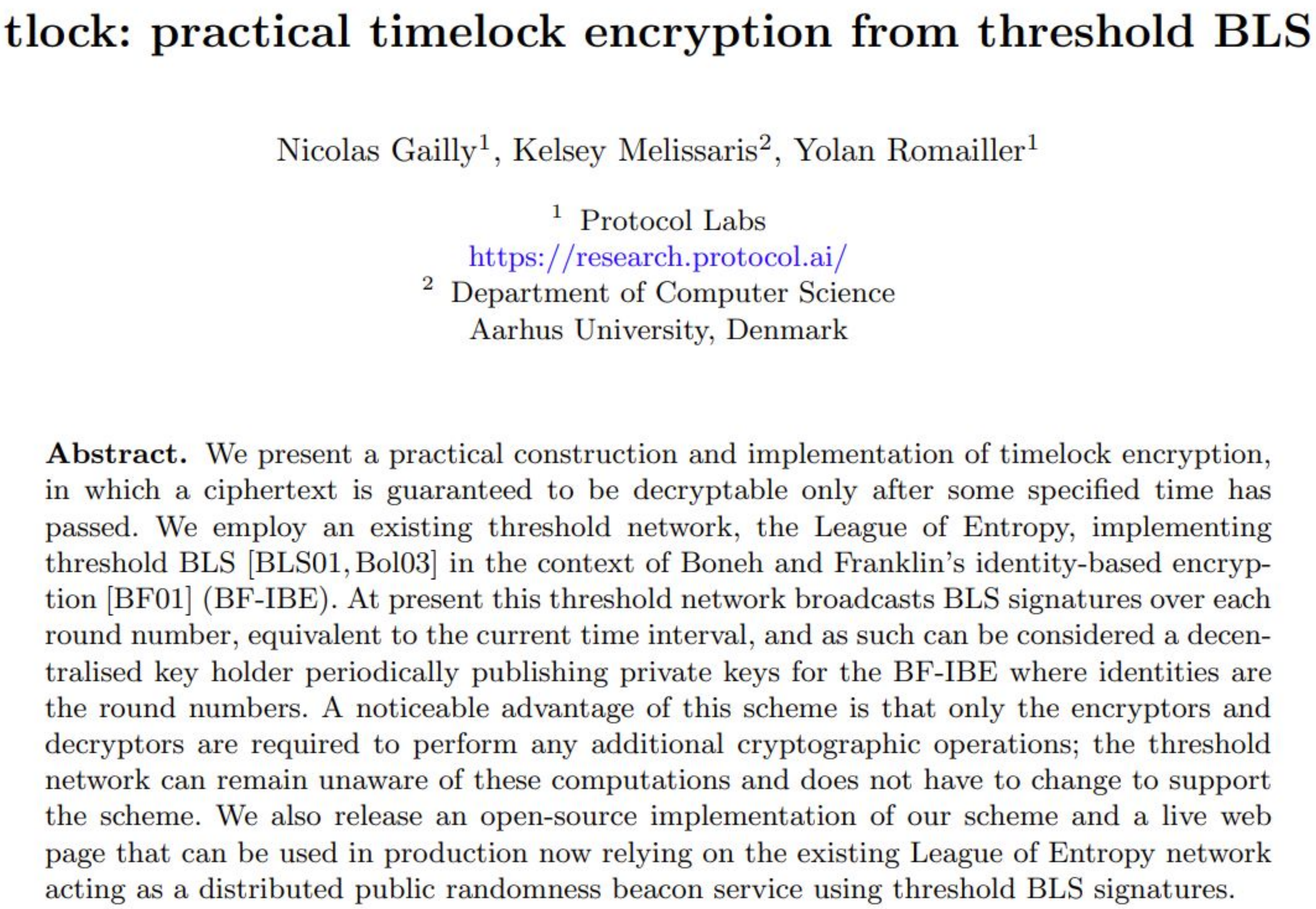
We can only encrypt small blocks of data using IBE/our timelock scheme, since we opted for using a hash for key derivation rather than a XOF... so we need to use **hybrid encryption**.

For ease, we used age to achieve this using a custom stanza for timelock and delegating key-wrapping and data encryption to it. In theory in a way compatible with its new plugin system:

```
age-encryption.org/v1
-> tlock 764081 dbd506d6ef76e5f386f41c651dcb808c5bcbd75471cc4eafa3f4df7ad4e4c493
```

Details

It’s almost all on ePrint


tlock: practical timelock encryption from threshold BLS

Nicolas Gailly[1], Kelsey Melissaris[2], Yolan Romailler[1]

[1] Protocol Labs
https://research.protocol.ai/
[2] Department of Computer Science
Aarhus University, Denmark

Abstract. We present a practical construction and implementation of timelock encryption, in which a ciphertext is guaranteed to be decryptable only after some specified time has passed. We employ an existing threshold network, the League of Entropy, implementing threshold BLS [BLS01, Bol03] in the context of Boneh and Franklin’s identity-based encryption [BF01] (BF-IBE). At present this threshold network broadcasts BLS signatures over each round number, equivalent to the current time interval, and as such can be considered a decentralised key holder periodically publishing private keys for the BF-IBE where identities are the round numbers. A noticeable advantage of this scheme is that only the encryptors and decryptors are required to perform any additional cryptographic operations; the threshold network can remain unaware of these computations and does not have to change to support the scheme. We also release an open-source implementation of our scheme and a live web page that can be used in production now relying on the existing League of Entropy network acting as a distributed public randomness beacon service using threshold BLS signatures.

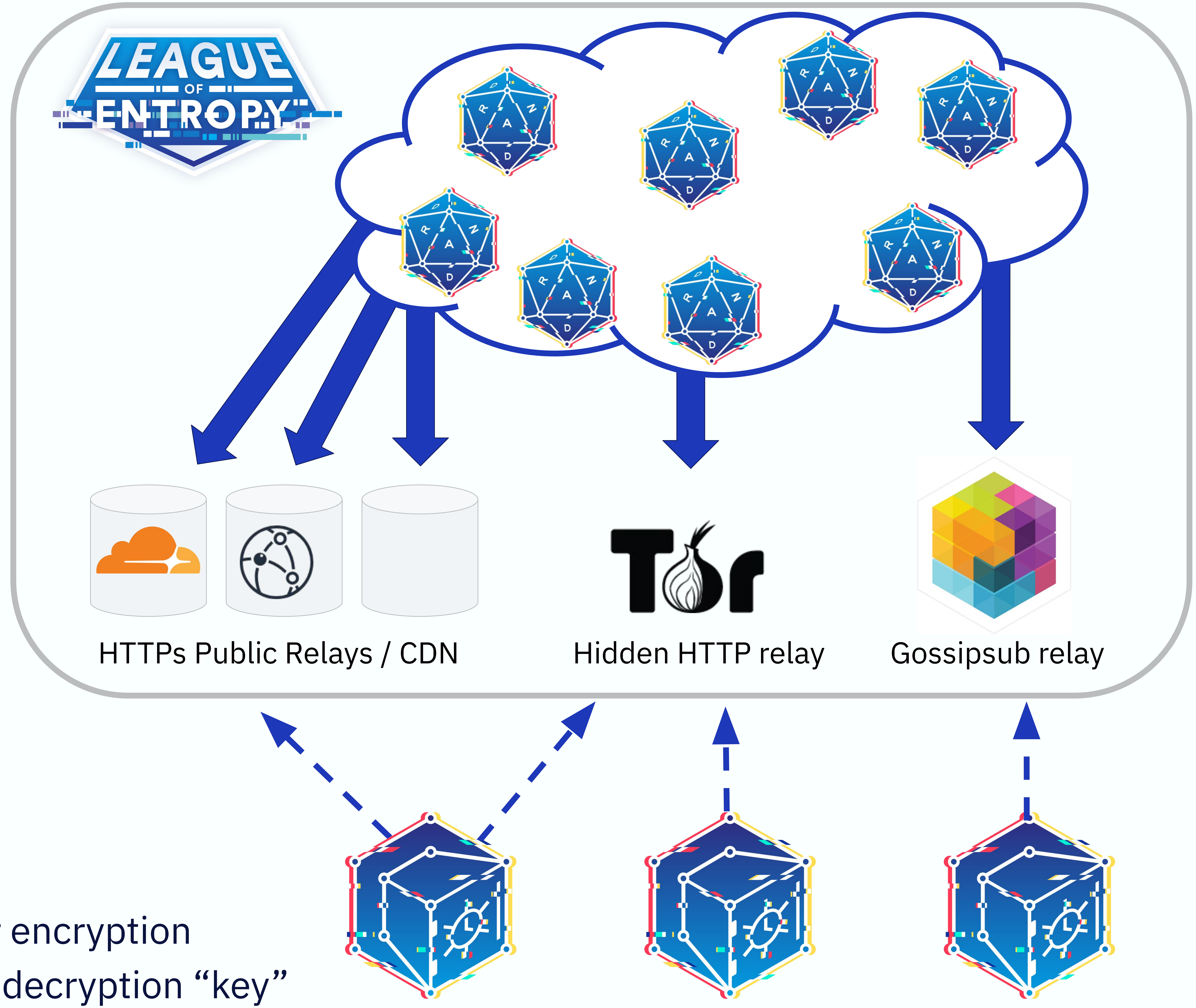
This work is explained in more detail in our ePrint paper, and we are looking into UC security proofs and extending it a bit more, so don’t hesitate to check it out:
https://ia.cr/2023/189

So, let’s look at the “Real World” part of it that’s not on ePrint! What does “practical” mean and why are we here today?

In practice
Our timelock

The League of Entropy part

- Permissioned network
- Threshold $t > (n / 2) + 1$
- **100% uptime** since mainnet launch in 2020
- **Stable group public key**
- Granularity of 3s
- Solid Distribution Network
- Is not dedicated for timelock



The Timelock part

- Client-side only operations
- Needs the group public key for encryption
- Queries the drand network for decryption “key”

In practice

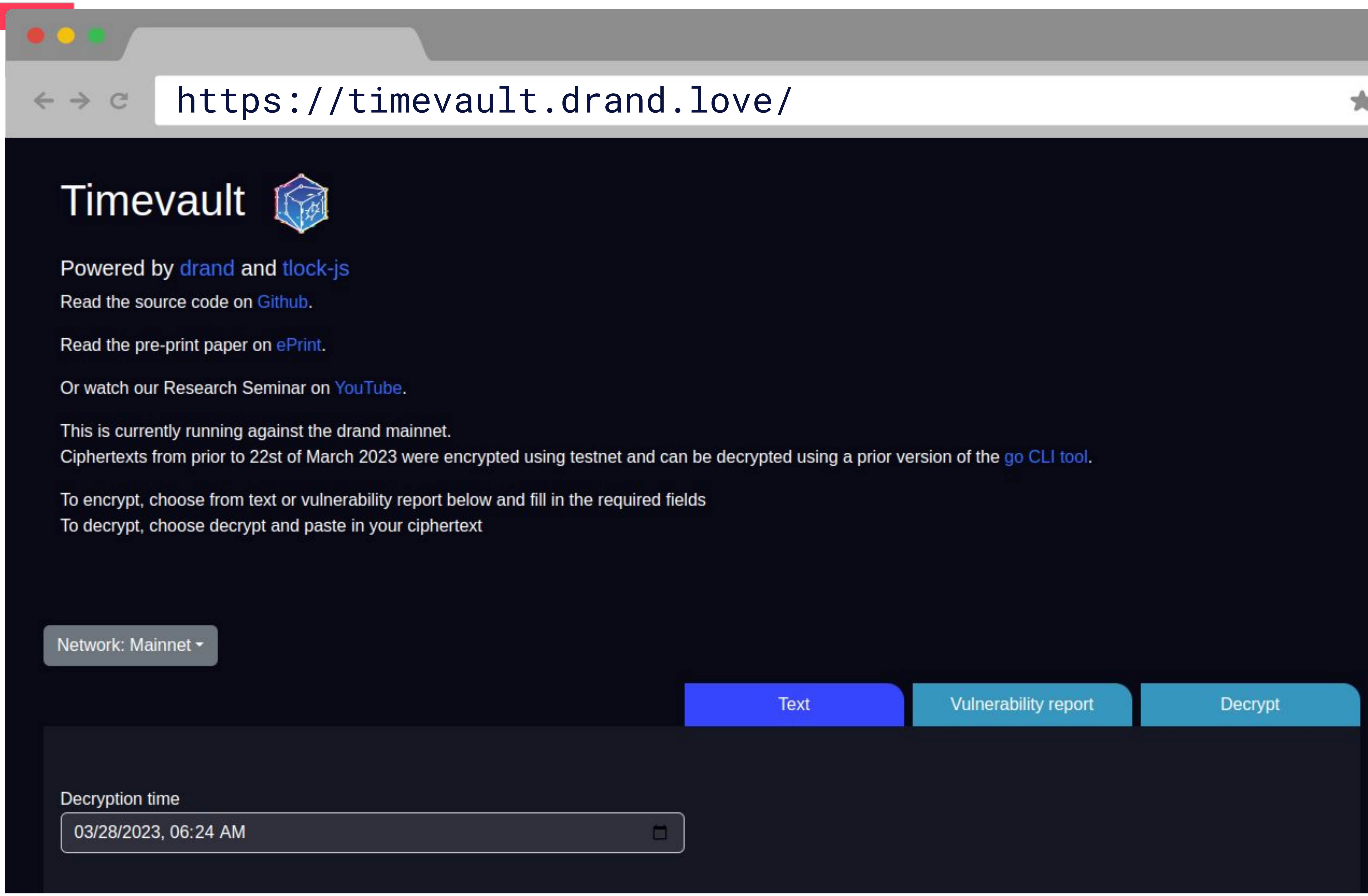
The ~~library~~ libraries

We have open sourced our work, providing two libraries, one in Go and one is JS/TS. Start using timelock encryption today in your projects!

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

And we already have a third party Rust implementation that’s interoperable with ours: https://github.com/thibmeu/tlock-rs

Try it live:



timevault.drand.love

What's next?

Future work

- Implement use cases! (Sealed bid auctions, MEV prevention, etc.)
- Not much research into "threshold post-quantum signatures"!
- Also not too much for "PQ-IBE" schemes.
- Most ecosystems don't have BLS12-381 built-in functions, need for a spec.
- Some implementations do not yet support signatures on $\mathbb{G}_1$.
- Look into doing ZKPs on the timelocked input.
- The League of Entropy is welcoming new members!

Tomorrow we are hosting the Randomness Summit 2023, in case you want to get in touch or learn more about it randomness beacons, VRFs, and more!

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/

yolan@protocol.ai
@anomalroil

Protocol Labs

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
```

And there's also obviously already an alternative 3rd party Rust CLI:

```
https://github.com/thibmeu/drand-rs
```

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.
```