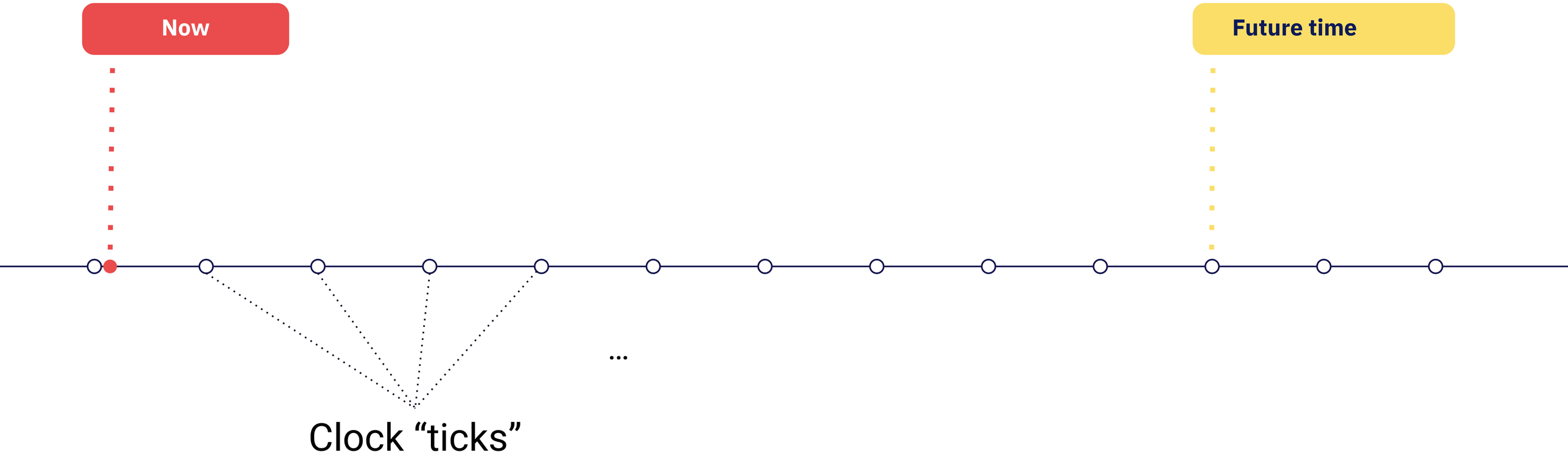
tlock: encrypting messages to the *future* in Go

Yolan Romailler

Protocol Labs

What we want

Encrypt something to the future



Not new

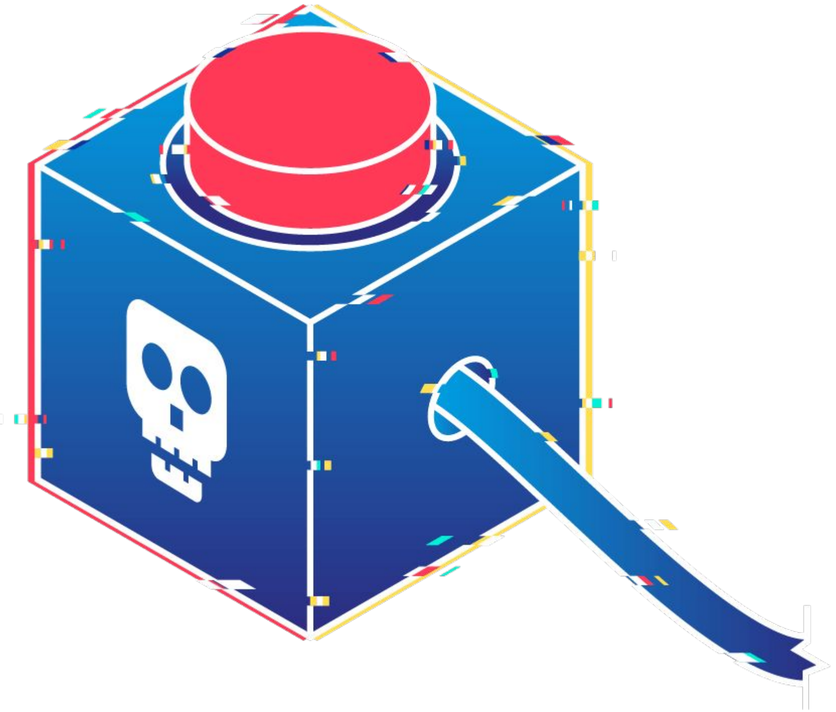
Timelock or Timed-Release Encryption

- 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

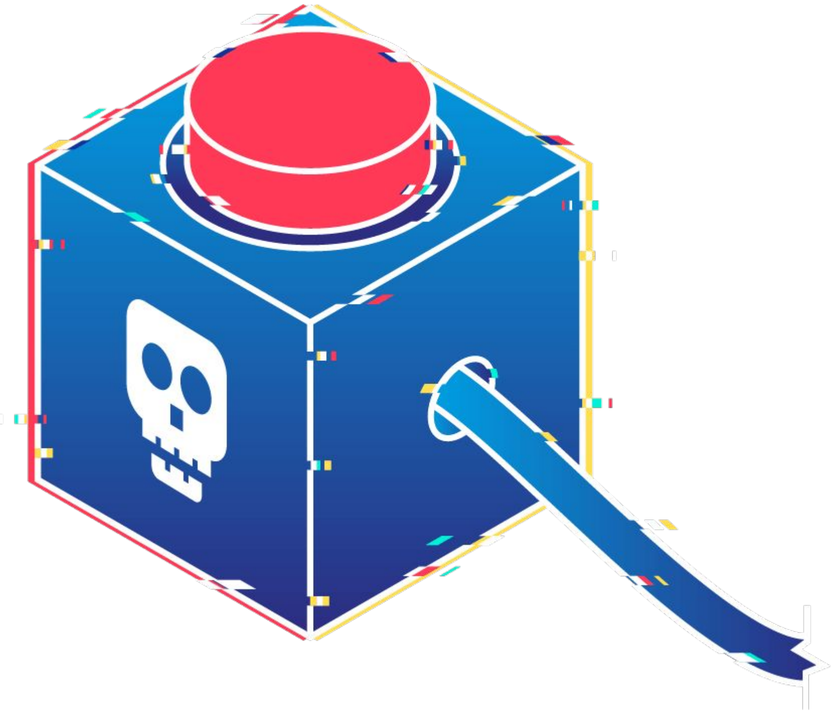


- Bids in **sealed-bid auctions**
- Could help with Electronic Voting
- Can help with **MEV and frontrunning issues**
- Key escrow: “**a dead-man switch for your BTC**”
- Issue documents with a **known embargo period**
- **Responsible Ransomwares**: “Pay now to get the decryption key, or wait.”
- Escaping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

Left as an exercise:

How does it work?

Super easily (https://ia.cr/2023/189):

$$\phi = V \oplus H_2(\hat{e}(U, \sigma))$$

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

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

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

By definition of bilinear pairings, we get

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

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

$$= \phi$$

In practice

Digression: hybrid encryption

We can only encrypt small blocks of data using our timelock scheme… so we need to use **hybrid encryption**: encrypt the data with a “data encryption key” using a (fast) symmetric algorithm, encrypt the “data encryption key using our timelock scheme.

For ease, we used age to achieve this using a custom stanza for timelock. In theory in a way compatible with its new plugin system:

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

In practice

Digression: age

age: “A simple, modern and secure encryption tool (and Go library)”

```
import (
  "filippo.io/age"
)
// generate private and public keys
bob, err := age.GenerateX25519Identity()
bobPublicKey := bob.Recipient()
[...]
```

In practice

Digression: age

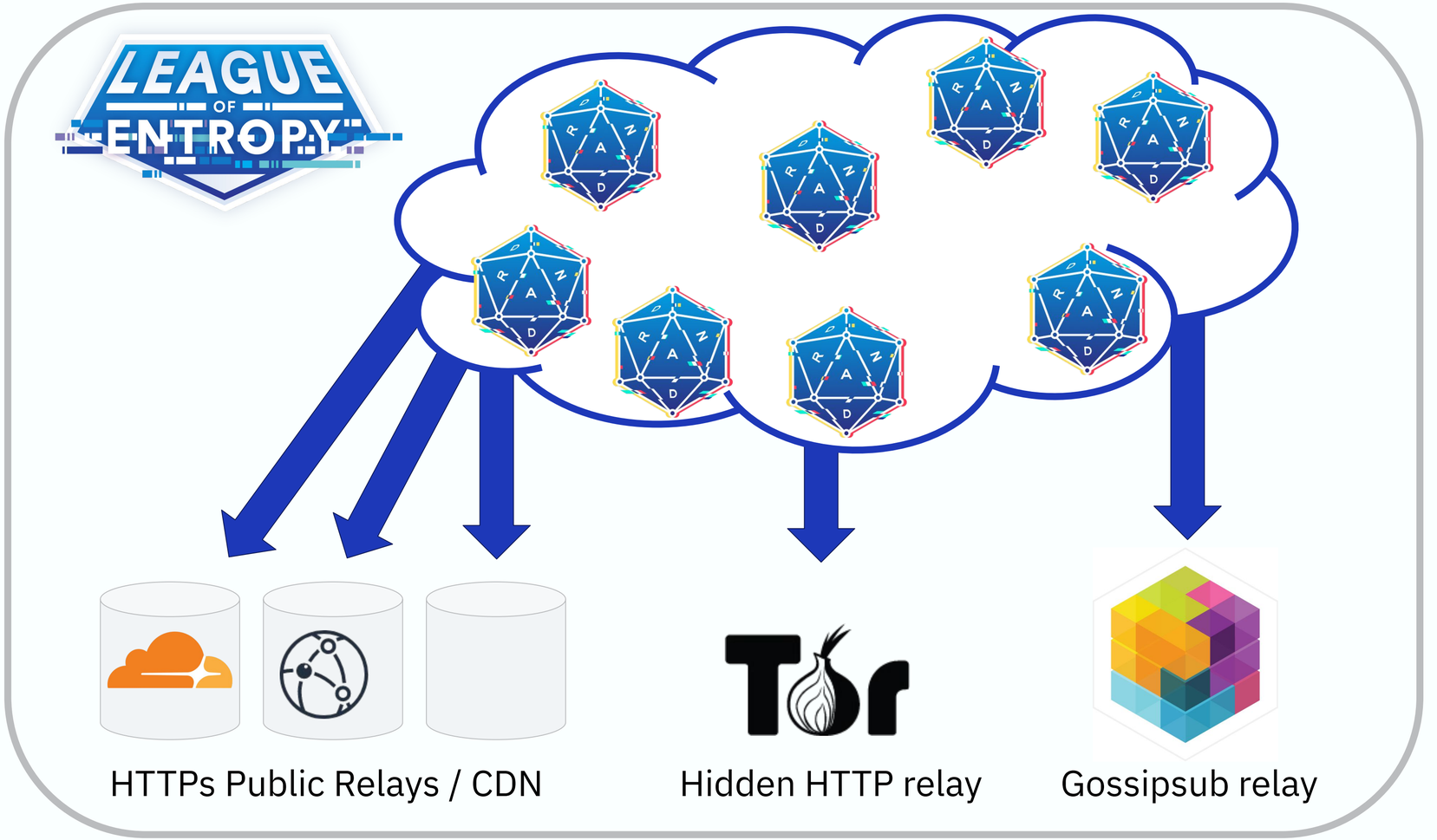
```
var ciphertext bytes.Buffer
encryptor, err := age.Encrypt(&ciphertext, bobPublicKey)
encryptor.Write([]byte("Hello GopherconEU!"))
encryptor.Close()
fmt.Println(ciphertext.String())
```

```
age-encryption.org/v1
-> X25519 ht++CvfgN6sW7azrSEyD0I7Avb1G1DBSfqqU33aixio
5dL+AseiDxvANIdTRQ3Ai/OLxxcbmKlLPoHHiNi6omQ
--- JLBpp45/31ef/nxVaW0bssN+gZxyptsTnndmujZsbqA
K��-H7��uʃ�f□�"ʒtG�����$���!��#�2H��R���
```

In practice
Our timelock

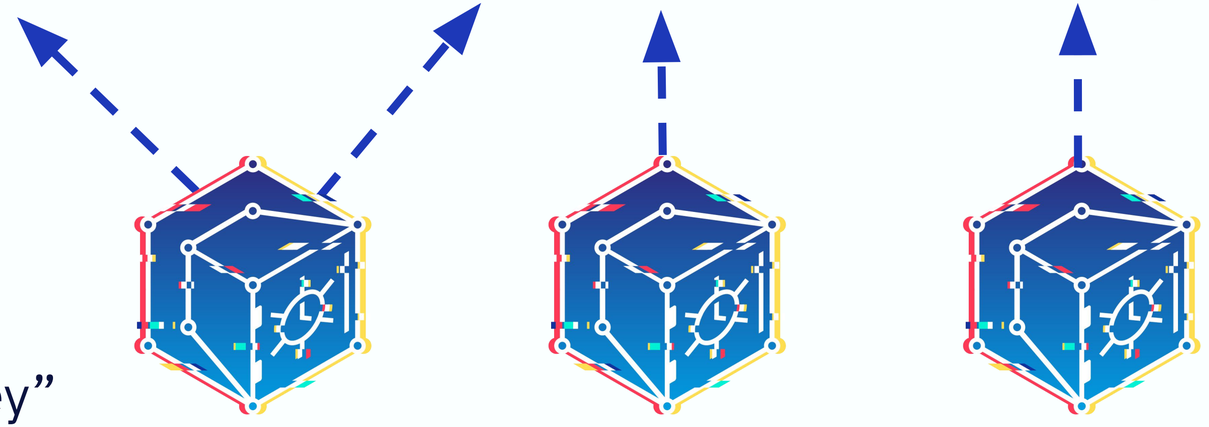
The League of Entropy part

- 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 to 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 in collab with Ardan Labs 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

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
cd tlock
go build cmd/tle/tle.go
```

The library

```
import (
  "github.com/drand/tlock"
  "github.com/drand/tlock/networks/http"
)
[...]
host := "https://api.drand.sh/"
chainHash := "dbd506d6ef76e5f386f41c651dcb808c5bcbd75471cc4eafa3f4df7ad4e4c493"
network, err := http.NewNetwork(host, chainHash)
[...]
err = tlock.New(network).Encrypt(ciphertext, data, roundNumber)
[...]
err = tlock.New(network).Decrypt(recovered, ciphertext)
```

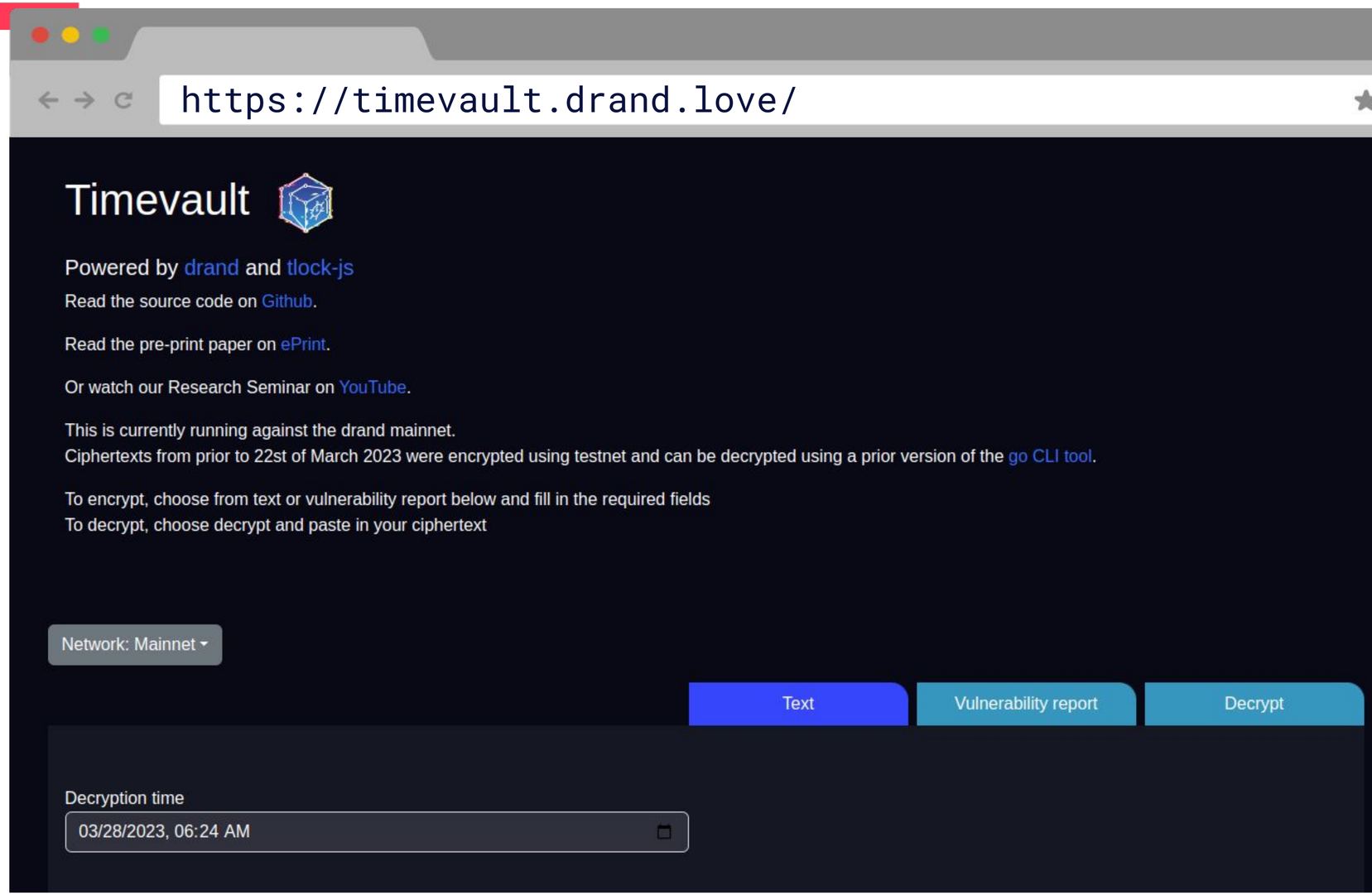
Playground link.

The library

And you get a ciphertext nobody can decrypt until 15h55 today!

```
-----BEGIN AGE ENCRYPTED FILE-----
YWdlLWVuY3J5cHRpb24ub3JnL3YxCi0+IHRsb2NrIDM0NTM5MDEgZGJkNTA2ZDZl
Zjc2ZTVmMzg2ZjQxYzY1MWRjYjgwOGM1YmNiZDc1NDcxY2M0ZWFmYTNmNGRmN2Fk
NGU0YzQ5MwpnNDZkbVNvYkVseDRBQXdBczE0bTJwbUdRVU93M20vMkU2UTBhVDg2
MXp4Qm1oTUhEMFpKTHREcm1kVEhGTjdXCkNZSWxDZ2hVWjk5TzBzNHRwU3NIYVpm
VEd0VXNpN0M4ZVlhY2NiZXRrZXU1OExSUmt6cEp5QlBSMXdtSlRkWmUKbGxVMTVG
ZU9tU0ZoTUQvVVdMVUNkRlExL2VrT1FLMkt1TmhSeW1KREEySQotLS0gYjh0c3Zi
clYycHRnUDlJTkVGcnUxUmoyS0VrMm5vRjBCVXU5alozSWk1bwoq8IbieaktDUMO
b/lgMdMeDkTG3N1+TT66atjA6UUdXGjnMMfVOLXeBJdg0kwgRPHFmSqsgQKXRDUE
HQvwlonWN4+PAjE0qQRwIDl6gc54Ng+0gUcXbMR3/tVOfmmjSFo=
-----END AGE ENCRYPTED FILE-----
```

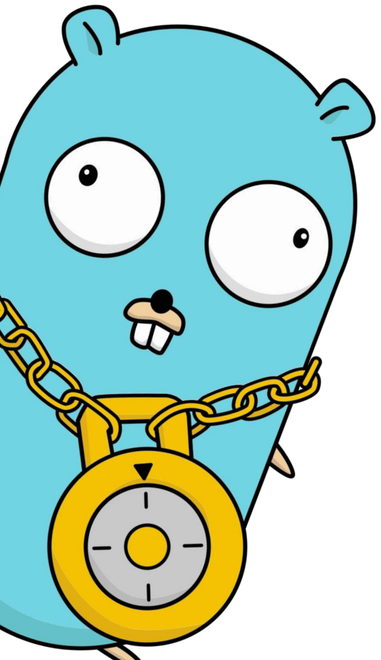
Try it live:



timevault.drand.love

Thank you !

For more information, go to:
https://github.com/drand/tlock



yolan@protocol.ai
@anomalroil

Protocol Labs