Building `dee`, an interroperable timelock client

Thibault Meunier

Game time

Fun, in a way

Guess the command - 1

```
____ remote add https://github.com
```

Guess the command - 1

```
git remote add https://github.com
```

Guess the command - 2

```
____ -X POST https://rwc.iarc.com/2024
```

Guess the command - 2

```
curl -X POST https://rwc.iarc.com/2024
```

Guess the command - 3

```
____ --expert --ec
```

Guess the command - 3

```
gpg --expert --ec
```

Guess the command - 4

```
____ <SOURCE...> <DESTINATION>
```

Guess the command - 4

```
cp <SOURCE...> <DESTINATION>
```

Guess the command - 5

```
____ keygen
```

Guess the command - 5

```
ssh keygen
```

In the next 15min

0. Pre-requisites
1. Demo
2. CLI design
3. Timelock API
4. Final words

Pre-requisites

Useful context

The League of Entropy

Verifiable randomness every 3s

```
93be67a8e0585f9e057888de0a2f6f2841f3bd76634e8c47209c16f108322067
```

Threshold Group Signature over `H(<ROUND>)`

Timelock

tlock paper by Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler

IBE scheme over the League of Entropy

Implemented by drand team in Go and Javascript

Interroperable across usages

- Web UI for text https://timevault.drand.love/
- Web UI for files https://dee.notshady.com
- Web API https://tlock-worker.crypto-team.workers.dev
- CLI `dee`

Demo

Try it at home

Live demo

Installation

```
cargo install dee
```

Add a remote chain

```
dee remote add quicknet https://drand.cloudflare.com/dbd506d...
quicknet
```

Live demo - 2

Retrieve public randomness

```
dee rand -u quicknet
3129db460507ff559f7fa5e71d6f8bc66aec27516de3d78f7461f6299a2bd483
```

Encrypt 30 seconds to the future

```
echo "Hello dee!" | dee crypt -r 30s > locked.dee
```

Decrypt, the future is now

```
dee crypt --decrypt locked.dee
Hello dee!
```

Designing a CLI

CLI experience is real

Limit default

No default network

```
dee remote add mainnet https://api.drand.sh
```

Choose your own

```
dee rand --set-upstream mainnet
```

Communication for everyone

Configurable output level

```
dee rand -l
Round     : 2820083
Relative  : 00:00:24 ago
Absolute  : 2023-03-28 19:58:30
Randomness: 66aba01bb54f200ef6363143615e1e193eaacbb89dcc7b38...
Signature : 82fb1e24bd603216241d75d51c3378b193d62e4fb8fdbeab...
```

Informative error

```
echo "Hello world!" | dee crypt -r 30s
error: remote must use unchained signatures
```

Mimic existing CLIs

git inspired

```
dee remote show mainnet
```

age inspired

```
dee crypt --decrypt --armor < cat.png
```

drand inspired

```
dee rand -u mainnet --json 1000
```

Rust specific devtooling

clap all in one argument parser, documentation, and manpages generation

```
/// Set default upstream. If empty, use the latest upstream.
#[arg(short = 'u', long, value_hint = ValueHint::Url)]
set_upstream: Option<String>,
```

Cross-platform support is simpler without openssl

```
cargo build --target wasm32-wasi
```

Considered two BLS12-381 libraries: zkcrypto/bls12_381 and arkworks-rs/curves.

```
cargo bench --all-features
```

My laptop "only" has 8GB of RAM

Timelock API

Encrypting towards the future doesn't negate API considerations

Work offline

Go

```go
func (t Tlock) Encrypt(
  dst io.Writer, src io.Reader, roundNumber uint64
) (err error) {
```

Rust

```rust
fn encrypt(
  dst: Write, mut src: Read, roundNumber: u64,
  hash: &[u8], pk: &[u8],
) -> Result<()> {
```

Work offline

Go

```
network := "https://api.drand.sh"
tlock := tlock.New(network)
tlock.Encrypt(dst, src, roundNumber)
```

Rust

```
let client: HttpClient = "https://api.drand.sh".try_into()?;
let info = client.chain_info()?;

tlock_age::encrypt(
    &mut dst,
    src,
    &info.hash(),
    &info.public_key(),
    roundNumber,
)?;
```

Interroperability

Two existing implementations: drand/tlock (Go), drand/tlock-js (JavaScript).

rage (Rust implementation of age) adds a grease stanza: `<rand>-grease <rand>`.

RFC 9380 Hash to curve is a beacon of light: hash_to_field, expand_message.

Elliptic curve serialisation is not standardised.

$$\begin{array}{cc} \mathbb{F}_{p^{12}} \to c_0 \| c_1 & \mathbb{F}_{p^{12}} \to c_1 \| c_0 \\ c_0 \to \text{big-endian} & c_0 \to \text{little-endian} \end{array}$$

Final words

Time to move on

What could be different

Hostname instead of chain hash

```
https://api.drand.sh/dbd506d6ef76e5f386f41c651dcb808c5bcbd75471cc...
-> https://quicknet.api.drand.sh
```

Stateless CLI

```
dee remote
-> dee rand -u https://api.drand.sh/<hash>
-> DEE_REMOTE=https://api.drand.sh/<hash>
```

Age Plugin

```
tlock {round} {chain_hash}
-> tlock REDACTED REDACTED
```

Takeaways

1. One academic paper, multiple engineering tradeoffs.
2. Building a protocol on top of an existing one changes the API.
3. CLI engineering is a thing - clig.dev
4. Discussions improve software. Thanks to everyone that answered questions.

Thank you

For more information, go to:

github.com/thibmeu/drand-rs

github.com/thibmeu/tlock-rs