Building `dee`, a simple timelock client

Thibault Meunier

In the next 15min

1. Demo
2. CLI design
3. Timelock API
4. Final words

Demo

Try it at home

Live demo

Installation

```
cargo install dee
```

Add a remote chain

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

Live demo - 2

Retrieve public randomness

```
dee rand -u fastnet
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
```

Timelock API

Encrypting towards the future doesn't negate API considerations

Work offline

Go

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

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 chain = Chain::new("https://api.drand.sh");
let client = HttpChainClient::new(chain, None);
let info = client.chain().info().await?;

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>`.

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

Elliptic curve serialisation is not standardised.

$$\mathbb{F}_{p^{12}} \to c_0 \| c_1 \qquad \mathbb{F}_{p^{12}} \to c_1 \| c_0$$

$$c_0 \to \text{big-endian} \qquad c_0 \to \text{little-endian}$$

Final words

Time to move on

What could be different

Hostname instead of chain hash

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

Stanza format

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

Stateless CLI

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

Takeaways

1. A new drand and tlock implementation.
2. One academic paper, multiple engineering tradeoffs.
3. tlock is not be constrained to existing drand API.
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