CoBox

MVP TRL 4-5

An implementation of a peer-to-peer
co-operative cloud infrastructure
on the hypercore protocol

Kieran Gibb
Greg Jones
Daniel Hassan

for CoBox - a Magma Collective project

twitter: @CoBoxCoop
url: https://cobox.cloud

This project has received funding from the European Union's Horizon 2020 research and innovation programme within the framework of the LEDGER Project funded under grant agreement No.825268. This report reflects only the authors' views, and neither the Contractor nor the EU is responsible for any use that may be made of the information it contains.

Table of Contents

Abstract

Our technical MVP addresses the data storage, accessibility, hosting and archiving needs of the co-operative sector by proposing a distributed file storage and replication mechanism. Our approach aims to engender treating data as a common good owned by citizens. To do so, we have leveraged the benefits of self-hosting combined with recent innovation in peer-to-peer technologies to share the responsibilities of maintaining a shared cloud infrastructure among multiple nodes distributed across a network. CoBox's stack provides data availability and resilience, allowing organisations and individuals to manage their own sovereign backup infrastructure.

A demo video of the software in action using the prototype command-line interface can be found on our website.[1]

Methodology

In our Architecture Schematics Overview[2], we outlined the challenges of on-boarding peers into P2P systems and the subsequent accessibility issues entailed were highlighted. The other primary design consideration for CoBox was the Ledger programme's focus on Human Centric technologies and businesses.

> "Human Centric puts purpose before profit, puts people before profit, that's what we mean with it Human Centric means that what we are developing first and foremost useful for the society and then it is sustainable, of course, you can make some money out of it, you can sustain and grow your organisation but it has to benefit the society around you, the people I will say even the living beings, plants and animals included."[3]

As well as building on our previous research on the *Usability of Peer to Peer Tools*[4] and the Accessibility of Cryptographic Tools[5], CoBox has attempted to approach the interface research through two main lenses to support a Human Centric design approach, making use of Tatiana Mac's *Building Socially-Inclusive Design Systems*[6] and Una Lee and Dann Toliver's *Building Consentful Tech*[7], which itself builds on foundational work by the *Design Justice Network*[8] and the *Detroit Community Technology Project*[9].

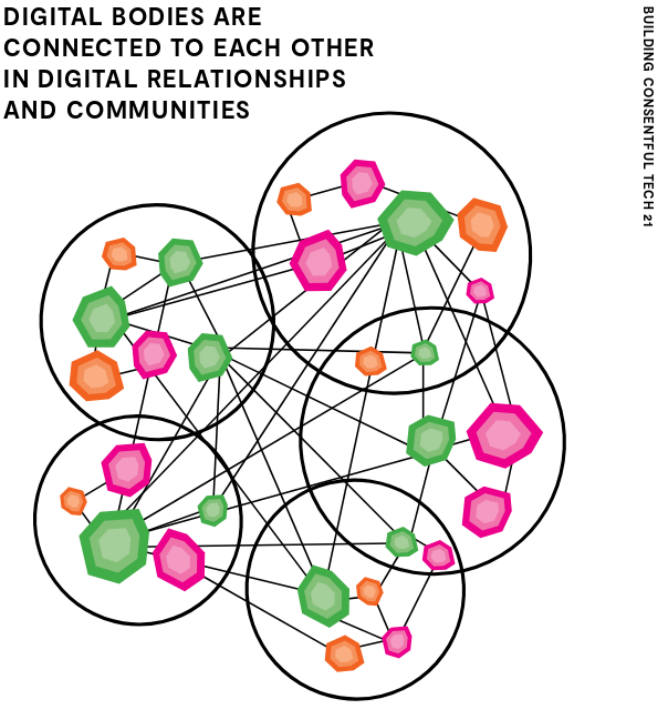
These can be summarised as:

1 G. Jones, *Cobox Command Line Demo Video*, 2019
2 Magma Collective, *CoBox MVP Mock Up*, *2019*
3 Denis Roio a.k.a. Jaromil, *Ledger Open Call*, 2019
4 K. Gibb, *Peer Testing and Usability Assessment*, 2019
5 G. Jones, *Accessibility of Cryptographic Tools*, 2019
6 T. Mac, *Building Socially-Inclusive Design Systems*, *2019*
7 U. Lee, D. Toliver, Building Consentful Tech, *2017*
8 Design Justice Network, *Collection of Zines*
9 Detroit Community Technology Project, *Teaching Community Technology Handbook*

Consentful Technology	*Socially-Inclusive System Design*
Freely given	Impact > Intent
Reversible	Perpetual Audit
Enthusiastic	Adaptable
Informed	
Specific	

With these principles as a guide, CoBox used three main vectors of peer facing research, and has utilised need finding interviews and peer design workshops to produce two reports which reveal our key findings: *CoBox in context: An overview of data management concerns in a group of co-operative organisations*[10], and *Engineering Trust - CoBox as a breakup-proof technology*[11].

Fig. 1: Photograph taken at Engineering Trust workshops, 2019



Following a protracted period of rapid need finding research at the inception and first few months of the project, our research whittled down the need finding report into a clear set of aims and deliverables for the MVP that would allow for the extensibility of the system while also aiming to meet the desires and needs of a broad a base of interviewed peers as possible. This was completed alongside developing the necessary lower level prototype components required to enable the kind of features we correctly envisioned coming out of the need-finding report.

10 S. Lerner, J. K. Brekke, M. Davarian, *CoBox In Context: An overview of data management concerns in a group of co-operative organisations*, 2019

11 A. Enigbokan, D. Hassan, Engineering Trust - CoBox as a breakup-proof technology, 2019

Throughout this paper we will refer back to the need finding report, in addition to the *Work Plan*[12] and *Mock Up*[13] documents produced in the prior months of the LEDGER programme. Points in our development process at which we have deviated from these documents can be explained by a change in design and approach following the integration of our need finding research in November 2019 into our development roadmap.

The MVP

Software

In order to deliver the MVP, balancing our initial Mock Up while ensuring our need finding research was properly integrated into our development roadmap, our team prioritised two key features. These commonly noted and necessary features for early adoption were:

> "Implement blind replication as a priority for all mutual back-ups
>
> Fast and safe ways to back up user keys are critical."[14]

Several prerequisites were necessary to enable blind replication of a shared file system, the major challenge being a multi-writer p2p file system. In addition, to ensure blind replication and privacy, we prioritised content encryption and asymmetric cryptographic authentication to build private spaces for groups to organise, persist files and synchronise data structures and information across a local area or remote connection.

Dat is a p2p hypermedia protocol, which provides public-key-addressed file archives which can be synchronised securely and browsed on-demand. It is particularly well suited to scaleable file storage systems, and will continue to be developed in such a way its core engineers. Dat is built on the hypercore protocol, a transport-agnostic message stream spoken between nodes in a swarm of peers. Kappa extends the protocol's exchange mechanism to include a meta-exchange, whereby using a shared public key peers share information about the data feeds they have locally, and choose which of the remote feeds they'd like to download in exchange.

An early specification from our need finding was that any build must be offline-first – functionality provided by the hypercore protocol. Data is persisted locally, and changes are synchronised across any available network connection. Several key features of the Dat and the Kappa stack fulfilled some of the need-finding's specifications.

> "Ensure that no single node can accidentally become responsible for destroying or exposing sensitive data belonging to someone else.
>
> Manage transfers of larger back-ups without crippling network speeds.
>
> Work should be possible online or offline.

12 Magma Collective, *CoBox Work Plan*, 2019
13 Magma Collective, *CoBox MVP Mock Up*, 2019
14 S. Lerner, J. K. Brekke, M. Davarian, *CoBox In Context*, 2

Ease of retrieval in case of local loss."[15]

Our stack takes advantage of the recently released hyperswarm distributed hash table (DHT) for replicating data from remote peers across the hypercore protocol. Using Dat and Kappa gave us three of these key features out of the box, while additional features have been either implemented or are road-mapped to enable, for example, easy and intuitive management of networking traffic.

In spite of our these strengths, during development we encountered a number of foreseen and unforeseen challenges that remain only partially or unresolved in the Dat ecosystem. Several of these were documented in our *Architecture Schematics Overview*[16] as part of the initial Ledger Mock Up. Our team saw this as an opportunity to contribute to a growing network of peer-to-peer innovators using these technologies, to expand the base of tools available to other

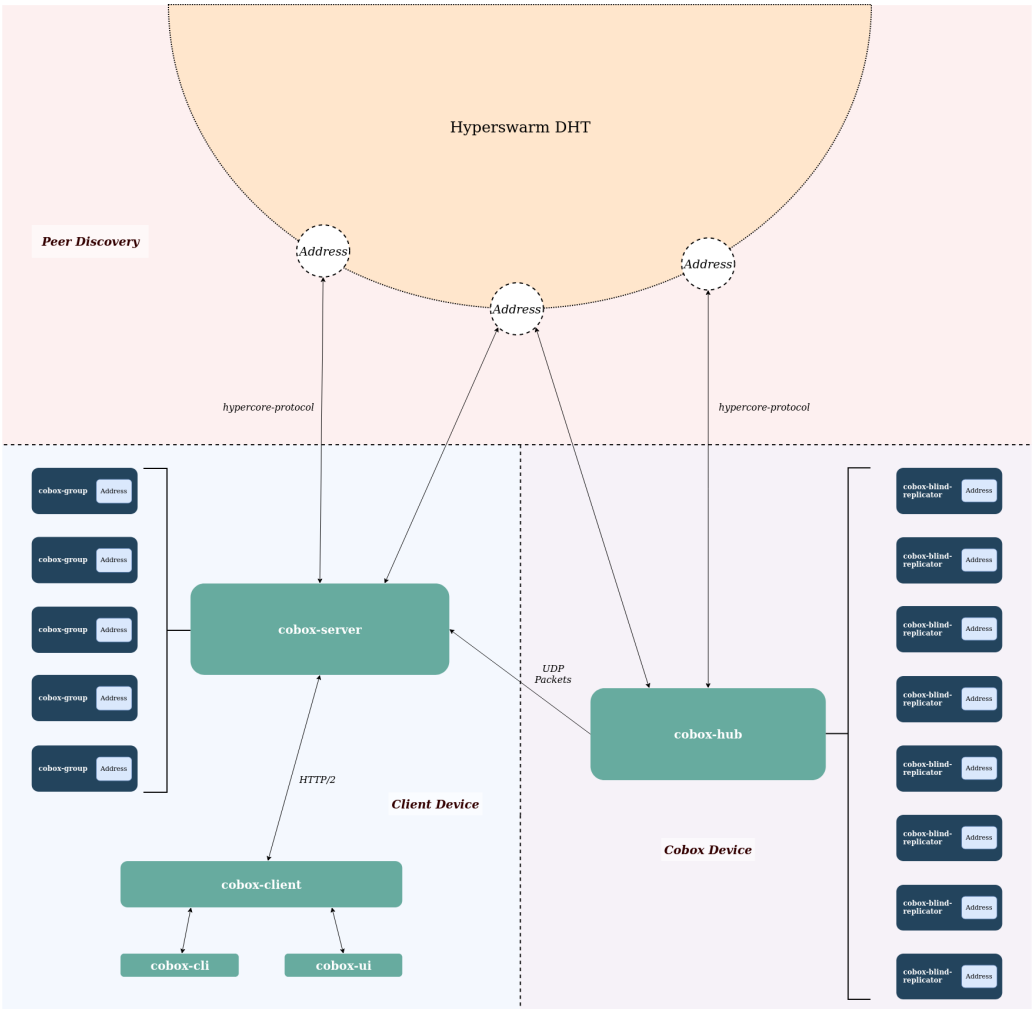


Fig. 2: A map of CoBox components and their relation to the hypercore protocol networking stack

15 Ibid., 2-3.
16 Magma Collective, *CoBox MVP Mock Up: Architecture and Schematics Overview*, 2019

developers in this OSS community. Drawing inspiration from existing project Cabal[17] – an IRC implementation and first implementation of a multi-writer application using the Kappa stack – CoBox took a similar approach, using Kappa's tools as a basis to extend the stack and innovate further.

The features detailed in this document bring our MVP to a point where our first adopters will be able to experiment with using our technology and we can begin a new phase of feedback and iterative design and development later in 2020. Below is a diagram to add some clarity on how the our different components fit together, and how they communicate over the hypercore protocol.

In the features section of this paper, we have documented some of the key challenges we faced, along with the solutions proposed and implemented in the MVP required to enable a minimal peer-to-peer cloud storage solution. In the packages section, we provide more detail as to the role the modules displayed in Fig. 2.

Hardware

A core component of building a distributed infrastructure for a data commons is a need for community managed or run hardware devices. As part of our Mock Up, we proposed to produce a hardware device that runs as a blind replicator, ensuring system uptime, and that peers are up-to-date, while ensuring the security and integrity of the data.

Our team has conducted research on using different open hardware ARM devices, in addition to making use of recycled laptops and other computers. We have also implemented a custom operating system for ARM devices – CoBox-OS – that comes preloaded with the relevant hub and administration software to smooth the process of setting up and configuring a blind replicator in a home or office or even a public location.

Concepts

Private Group

It became clear early that in order to provide a sovereign and secure, private cloud infrastructure rooted in commons economics and using a distributed system such as Dat, that we would need to assemble private cryptographic groups. In order for these groups to be viable for every day use, they must be easily discoverable by new peers, accessible for verified / permitted peers to authenticate and gain access, while solid and secure for those already internal to the group. Our early recognition was clearly articulated by interviewees and ratified in our need-finding report.

Address

An address is a commonly recognised identifier that an individual group shares. Using this address, peers may meet other group members or blind replicators (see below) on the distributed hash table, create a secure encrypted connection, and sync the latest changes across each device.

17 Cabal, *An experimental p2p community chat platform*

Blind Replicator

A blind replicator is a peer that remains connected to a given address to guarantee replication and backup of the latest changes made by group members, but it holds no knowledge of how to decrypt the data it contains – it is effectively a ‘dumb node’. Blind replicators can be setup to run from your home, a partner organisations office, or other location with a stable network connection, to ensure that your group’s backup retains integrity. Our interface provides an intuitive mechanism by which CoBox device owners can configure their device as blind replicators, form partnerships with different peer organisations to back up their groups. These actions are performed by any number of authenticated device administrators. Replication occurs by meeting that group’s peers at the address, and persisting the latest encrypted changes locally.

Features

Distributed

Multi-writer File System

In order to build a distributed cloud system, behaving like Dropbox or Google Drive, we needed a multi-writer file system. Dat archives, known as hyperdrives, currently enable only single-write actions. Each hyperdrive is composed of two hypercore append-only logs, each hypercore log has a public and a signing key. Only those who hold the secret signing key can make any changes to a hypercore. Therefore, in order to produce multi-writer Dat archives, whereby *n* peers could participate in a view on a file system, we needed to work from the base that:

1. Each peer has write access to their own hyperdrive (they hold the signing key).
2. Each peer has a read-only copy of other peer’s hyperdrives locally.
3. A multi-writer archive is the aggregated state of all hyperdrives under a shared address or public key.

To achieve these key requirements, we made use of Kappa's multifeed hypercore aggregation and replication manager, along with kappa-core, an index plug-in for multifeed. Using these tools our team built a tool called kappa-drive, which creates an abstraction of a file system across a set of hypercores using a set of local indexes stored in leveldb.

Conflict Resolution

Due to the distributed / local-first nature of the protocol, conflict resolution is necessary. Given that both Alice and Bob have made a declaration about a file when it was at different states, once syncing, Alice and Bob need to arrive at the same (or as close as possible) conclusion about the current state of a given file and the file system overall. In order to do so, we included a rudimentary implementation of vector clocks, a popular conflict resolution mechanism for distributed systems.

Vector Clocks autonomously detect causality violations by comparing logical timestamps. This significantly improved the performance of the drive. There is more room for improvement. Future development in this area would concern itself with introducing the ‘bloom clock’, using false positives to infer a partial ordering of events with high confidence.

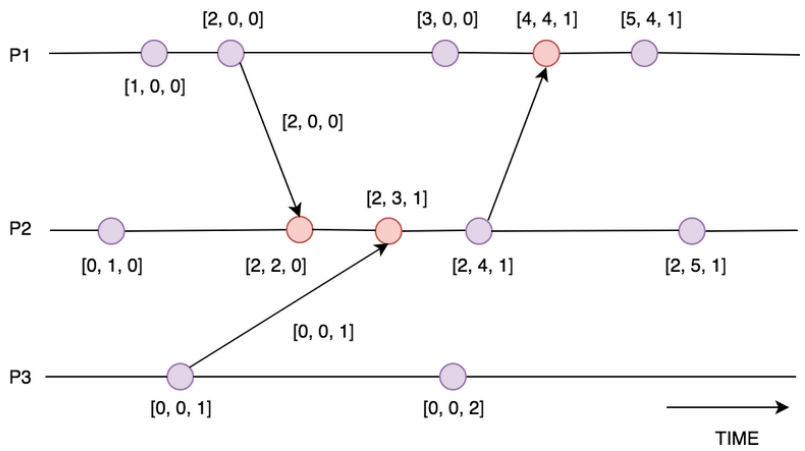


Fig. 3: With a vector clock, time is measured logically in terms of ‘number of events experienced’. Each peer maintains a ‘vector’ composed of their subjective clock for each other peer. When a peer experiences an event (a file system change in our case) they increment their own clock, and publish their version of the vector. These vectors can be merged by ‘adding’ them, and compared, to give a notion of time based on causality.

Privacy with Persistence

Content Encryption

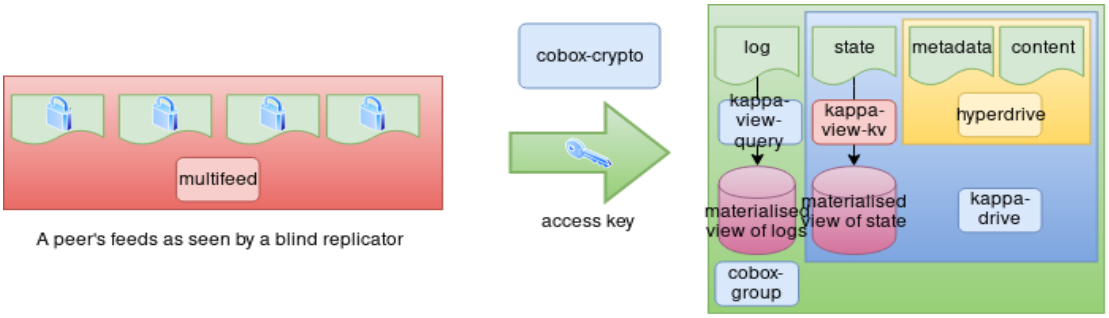
In order to enable a peer-to-peer, co-operative cloud, that ensures privacy-by-design, we had to guarantee that any partnership organisations replicating and backing up a group’s data on their own CoBox device, must not be able to access the data contained therein. Similarly, were any leak of a groups public address to occur, we must be able to guarantee that even if an attacker were to replicate the data, having it would be effectively useless.

> “Encryption on top of CoBox should be seamless.”[18]

To resolve this challenge, we determined that in order to gain full access and knowledge to a group, access its contents, we must have some kind of shared key or passphrase between group members. Each group currently relies on a single 32 byte symmetric encryption key used both to encrypt/authenticate and verify/decrypt messages. This is rendered in our key export system as a BIP39 mnemonic and easily human readable for communication out-of-band. Hypercore, the append-only log that lies at the base of the stack that replicate across the hash table, permits the use of a custom encoder when appending content to the log. Using the secretbox API in JavaScript library sodium-native - which provides bindings to C library libsodium - our team implemented our own cryptographic encoder for each hypercore’s content. In order to keep encryption keys secure, as any possible leak of a key would allow an attacker to decrypt the group’s content, they have been made unavailable through our interfaces. Instead, as detailed in the Authentication section below, we have included an invite code system.

18 S. Lerner, J. K. Brekke, M. Davarian, *CoBox In Context*, 2

Blind Replication



A peer's feeds as seen by a blind replicator

A peer's feeds as seen by a peer holding the access key for the group

With content encryption achieved, blind replicators can easily be established with partner organisations, by simply sharing the group's address with the partner organisations device administrator.

The CoBox hardware device - setup with cobox-os which comes with pre-installed package cobox-hub - is setup to receive commands from authorised administrators established on device setup. Once verified, device admins communicate over the DHT with the device across the hypercore protocol using a device specific unique address & encryption key combination - much like regular private groups. Across this connection, device administrators can send commands to a CoBox device to begin or stop replicating a given address. This is made accessible through the administration panel in the interface.

Our future development roadmap will begin to implement some of the core desired features specified by our early adopters in the need finding report, such as customisable replication scheduling, ensuring administrators can manage network latency and switch their CoBox device on and off at given times. There is currently no cap for on the amount of data a device will blind replicate for specific groups. Future development will allow administrators to configure their device's space capacity. For those peers requiring more support, device administrators can specific a top-level limit to ensure replication does not cease when backing up additional organisations data. These two core administration features create space for dynamic partnership agreements based on storage capacity and network uptime. In addition, new features added within the Dat stack enable us to be more granular about our replication policies, allowing group client devices to be more specific about what files and content they need, enabling peers to manage local storage space more efficiently.

Accessibility

In the *Mock Up*, we detailed one of the central adoption challenges cryptographic distributed systems face: key management.

What happens when someone looses a device?
Can access be revoked if a key is compromised?
Is it possible to restore an account or retrieve old messages when a key is lost?

Systems where peers manage their own keys are empowering, since peers themselves have ultimate control over their data. But this comes at a cost. There is no trusted intermediary who can intervene when things go wrong, such as by resetting the account. Our MVP addresses these issues in two ways. Firstly, by minimising the dangers associated with keys being lost or compromised, and secondly by offering and encouraging good key management techniques.

Key Management

Controlling group membership is a hard problem in cryptography, because once somebody has a key there is no way it can be taken away from them - access can be granted but not revoked. A common solution to this is to revoke access by creating a new group, distribute the new key to the desire group members, and import data from the old group. The cost of this approach is that the service is interrupted, and that a new address needs to be established in order to avoid duplicating the existing encrypted data, which in our case would mean blind replicators need to be reconfigured for the new group.

As detailed below, peers are authenticated before replication occurs. This means the group does not need to be forked following a lost key. If, for example, a peer loses a device holding keys, they can simply create a new account on a device, receive the encryption key via an invite code, authenticate with the relevant group and continue to use the same group as before without it needing to be re-initialised. Future features can be implemented to allow peers to easily ‘tombstone’ or declare multiple different device public keys to be the same person, easing possible confusion arising out of old expired public keys being present in the group.

Reducing the effect of lost of stolen keys directly addresses a key requirement in the need finding report:

> *“Fast and safe ways to back up user keys are critical. Given the challenges many people face in choosing and managing secure passwords this is the area that could make or break CoBox.”*[19]

19 Ibid., 2

Key Derivation

CoBox uses multiple asymmetric keypairs, which would make backing them up and recovering them very complicated. To simplify this problem we have used a key derivation function.

All keys specific to a peer or device are derived from a single parent key. Signing keys for each group are kept unique by including the group address in the derivation function. The implication is that each peer / device can be secured and recovered using only one 32 byte parent key.

In addition to their own parent key, group members should also safeguard their group encryption keys. These are not derived from a particular peer's key because they belong to the whole group, so all members should take care of them.

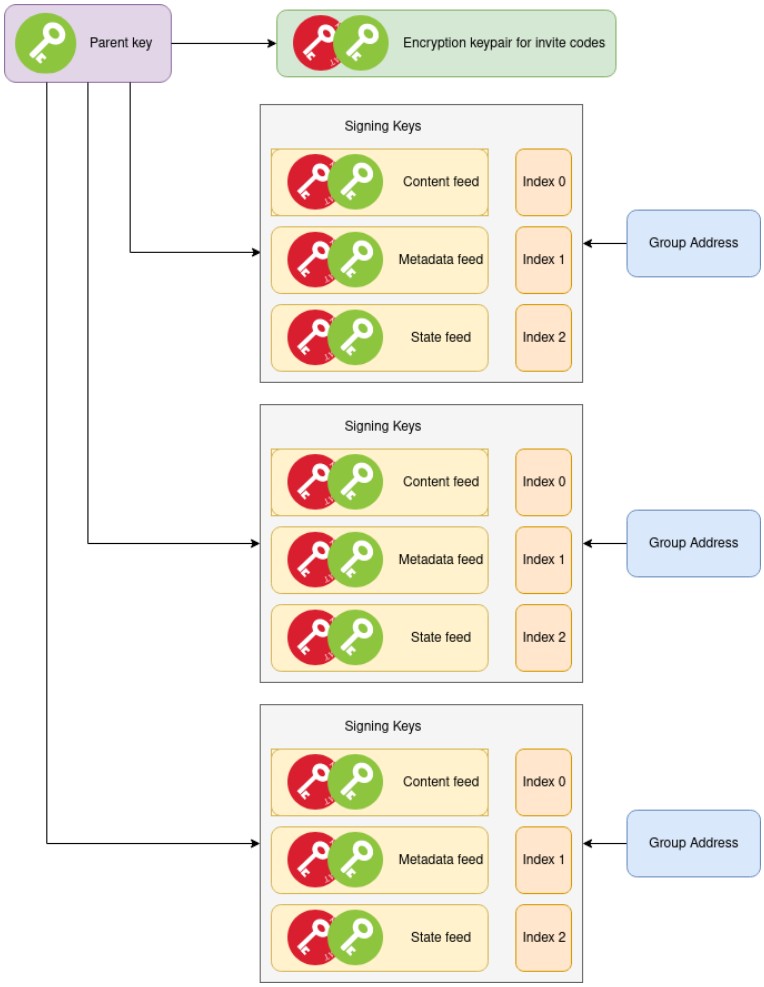


Fig. 4: Each group contains three feeds which have unique asymmetric keypairs. Our implementation derives these from a single master keypair with a given context, the group's unique address.

The parent key, and encryption keys for each group, are stored on the local filesystem, and are never exposed in the HTTP API. They are stored as binary files with restrictive permissions allowing only read access by the individual peer.

Paper Key Export

Our current key backup mechanism is simple but robust. A printable PDF can be generated and keys printed on paper, which should then be stored in a secure location. The PDF can then be deleted using a secure deletion mechanism. We feel this is a solution which also appeals to non-technical peers.

Authentication

We have designed an invitation protocol for on-boarding new peers to a private group simply over either a LAN, or from a distance using a clickable invite code served out of band, to ease the process while ensuring authentication. We also plan on developing this further, removing the need for the invite code altogether.

Currently, groups internally generate an ‘allow-list’, a summary of known public keys of all group members. Alice has sent Bob a copy of her public key, and Bob has shared that in the group. This means that when Alice attempts to gain access to a group, she can authenticate with whoever it is in the group she is connecting to. When swarming on the DHT, after the initial noise handshake and a secure connection is established, Alice meets Clare, and exchanges signatures. Clare knows Alice’s public key because she is in the same group as Bob and can draw from her group’s copy of the allow-list, so can authenticate Alice, and Alice is given permission to replicate from Clare. Derek attempts to join at the same time as Alice, but Derek’s public key has not been placed on the groups allow-list, and so because he cannot provide a valid signature, he will fail to authenticate with Clare.

While it may be the case that this helps lock out unknown peers, it does not necessarily smooth the process of sharing the content encryption key. In the above scenario, known blind replicators are authenticated just like regular group members are. In order to smooth the process for Alice to gain access to the group, content and all, though she does not hold the encryption key, we have derived two approaches, one of which we have implemented in the MVP, another of which we envision for future development. In our current implementation, Bob sends Alice back an invite code, which is a copy of the group address and its encryption key, encrypted with her public key. This is then plugged into the UI, and Alice has full and secure access to the group.

Usability

> *“Users felt most mastery over the tools they were using when things worked smoothly,and worried most when they did not understand what was going on. Design decisions should non-intrusively support awareness of things working well, of users being in control of their infrastructure, and not exclusively responsible for that of others. It should be easy to see that both your data and the data you host is replicated in enough other places to be safe if your building floods.”*[20]

> *“…just as much about technological solutions as it is about the social tissue that we can nurture-culture, [as a way] to distribute ownership and establish a shared immune system.”*[21]

Attempts to achieve human centered design goals by reducing the complexity of the interface whilst highlighting and reinforcing *what is actually happening*. It does so by drawing on the expertise of Magma Collective in on-boarding new peers into peer-to-peer systems such as secure-scuttlebutt as well as blockchain projects such as Bitcoin and Ethereum. One of the learnings from these experiences is *what can go wrong* with the system and how this can overload the social fabric leading to strain and collapse. On the other hand it is a design aim to communicate the underlying complexity, how this intersects with the principles of consentful tech whilst not overloading context on peers who are new to the system.

20 Ibid., *3*

21 Anonymous, *Peer Design Workshop*, 27th September 2019 , London, UK.

VueJS JavaScript Front-End

CoBox has taken a component-driven development approach building out components with *storybook*[22]. The interface has been driven by the research outputs mentioned in the previous section. This was followed by a rapid prototyping using *Figma*[23] to create clickable interfaces to surface required endpoints and do rapid peer A/B testing has then happened.

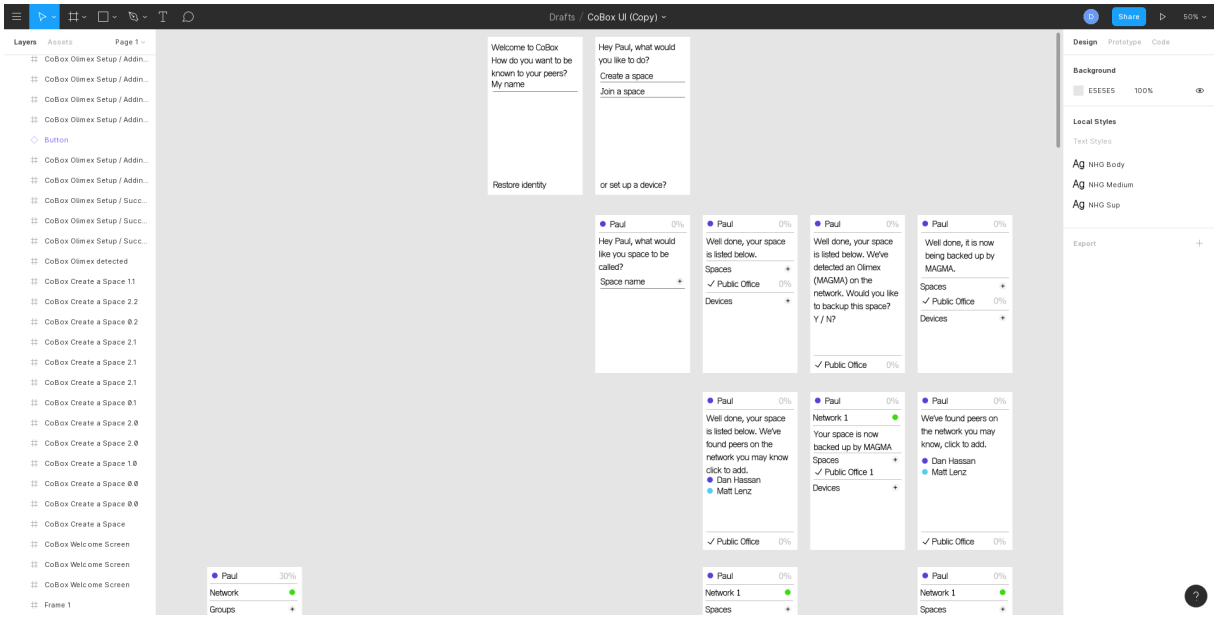


Fig. 5: Figma plots the pathways and peer journeys from setup to regular use

Once peer journeys were confirmed we figured out which API endpoints would be required to be able to support such functionality. Frontend framework selected has been VueJS which is popular for its ease of use and establishment - with a balance of sane defaults without locking you into top down design decisions. In our *Architecture Schematics Overview*[24] some proposed approaches were: file system mount, system tray server, all in the browser, browser extension, electron application or beaker browser integration. For our MVP, we have decided to build an open ended mobile-first framework, initially for trial in the browser. This does not close down the possibility that we could pursue any of the other routes outlined in the Architecture Schematics Overview, such as an Electron application, or a system tray application. Future peer design workshops will investigate, using our working prototype, where feels the most appropriate location for our application. Our interfaces will be simple, intuitive and clear with large script.

22 Storybook
23 Figma
24 Magma Collective, *CoBox MVP Mock Up: Architecture and Schematics Overview*

FUSE (File System in Userspace)

In order to make use of our peer-to-peer file system in a fashion similar to Dropbox, as per our original specification and the mock-up, we created a FUSE mount point for our file system abstraction. Using a regular file system in user space, the CoBox client software eases the process for peers to easily interact with, append, amend and archive files and directories to a group's drive. A key driver for having a simple FUSE file system is user familiarity and it being fundamentally against user-lock-in, another key requirement drawn from our need-finding.

> *"Build in support for non-technical professional tools that do not force an all-in approach with locked-in data."*[25]

JSON REST API

To make our file system accessible and group application layer visible to users and developers through more complex and user-friendly interfaces than a FUSE mount, we've composed a REST API using the popular JavaScript web application framework Express. This API opens a series of end-points from which a CLI and UI talk to, in order to switch network sync (replication) on and off, to establish scheduling, to mount a group's drive to a given directory, and to read data back from the file system. It also provides an end-point for querying our group's aggregated hypercores that contain peer's application layer data, in order to compile, for example, features as the allow list, a peer's self-defined name in a group, and other application layer data. To do this we've built an intelligent index for running map-filter-reduce queries over a collection of append-only logs.

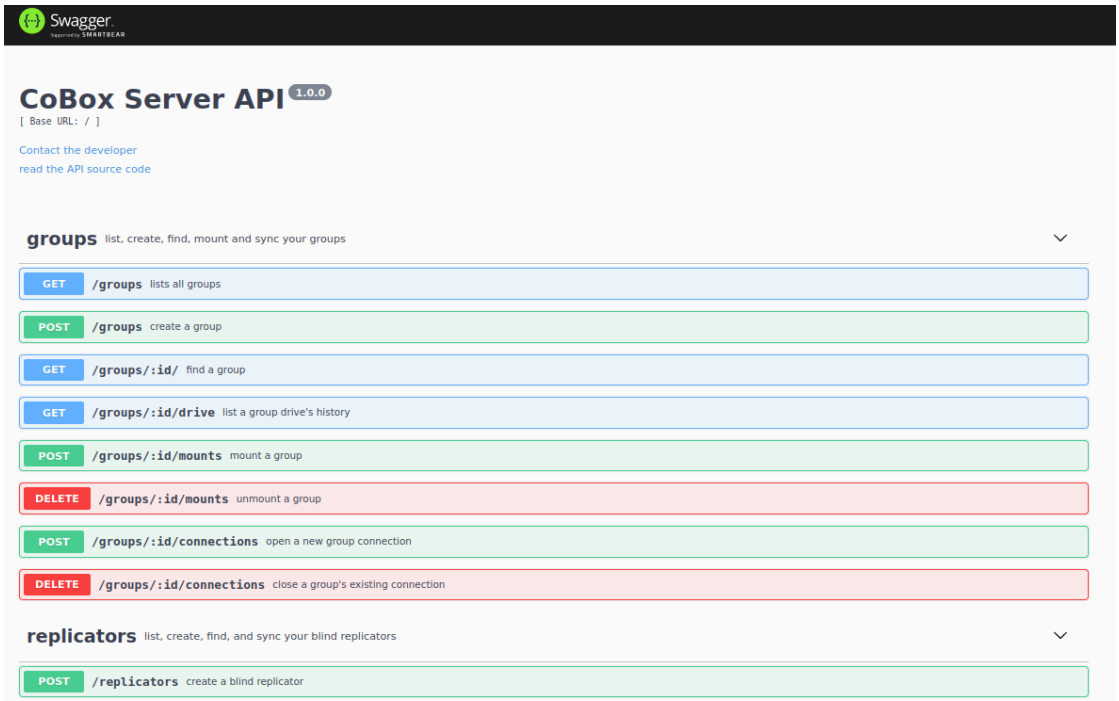


Fig. 6: Our swagger documentation can be found by running cobox-server and visiting http://localhost:3000/api-docs

25 S. Lerner et al., *CoBox in Context, 3*

We've implemented Swagger Documentation for clear request/response instruction for interested developers to easily compose their own front-end interfaces on this p2p stack.

Yargs Command Line Interface

We've implemented a simple CLI client for creating new groups, joining existing groups, opening a connection on the DHT to sync, mounting a group's file system, for querying a group's file system to render it's history and show remote updates. The CLI also enables starting and stopping of blind replication.

Packages

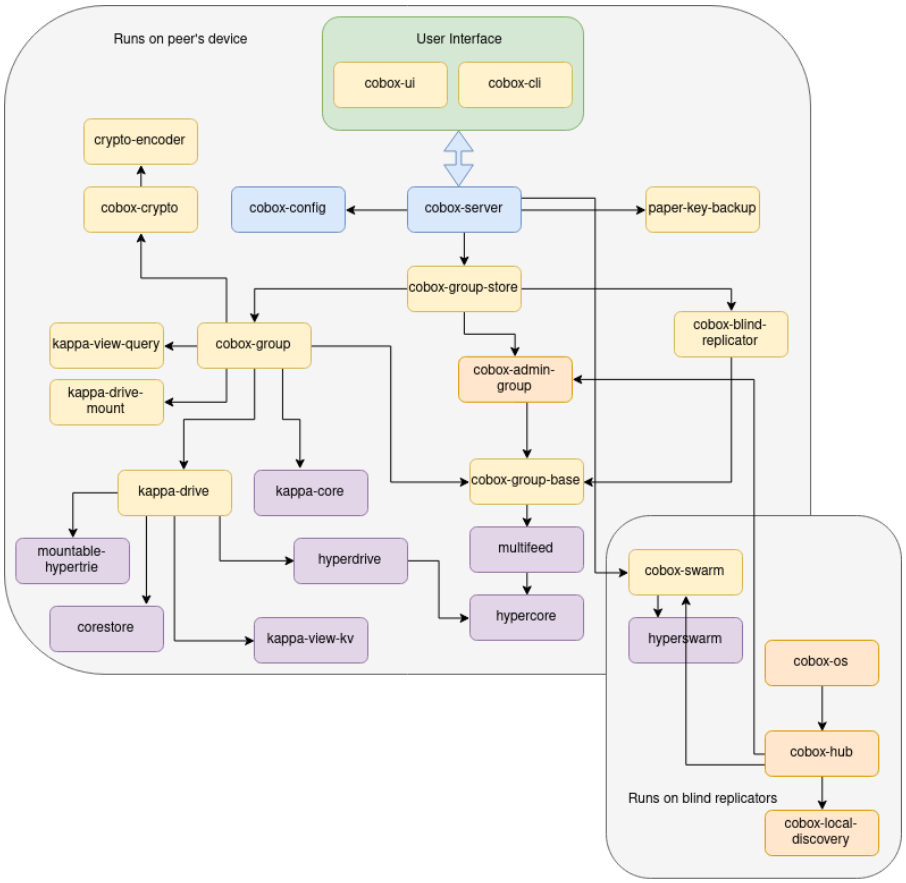
All our back-end packages are wrapped using git submodules in a mono-repository called cobox-core for more easy development and installation.

Below is a comprehensive list of the stand-alone libraries we have built since the project's inception.

Package Name	***Description***
cobox-core	A mono-repository containing all the main backend packages.
cobox-ui	A JavaScript front-end built with VueJS. Talks to cobox-server's REST API.
cobox-cli	A command-line interface build with javascript library Yargs. Talks to cobox-server's REST API.
cobox-os	A Devuan image builder for use on ARM devices or in VMs. Easy installation and configuration of your CoBox device, installing and running cobox-hub.
cobox-constants	System wide constants.
cobox-config	Local configuration settings stored in YAML.
cobox-server	A JSON HTTP/S REST API to run on the client. Exposes end-points to a front-end to manage your groups and blind replicators, setup CoBox devices, and perform admin functionality.
cobox-group-store	The repository pattern for finding a record based on a dynamic set of parameters from an existing collection. Used to initialize and cache multiple cobox-group instances in a single node process.
cobox-group-base	A base model for p2p private groups, containing storage and application layer data.
cobox-group	Adds content encryption and kappa-drive to cobox-group-base model. Provides an additional feed and indexes for group specific messages, and additional indexes for kappa-drive state / history.
cobox-blind-replicator	A simple wrapper for cobox-group-base model to initialise multifeed. No knowledge of content encryption so functions as a blind replicator.

cobox-admin-group	An RPC implementation over a multifeed instance. Content encryption included.
hypercore-crypto-encoder	A content encryption encoder for hypercore using an encryption key.
cobox-crypto	Cryptography primitives used by CoBox. All cryptography is compatible with the existing Dat ecosystem for future extensibility and interoperability.
kappa-drive	Multi-writer peer-to-peer filesystem, built on kappa-core and hyperdrive.
kappa-drive-mount	Mount a kappa-drive using FUSE.
kappa-view-query	A materialised view for kappa-core, aggregating all feed data in a multifeed instance, serving a dynamic and intuitive map-filter-reduce querying engine.
cobox-hub	Utilises cobox-admin-group with additional setup functionality including UDP packet broadcast over a LAN. Enables CoBox device administration over both a local or remote connection.
cobox-local-discovery	Find peers on a local network by broadcasting / receiving UDP packets.
cobox-command-schemas	Command message schemas for cobox-hub RPC.
kappa-private	Use private-box for sending encrypted messages between kappa-core feeds.
plot-logs	To plot a graph of CPU/memory usage from a CoBox log file using R.
paper-key-backup	Export cryptographic keys to an easily printable format.

These packages can be laid out in a hierarchical structure to illustrate the stack's dependency tree.



Evaluation

Its important to note that while our MVP can deliver a working peer-to-peer cloud storage solution for small collectives and organisations, there remain many core issues at the base of the stack which were beyond the scope of delivering an MVP for CoBox. Many of these were documented extensively in our mock-up document and are relevant particularly to the underlying Dat and Kappa stacks, and there are several additional features detailed in the need finding reports that would make our MVP adoptable. Our development team strongly believe that further development is needed to build a scalable cloud infrastructure. The core Dat protocol is a popular and actively developed project and will remain so moving into the future. We are in conversation with the key stakeholders in the ecosystem - indeed, we are recognised as one ourselves - and are in dialogue with the core team to collaborate to ensure our future aspirations are aligned and therefore our needs and the needs of CoBox's early adopters and future clients are well met.

The Future

Updates

KappaCore Version 7

After liaising with companion projects using the Kappa stack and comparing notes, we have specced out a planned release of a significant upgrade of the indexing system, Kappa Core, which will provide necessary improvements to our current index setup. The release of v7 is planned for early 2020. Included are breaking changes to its current API, so a sprints worth of development will be necessary to integrate the latest upgrades and make the stack production ready.

Features

Health Check

Its clear from our own experience, in addition to being ratified by every interview conducted in the research phase and summarised in the need-finding report, that our potential first adopters and clients who currently use cloud infrastructure depend on the stability of those cloud service providers to make it work. Any distributed and democratically organised replacement for corporate cloud services would need to create visibility and ensure transparency on the current state of remote backups. Metrics such as whether they are online, how often they are online, are they staying up-to-date, how regularly do they sync updates, and other important metadata that can provide peace of mind to organisations using a distributed backup mechanism. This is summarised in the statement from the need-finding report:

> *"It should be easy to see that both your data and the data you host is replicated in enough other places to be safe if your building floods."*[26]

In order to create adequate transparency around an group's remote backups, the team determined that a health check system should be high up the list of features for any roadmap for future

26 Ibid., 3.

development. This would include an intuitive interface rendering up-to-date information about the activities of your blind replicators. Each peer in a group can collect and regularly publish a report on its activities and what it has seen in terms of network traffic. All group members network metadata can then be aggregated locally to produce a common view on the current health of their group's backups.

Scheduling, Bandwidth Limitation and Customisable Replication

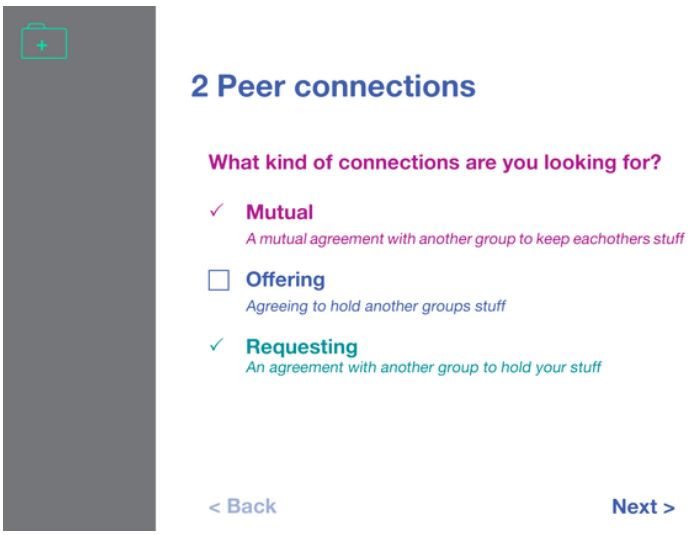
Drawing from our need-finding report, there was a clear articulation my multiple interested interviewees for the following feature set:

> "Manage transfers of larger back-ups without crippling network speeds.
>
> Allow users to allocate bandwidth around their working hours."[27]

Concern about bandwidth usage and the need for scheduling can be interpreted in two different ways, both of which will be useful. From a device administrators perspective, they will want to setup asynchronous time-based processes or cron workers to run on the blind replicator device. From a client devices perspective, similar functionality would be useful for either a global cron setting for replication of all associated groups, or more granular group-by-group scheduling. This was specified as a deal-breaker in several of our need-finding interviews. As such, while this feature is not core functionality for the MVP, this is a high priority moving forward.

Match-Making and the Registry

In our MVP document, we clearly articulated both a technical and a business case for establishing a registry of group addresses. The primary role of the registry, likely would be to provide a match-making service or functionality to enable organisations to easily locate partners to fulfil their backup needs. These would be fleshed out in the interface in the form of intuitive statements of need and non-binding / trust-based agreements, examples of which can be viewed in our initial clickable PDF[28]. On the platform itself, the registry would host requests - such as group's specification of data needs - and offers - detailing a device's storage limitations - for backup, and then provide a group address exchange mechanism to allow device administrators to



27 Ibid., 3.
28 J. K. Brekke, *CoBox MVP Mock Up: Clickable PDF*, 2019

easily begin blind replication. At the ecosystem level, the health check feature detailed above creates visibility on the implementation of backup agreements. Since improving our specification for distributed authentication, the registry can also act as PKI to ease the process of inviting new peers into groups.

The registry would be setup, either as a service - run by the CoBox project to assist with ‘backup match-making’ and expose these addresses for backup - or as a distributed global ledger - hosted by each CoBox device which acts as a stable node. Which path the CoBox project will take moving forward will depend largely on our business case, as well as the specifications and needs of our early adopters. There are clear privacy considerations to take into account when exploring the possibility of a distributed model, such as global exposure of group addresses.

Improved Authentication

In future, we can extend our authentication approach further, whereby Alice can ask Bob to meet her on the DHT at her public key as the meeting location. Alice and Bob can then conduct a similar authentication handshake, whereby Bob can verify that a peer is Alice, and so Bob can then send Alice the encryption key. Alice can then join the group, and no invite code is exchanged. The beauty of this mechanism is PKI can host peer’s public keys. All a group needs to do is find the peer they wish to add, and add them to a group. With future features such as the CoBox registry, a common infrastructure for hosting public keys and providing matchmaking for blind replication, this becomes significantly easier.

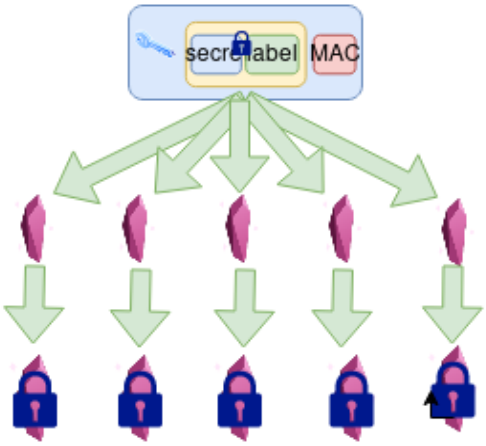
An effective distributed authentication mechanism for private groups provides us significant space for further development. By aggregating the allow-list and applying new application logic, such as a threshold signature scheme or Loomio-style poll, we will be able to introduce democratic consent mechanisms internally to groups, to give new members access or to remove existing members from a group. Given Edmund is leaving the organisation and no longer needs access to the group, Clare, Bob and Alice can all agree to revoke access to Edmund’s public key, thus ensuring Edmund can no longer replicate the latest changes to the group’s file system.

Nested Group Authorization

Sadly due time constraints we have not yet been able to implement an internal authorisation mechanism for groups. However there is a clear roadmap to create ‘nested groups’, allowing for custom internal permissions. In this case, Alice is a member of Magma ‘group’. Magma contains a list of nested group addresses. Bob and Clare are part of an internal working group for Magma’s research team, Magma/Research, and are collaborating on a document in their file system. This file system is not visible to regular Magma team members, though they are aware of the working group’s existence. Alice has recently joined the research working group and needs to gain access to their file system. Applying the same authentication logic as above, Clare and Bob authenticate Alice, giving her permission by adding her public key to Magma/Research allow-list. Alice has then been authorised to participate in a specific folder. The flow of this process can be smoothed significantly with an intuitive UI, meaning Alice should not need to do anything different, she will simply receive a notification stating she has been given access to a new area in Magma.

Social Key Recovery with Dark Crystal

Social backups allow group members to share responsibility for safeguarding keys. Keys can be sharded (broken up) into a number of components in such a way that each individual share has no information about the original key, yet a specific threshold of components can be combined to recover the key. The implication is that keys can be recovered from other group members following a specified degree of consensus within the group.



Dark Crystal is a social key management protocol which uses such threshold-based secret sharing techniques. We plan to introduce this key backup and recovery mechanism to CoBox.

CoBox-OS

CoBox-OS is a custom Linux operating system designed specifically for CoBox blind replicators. It is designed to run on ARM devices such as the Olimex Lime2, which will be our recommended CoBox hardware, as well as on a virtual machine, and comes with cobox-hub, the blind replicator software, pre-installed and ready to automatically run on booting.

Although we have a build script, CoBox-OS has not been extensively tested and needs further work doing. It should be noted the cobox-hub can also be run on other operating systems.

Conflict Resolution with ‘Bloom Clocks’

Future development in this area would concern itself with introducing the ‘bloom clock’, using false positives to infer a partial ordering of events with high confidence.

Multi-Multifeed as a Micro-service

Our package cobox-group-store is currently a rudimentary repository pattern for loading and caching multiple groups or multifeed instances into memory from which we can perform queries, operations and other functionality. A much more desirable layout would be to separate our collection of multifeeds into a micro-service, with an easy-to-use API to which our core REST API can communicate with using a client package. This would be a notable optimisation of our current setup, enabling more fine-grained process handling with lightweight stand alone packages. This would also be a significant contribution to the Kappa community - enabling developers to begin to use the kappa-core stack intuitively in languages other than NodeJS.

References

S. Lerner, M. Davarian, J. K. Brekke, *CoBox in context: An overview of data management concerns in a group of co-operative organisations*, 2019

A. Enigbokan, D. Hassan, *Engineering Trust: CoBox as a breakup-proof technology*, 2019

Magma Collective, *LEDGER Work Plan*, 2019

Magma Collective, *LEDGER MVP Mock Up*, 2019

J. K. Brekke, *Clickable PDF*, 2019

G. Jones, Cobox Command Line Demo Video, 2019

G. Jones, *Dark Crystal: Accessibility of Cryptographic Tools Report*, *2019*

K. Gibb, *Dark Crystal: Peer Testing and Usability Assessment*, *2019*

T. Perrin, *The Noise Protocol Framework*, 2018

M. Ogden, K. McKelvey, M. B. Madsen, *Dat - Distributed Dataset Synchronization and Versioning*, 2017

L. Ramabaja, *The Bloom Clock*, 2019

C. Fidge, *Timestamps in Message-Passing Systems That Preserve the Partial Ordering*, 1988

D. Keall, *How DAT works*, 2019

J. Kreps, *Questioning the Lambda architecture*, 2014

U. Lee, D Toliver, *Building Consentful Tech*, 2017

T. Mac, *Building Socially-Inclusive Design Systems*, 2019

Design Justice Network, *Principles for Design Justice*

Design Justice Network, *An Exhibit of Emerging Practices*

Design Justice Network, *Design Justice for Action*

P. Frazee, B. Newbold, *DEP-0010: Hypercore Wire Protocol*, *2019*