



POSSIBILITY SPACE • FULL USE CASE

07

Sign in without handing anything over

One identity across many apps, where the data stays with the person and each app only sees the slice it is lent.

BUILD STATE

Claimed

COMPLEXITY



Vision

Every app starts by asking for an account, and the account is where the work ends up living. Notes, photos, documents, the shape of a project: all of it accumulates inside somebody's product, and leaving means finding the export button and hoping it covers everything. The data was produced by the person and stored by the company, and the second half is the part that decides who has leverage.

The permission dialog is where this becomes visible. An app asks for access to a photo library, and what it actually takes is a copy. From that moment permission is a formality: revoking it later removes the app's ability to ask again and does nothing about what it already holds. There is no dialog for "read this once and do not keep it", because in that architecture there is nowhere else for the data to be.

Accounts do something else that nobody asked for, which is to merge. Work, family, and a political interest end up behind identities that a service can link, because each service insisted on identifying the person in the first place. Keeping parts of a life separate takes deliberate effort and usually fails at the first shared email address.

Swarm inverts where the data sits. The store belongs to the person, and an app is handed a scoped, revocable capability to read or write one part of it, writing into the person's own space rather than its own. Switching apps stops being a migration, because two apps can read the same material when the material is not inside either of them. A person can hold several separate identities and use them with different apps without a service being positioned to connect them, and the identity of the person stops being tied to the identity of any one machine.

It changes sharing between people too. Showing a document to one person stops meaning uploading it somewhere that then decides who else can see it; the person who wrote it says who may read it, and that decision is enforced by which keys exist rather than by a service's policy. The honest limit is worth saying in the same breath: giving somebody the ability to read something cannot be taken back later for what they have already read. What can be controlled is what comes next.

The result is a smaller thing to trust. An app still gets to see whatever you show it, and no architecture fixes that. What changes is that the app is no longer where your life is kept, so choosing to stop using it costs you nothing.

Architecture

The word identity hides three separate problems, and the brief separates them usefully: authenticating a person to an app, controlling what an app may do on that person's behalf, and controlling what other people may read. A fourth thing has to exist underneath all three, which is somewhere the person's data actually lives that is not inside any app. Swarm's answers sit at different stages of maturity, and saying which is which is most of the value of this page.

The premise: a personal store, not an app's database. All of this only means anything if there is a place a person's data lives that outlasts whichever app wrote it. On Swarm that place is a postage batch plus the feeds and manifests written against it: prepaid space the person holds the keys to. It is worth giving that a name at the product level, because "your postage batch" is not a concept anybody should have to learn. Swarm ID calls it a Drive, and the framing does real work — an app asks to write into a drive the person owns, the way a desktop application writes into a folder, rather than asking the person to deposit their data into the app.

Identity, decoupled from the node. The historical awkwardness is that a Swarm node has one identity, and a person is not a node. They may want several identities, and they certainly want the

same identity across devices and apps without running infrastructure per persona. Separating person from node is the whole point: identities become keys a person holds, personas become selectable, and the node becomes a way to reach the network rather than the thing that defines who you are. In product terms this lands as an account switcher of the kind Google popularised, where several identities are signed in at once and switching is one click, which is the interaction people already understand and the reason this can be built without asking anyone to think about keys.

Authenticating to an app, with the browser doing the isolating (implemented). One live approach opens the identity UI in a popup and, using the browser's Storage Access API, derives an app-specific secret from the master identity. What matters architecturally is where the isolation comes from: the browser's own storage partitioning between the identity origin and the app origin, rather than an application promising to behave. That is a stronger position than a library policing itself, and it is why the identity UI has to be served from a genuinely separate origin. Two caveats travel with the approach. Safari runs download-only, because its tracking prevention partitions storage in a way that blocks the upload path. And being connected is not the same as being able to write: with no postage batch and no subsidised gateway behind it, an identity can read and not upload, so the two conditions have to be checked separately.

One drive, several devices (implemented). A personal store that only works from one machine is not a personal store, and this is the part that looks trivial until two devices write at once. A postage batch is a two-dimensional grid: 65,536 buckets, with a chunk's bucket determined by the first sixteen bits of its own address, and a number of slots per bucket set by the batch's depth. A stamp commits a chunk to one bucket and slot, and on a mutable batch re-stamping the same position overwrites what was there, which is the lever everything else uses. Two devices writing the same batch would collide, so each bucket's slots are split into two partitions, with one reserved slot per partition holding that partition's lock and write counter and the data slots interleaved between them. Two partitions rather than one lane per device, because every extra partition shrinks a partition's per-bucket budget and brings on a full bucket sooner while other buckets still have room. Handoff between devices is ordered by commit, resuming at the exact published counter through a rotating state pointer, with fencing on release so a device that wakes up late cannot write over newer state.

Access control between people (shipped). Sharing on Swarm is encryption at the chunk level, and the presence of a key is the only thing separating private content from public content. A publisher grants access using grantees' wallet public keys. Each grantee's public key is their session key, and two further keys are derived from it by Diffie-Hellman. A lookup table living inside a Swarm manifest holds one lookup key and one encrypted access key per grantee, and the grantee list itself is encrypted with the publisher's own lookup key, so nobody but the publisher can even read who has access. This is the mechanism behind the plain-language promise that the person decides what to share and with whom.

What revocation does, and what it cannot do. The access control trie keeps a version history with timestamps, and a grantee keeps access to the versions they were originally granted even after being revoked from newer ones. Revocation is therefore forward-looking: it stops somebody seeing what comes next, and it does not retract what they could already read. Anything already decrypted and copied is outside anyone's control, permanently. This follows from the mechanism rather than being a defect in it, and any product built here should be designed around it rather than papering over it, because a promise of retroactive revocation is one that nobody can keep.

Permission between a person and an app is the open problem (research). Granting an app a time-bound, revocable authority to act on someone's behalf is hard for a specific reason: content-addressed storage is immutable, so "expired" and "revoked" have to be represented somewhere that can change. The approach on the table is a certificate signed by the person naming a session public key, a validity window, a permission list and a nonce, stored on Swarm and verifiable by anyone, with revocations published to a feed that verifiers check. Of the requirements, explicit revocation and compatibility with the access control trie are rated the hard ones. This is a designed approach with no shipped implementation, and anyone who needs it should budget design and review time rather than assume it exists.

PRIMITIVES

Content addressing

Single Owner Chunks

Feeds

Manifests

ACT

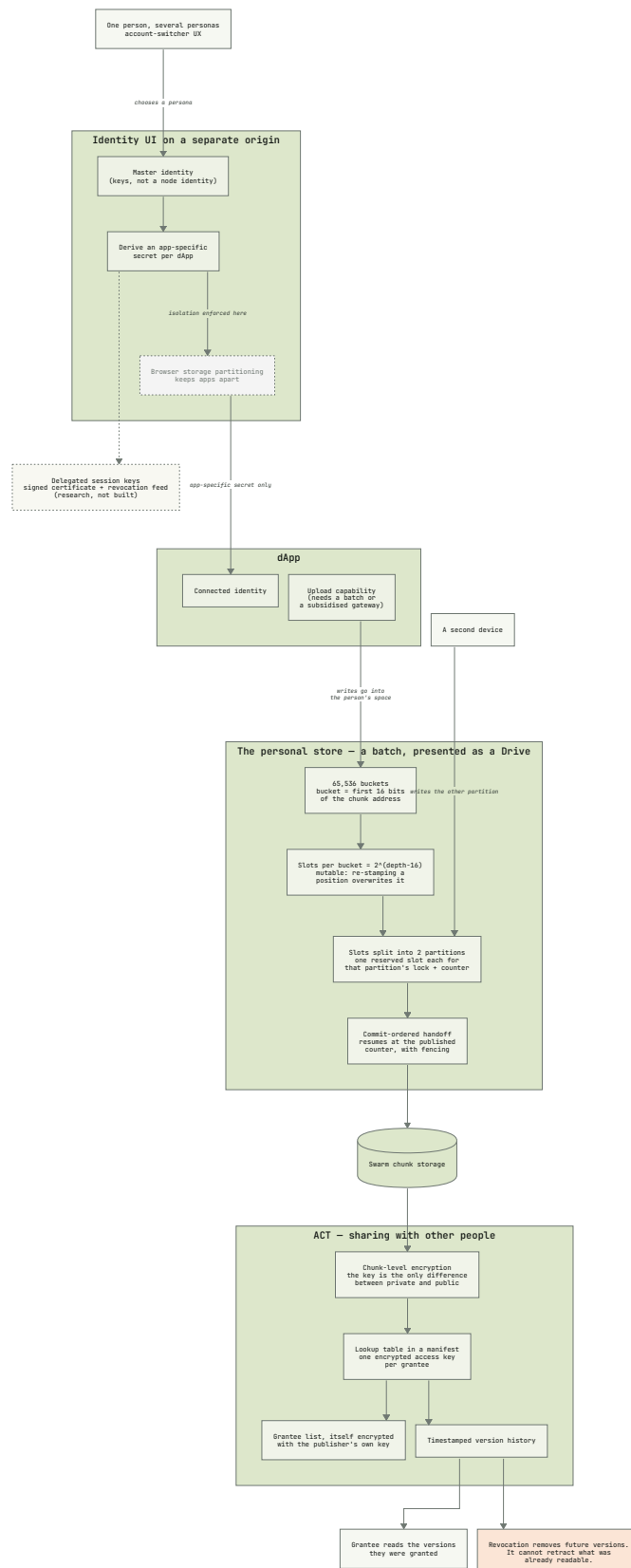
Postage stamps: mutable batches, buckets and slots

EXTERNAL DEPENDENCIES

Storage Access API

browser storage partitioning

swarm-cli



Data flow. Underneath everything is a personal store: a postage batch the person holds the keys to, presented as a drive. Identities are keys rather than node identities, so several personas sit behind one switcher. An app authenticates through a separately originated identity UI and receives only an app-specific secret, isolated by the browser's own storage partitioning. Writes land in the person's own batch, whose buckets and slots are partitioned so two devices never collide. Sharing with another person runs through a per-grantee encrypted access key, whose timestamped version history is why revocation stops future versions and cannot retract earlier ones. Delegated permission is the unbuilt branch.

Development

Nothing here needs inventing from scratch, and the useful decision is which parts to adopt rather than which to build.

Take an existing identity client rather than rebuilding the popup flow. The pattern is a client constructed with the origin of an identity UI and some metadata about the app, an initialise call, and a subscription to connection changes; @snaha/swarm-id is a published, maintained implementation of exactly that. The one thing to get right immediately is the write gate, because an identity can be connected while its upload capability is false when there is no postage batch and no subsidised gateway behind it. Check the identity and the ability to write separately, and treat a read-only session as a normal state rather than an error. Local development needs no certificates, since localhost counts as a secure context and the app and identity UI can run on two ports.

Treat the identity origin as part of the security model. Cross-app isolation is the browser's storage partitioning between the identity origin and the app's, so those origins have to be genuinely distinct and the identity UI has to sit on a domain that is trustworthy in its own right. If you deploy your own, that domain is now a security boundary. Plan Safari as a supported reduced mode rather than a bug in your application: authentication and downloads work there, uploads do not.

Model the person's storage as a drive, whatever you call it. The interface should let somebody see what an app has written on their behalf, and withdraw the ability to write more, without learning what a postage batch is. Underneath, that means the app holds a capability to write into space the person owns, and every write is attributable to the app that made it. If the app needs one drive across several devices, reuse the existing slot partitioning, per-partition counters, leases and commit-ordered handoff rather than reinventing them.

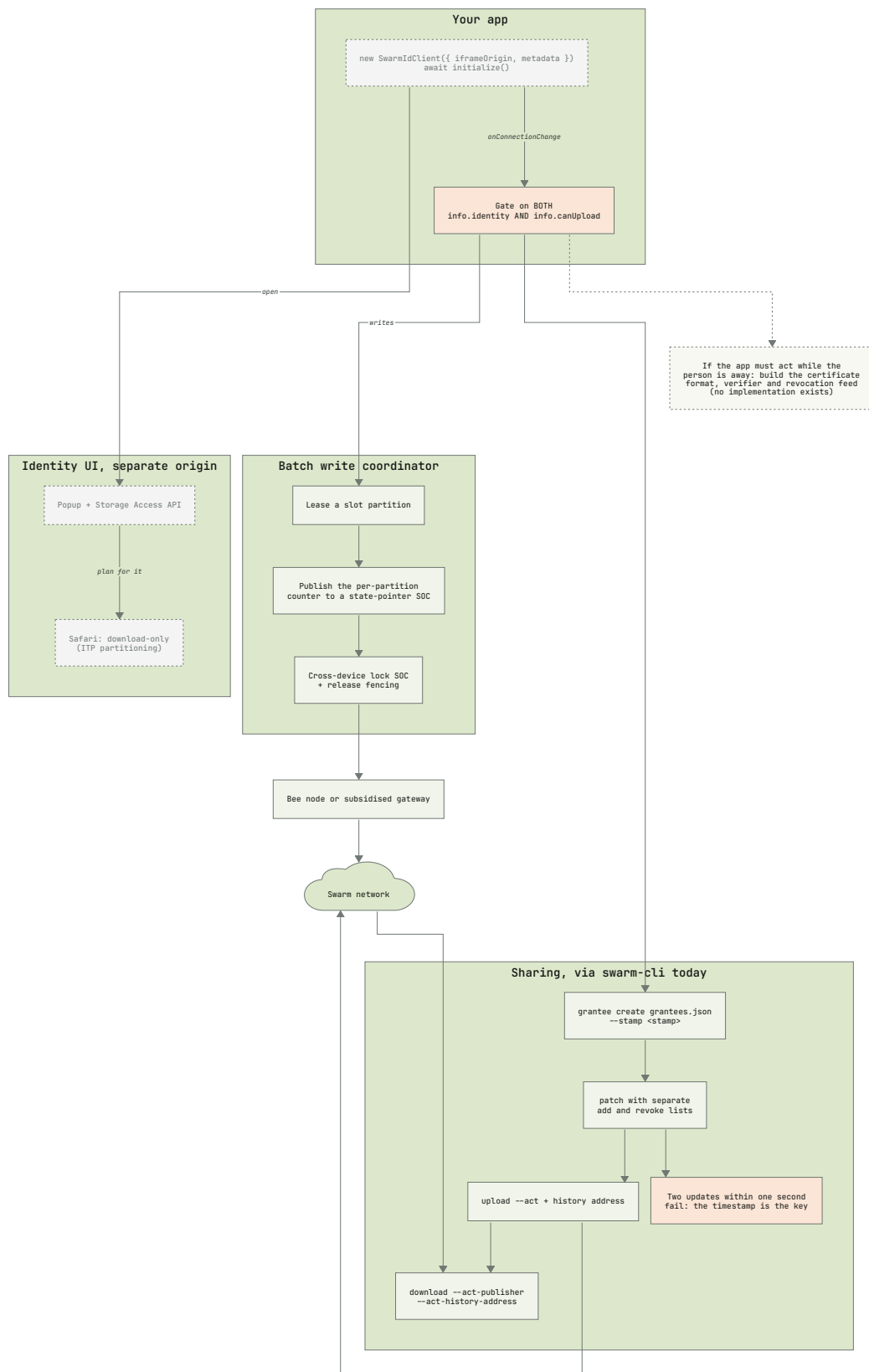
Sharing with other people means the access control trie, and today that means swarm-cli. Create a grantee list, patch it with separate lists of keys to add and keys to revoke, upload content with the ACT flag and a history address, and download with the publisher's public key and that history address. Two practical notes. The grantee list is keyed by timestamp, so calling an update again within a second of the last one fails with an invalid-input error; batch grantee changes rather than issuing them one at a time. And write the revocation semantics into the interface honestly, because somebody removing a name from a list will otherwise assume more happened than did.

If the app must act while the person is away, you are building the unbuilt part. Expect to write the certificate format, the verifier and the revocation feed yourself, and to have somebody review the result. This is the gap in the space, and it is where the interesting work is.

TOOLS & REFERENCES

- Access control concepts — <https://docs.ethswarm.org/docs/concepts/access-control/>
- ACT with swarm-cli — <https://docs.ethswarm.org/docs/develop/act/>
- swarm-cli — <https://github.com/ethersphere/swarm-cli>
- bee-js library — <https://bee-js.ethswarm.org>
- Feeds — <https://docs.ethswarm.org/docs/develop/tools-and-features/feeds/>
- Postage batches, top-up and dilution — <https://docs.ethswarm.org/docs/develop/tools-and-features/buy-a-stamp-batch>
- Swarm ID — <https://github.com/snaha/swarm-id>
- Swarm ID documentation — <https://swarm.snaha.net/docs>

- Swarm ID identity UI, live — <https://swarm-id.snaha.net>
- **External dependencies (not Swarm primitives):** the browser's Storage Access API and storage partitioning · @snaha/swarm-id and @swarm-id/ui, which are third-party rather than Swarm Foundation packages



Component assembly. The app holds an identity client pointed at an identity UI on its own origin, and gates writes on both a connected identity and an upload capability. Writes go through a batch write coordinator that leases a slot partition and publishes its counter so a second device can resume in commit order. Sharing runs through swarm-cli's grantee list and the ACT upload and download flags. Delegated session keys are drawn as the unbuilt branch. External components are dashed to mark them non-Swarm.

Reference implementations

Swarm ID is the project in this space, and it is nearing completion: a TypeScript client library, an identity UI meant to be served from a trusted domain, a demo application and a documentation site, all deployed at `swarm-id.snaha.net` and buildable locally without certificate setup. It contributes the two pieces this document leans on most. The first is the popup authentication flow, where the browser's own storage partitioning does the isolating. The second is the multi-device postage-batch partitioning, which is the most useful engineering artifact here, because sharing one batch across devices without chunk collisions is a problem most people would meet late and solve badly. Its Drive abstraction and its account-switcher interface are the two decisions worth copying at the product level, whatever the underlying implementation. Its session-key work is a research report rather than an implementation, so the permission half of this space is open.

The access control trie is a Bee feature rather than a project, usable today through `swarm-cli`, and no named application in this space is built on it yet. Anyone wanting sharing between people has a shipped primitive and an empty field.

The gap worth naming: authentication is built, personal storage is built, sharing between people is shipped and unused, and delegated permission between a person and an app is designed and unbuilt. That last one is the piece that would let somebody use several apps over one store without handing any of them standing authority, which is the thing this whole space is for.