

Slate Cryptographic Design

The end-to-end encryption of Slate's sync, written down so that it can be read and challenged.

Release	2
Date	10 October 2026
Describes	the design as it stands at commit e6cb4b766 of the private repository (sealing protocol version 2, relay API version 1)
Status	No external penetration or cryptography review of Slate has been performed. This document exists so that one can start from something written rather than from the code.
Contact	security@slateoutliner.com

PART I. WHAT SLATE PROMISES

1. Introduction

Slate is a notes application. Your notes are plain markdown files in a folder you choose, and they are the originals. The application's database is a cache that can be rebuilt from them, and sync is optional. When sync is on, each device sends an encrypted copy of its changes to a server called the relay, and the relay passes them to your other devices. The relay never holds a key that opens anything.

This document describes that encryption: which keys exist, how each one is made, and what each one protects. It describes how every encrypted object is bound to its place, and how a vault keeps its identity across resets. It describes how a lost password or a lost device is recovered from, and what an operator of the relay can and cannot do. It also lists what the design does not do, because a design document that omits its limits is an advertisement.

1.1 What the design promises

The promise comes in three tiers, and the rest of this document uses their names. A claim is exactly as strong as its tier.

Prevented. Cryptography makes these impossible for anyone who does not hold the account's keys, the relay included; they are not merely noticed afterwards.

- Reading a note, a file, the name of a file or folder, or a key.
- Forging or altering a sealed object so that a device accepts it.
- Serving a page's compaction under a cut other than the one it was sealed with, so that a device skips part of the page's history: the compaction does not open, and the device takes

nothing from it ([section 6.2](#)).

- Moving a sealed object to another place, whether another page, vault, generation, asset, revision or kind of object, so that it opens there.
- Giving a vault a new identity, or pointing a device at another of the account's vaults, without the authorship of someone who holds the account's keys. Neither the relay nor a thief holding only a session token can do it. Someone who knows the master password can, as any of the owner's devices can ([L-4 A lineage certificate proves authorship by someone who holds the account's keys, not by a particular device](#)).

Detected. These a hostile relay can attempt, and a device notices, under the conditions [section 8.3](#) states. The first five concern notes, whose histories devices compare. No image state crosses between devices, so for images only the last applies ([L-56 Images have no freshness signal](#)).

- Serving a device a note's history with a gap in its middle, or bytes that differ from what the device already holds at that place: the device notices, and that page stops syncing and shows as needing attention ([section 8.1](#); [L-39 A failed open or a divergence on a note's page raises no alert](#)).
- Withholding from one device objects that another device's heartbeat shows exist.
- Showing two devices different page lists where both stand at the same point in the list's history.
- Showing two devices different parts of one author's notes, caught between two moments at each of which a device held exactly what the relay held.
- Hiding one device's heartbeat from another, caught on a single observation the relay itself stamped.
- Serving a second, different record of an image at a revision the device already holds: the device keeps the first, drops the second and logs it, and shows nothing.

The second to the fifth show a sync integrity alert on the device. The first shows only as a failure of that page, with no alert ([L-39](#)). The check on withheld objects is bounded in completed sync rounds: three rounds that asked the relay and drained its answer. A relay that refuses the rounds instead is never accused, and the vault visibly stops syncing ([L-40 A relay that refuses every sync round is never accused](#)).

Neither. These look exactly like a slow network, and no design in which the relay is the only channel between devices can tell them apart ([section 8.2](#)). The check is the owner's own: sync two devices and compare what their screens say, or, for images, compare their files.

- Withholding new objects, or serving an old view: it reads as lag, never as wrong content, and heals when an honest relay answers ([L-53 Withholding and old views of notes read as lag](#)).
- A lasting partition of the devices into groups that never hear from each other again ([L-54 The eternal total partition is undetectable in band by construction](#)).
- Refusing or failing every sync round: the device shows an error or a decision to make, and does not sync ([L-40](#)).

- The evasions of the detectors that [section 8.3](#) names, each of which reads as staleness, or as nothing.
- Withholding an image, or serving an older revision of one: both screens can read as synced ([L-56](#)).

No redirection to another vault is left to this tier.

What a stolen session can do is a separate question, with a separate answer. The three tiers are about a hostile relay. A session alone, against an honest relay, is refused by that relay wherever a request would write into a vault's content, or remove a part of it. That is because each such request must present a value that only a holder of the vault's key can derive ([section 5.3](#)). That relay also refuses a session alone wherever a request would write the account's keys, its vaults' identities or its device slots. That is because each such request must present a value that only a holder of the password's wrapping key can derive, or prove the password or the mailbox ([section 5.2](#)).

What a session alone can still do is the following ([L-14](#) *A session alone can read, end things, and hold things; whoever holds the account capability with a session can destroy the relay's copy of a vault whole*):

- Read the account's wrapped keys and most of what else the relay holds for it, and guess the password offline against it.
- End its own session and a password change in flight on the device it was copied from.
- Hold some of the account's resources for as long as it keeps at it.

Against an honest relay, a session alone cannot destroy the relay's copy of a vault, or write anything around the vaults.

What "synced" promises. Once every device of a vault reads as synced, any one of them, or the relay, can be lost without losing anything ([section 9.3](#)). An image kept on a device on purpose, because it is too large to sync or its path is too long, is not counted. Three exceptions are known: [L-58](#), [L-59](#) and [L-60](#). The promise assumes a relay that answers: a relay that withholds can leave a device reading as synced over an old view, [L-53](#), [L-56](#).

1.2 Who this is for

A reader who wants to check whether the encryption does what the website says, and who is comfortable with the vocabulary of applied cryptography: a key derivation function, an authenticated cipher, associated data, a nonce. [Section 2](#) states the principles in plain words and [Appendix A](#) defines every term, so a careful reader without that background can follow the argument, if not every byte.

1.3 What this document is not

It is not a specification for writing a second client. The relay's request and response formats, the merge algorithm, and the local storage layout are out of scope except where one of them touches a key or a seal. It is not a proof. Where a property is tested, the test is described in words in [section 12](#). Where a property is only argued, the argument is given and marked as such.

1.4 Shape and conventions

The document is in three parts. Part I says what Slate promises: the promises themselves, the principles, the system and its trust boundaries, and the primitives. Part II describes the design, one part of it to a section, each moving from what that part is, through how it works and its parameters where it has them, to its security considerations; a caveat that belongs to one rule is stated beside that rule. Part III says what can go wrong: what the relay can see and attempt, how the design is verified, the limitations, each with its one entry in [section 13](#), under a number it keeps, and how to report a weakness. The order follows the published security design papers of comparable products.

Notation:

- `a || b` is concatenation of byte strings.
- Integers on the wire are big-endian and fixed-width; a width in brackets, such as `seq (8)`, is a byte count.
- `HKDF(key, info)` is HKDF-SHA256 expand over `key` as the pseudorandom key, with `info` as the label and a 32-byte output. Slate never runs the extract step: every input is already a 32-byte pseudorandom value, either a random draw or, for the master key, the output of Argon2id, which is only as unpredictable as the password it was made from.
- `AEAD(key, nonce, ad, plaintext)` is XChaCha20-Poly1305 sealing, returning ciphertext with its 16-byte tag appended.
- Domain strings are ASCII and are quoted, such as `"sLate:op:"`. [Appendix B](#) lists every one.
- A section is cited as "section 5.2". A limitation is cited by its number, and at its first citation in a section outside a run of citations by its headline in italics as well, as in [L-12](#) *Root keys are permanent, so there is no cryptographic revocation*.

2. Principles

Each principle below is a rule the rest of the design obeys. Where a later section seems to allow an exception, the principle wins and the section is wrong. What the rules add up to, for someone deciding how far to trust the relay, is the promise of [section 1.1](#) in its three tiers.

Files are the originals and are never encrypted at rest. Encryption exists for the copy that leaves the device. A lost key costs the encrypted copy on the relay and nothing else, because the plaintext was on your disk the whole time. This is the single decision that makes the rest of the design low-stakes to get wrong: no cryptographic failure described in this document loses a note on disk. Two things it does not promise: sync applies edits and deletions made on one device to the files on the others, and a lost sole device loses what only it held.

The relay is a blind mailbox. It stores and forwards opaque bytes. It authenticates accounts and enforces sizes and sequence, and it holds no key to anything. It admits a write into a vault only from a caller that presents the write capability of the generation the write goes into, a value derived from the vault's key, of which the relay keeps a digest. The capability opens nothing, and no device ever trusts it ([section 5.3](#)). It admits a write to the account's keys, its vaults' identities

or its device slots only from a caller that presents the account capability. The exception is the few requests [section 5.2](#) lists, which prove the password or the mailbox instead. The account capability is a value derived from the password's wrapping key and the relay's address, of which the relay keeps a digest on the caller's session. That capability opens nothing either ([section 5.2](#)). Every design choice that follows asks first what a hostile relay could do with it.

One master password, no stored secret. Every key on a device derives, through the master password and the account's email address, from material the person can re-enter on a blank machine, plus wrapped records the relay and the device keep that reveal nothing. The login credential also takes the relay's address. There is no stored salt to lose, no server-held key share, and no key the relay could substitute for one the password derives. Two derived keys are kept on the device between launches, in the operating system's secret store, so that a launch does not ask for the password ([section 5.5](#)). They are copies of what the password derives, not a secret the account depends on keeping ([L-23 A lost device whose cached key is still live is one of the owner's devices to the finder who launches it](#)).

Every ciphertext is bound to its place. Each sealed object carries, as authenticated but unencrypted associated data, a domain string naming what kind of object it is and the coordinate it lives at. A ciphertext served anywhere but its own coordinate does not open.

Identity is assigned fresh and never reused. When a history restarts, the restarted history gets a new identifier. On the note streams an honest relay keeps the first object written at a coordinate and ignores any other written there (a page's snapshot and compaction slots, [section 6.2](#), are overwritten instead), so a device that is served different bytes at a coordinate it already holds treats that as an attack. On the image plane a device restored from a backup can repeat itself, so a device there keeps the entry it already holds and drops the other ([section 9.2](#); [L-52 Two different image entries at one revision are caught only where a device already holds one](#)).

Forgery fails to open; staleness is tolerated. Reading, forging, altering or moving a sealed object is prevented: without the key a sealed object cannot be read, and a forged, altered or moved one does not open. Where it matters most, the note streams, an object that fails to open, or differs from what the device already holds at that place, is refused, and the page it belongs to stops syncing. It raises no alert ([L-39 A failed open or a divergence on a note's page raises no alert](#)). Withholding data or serving old data looks like a slow network, degrades to a "stale" state, and heals when an honest relay answers ([L-53](#), [L-56](#)). [Section 6.4](#) says what each plane does with an object that fails to open, and [section 1.1](#) says which misbehaviour is detected and which is not.

Boring primitives, owned composition. The primitives are Argon2id, HKDF-SHA256, XChaCha20-Poly1305 and SHA-256, from exact-pinned, widely used Rust implementations. Nothing cryptographic is invented here. What Slate owns, and what this document is mostly about, is the composition: which key seals what, and what each seal is bound to.

3. The system and its trust boundaries

3.1 Components

Component	What it holds	Trust
The application on a device	the plaintext files, the derived keys while unlocked, the wrapping key and the attest key cached in the operating system's secret store, the session's refresh token, the local databases	Trusted. Everything the design protects is protected from parties other than this device.
The relay	account records, wrapped keys, opaque sealed objects keyed by their coordinates, hand-over records and lineage certificates, a digest of each generation's write capability, a digest of the account capability on each session, sequence and frontier bookkeeping, sealed device heartbeats	Untrusted for confidentiality and integrity. Relied on for availability and freshness, which it can deny: a refusal shows, and withholding reads as a slow network (L-53 , L-56).
The mail path	the address-verification, password-reset and invitation messages, the notices sent in place of the first two, and the alerts that report a change to the account (section 10.1)	Untrusted. A mailbox compromise is treated as an account takeover, bounded as section 10.8 describes (L-26 <i>A mailed reset is an account takeover, bounded only by an alert the thief can delete and a window the thief can outwait; with the kit as well it is unbounded</i>).
The network	TLS between device and relay: the application dials a relay only at an address that starts with <code>https://</code> , or at a loopback address on the device itself (section 10.2)	Untrusted. TLS is a second layer; every guarantee below holds against a relay that terminates TLS and is itself hostile.
The owner	the master password, the recovery kit, and knowledge of which of their own devices are active	Trusted, and used as an out-of-band channel in section 8.3 .

3.2 Actors

The design is argued against these adversaries. Each later security-considerations subsection names which of them it is answering.

Under each adversary are listed, by number, the limitations of [section 13](#) whose entries name it, or something it holds whole, as able to do what the entry states; each entry states its conditions.

- **A curious relay operator**, who reads everything the relay stores and every request it receives, and answers honestly.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-9](#), [L-10](#), [L-14](#), [L-18](#), [L-28](#), [L-30](#), [L-31](#), [L-33](#).
- **A hostile relay**, who in addition serves any bytes they like, withholds or reorders anything, serves different devices different histories, and restores the relay from an old backup at will.

- Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-5](#), [L-6](#), [L-9](#), [L-10](#), [L-14](#), [L-15](#), [L-18](#), [L-19](#), [L-28](#), [L-29](#), [L-30](#), [L-31](#), [L-33](#), [L-39](#), [L-40](#), [L-41](#), [L-42](#), [L-43](#), [L-44](#), [L-45](#), [L-46](#), [L-47](#), [L-48](#), [L-49](#), [L-51](#), [L-53](#), [L-55](#), [L-56](#), [L-64](#), [L-67](#), [L-70](#).
- **A network attacker** between a device and the relay.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: none by what it holds on the wire; one that terminates or breaks TLS: [L-9](#), [L-10](#), [L-18](#), [L-38](#).
- **A person holding a session token** and nothing else: a refresh token copied from a device, with no password and no key.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-14](#), [L-21](#).
- **A person holding a lost or stolen device**, with whatever the device holds at rest.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-12](#), [L-14](#), [L-22](#), [L-23](#), [L-24](#).
- **A person holding the recovery kit** without the master password.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-5](#), [L-12](#), [L-13](#), [L-29](#).
- **A person controlling the account's mailbox** without the master password. Whoever completes a mailed reset chooses the new login credential and the digest the reset's session is opened under, and the relay takes the two independently. The application derives both from one new password, but a custom client need hold no wrapping key, and holds the capability whose digest it chose to send. Unless it chose a digest whose capability it does not know, that session is not a session alone ([section 10.8](#)).
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: [L-8](#), [L-10](#), [L-14](#), [L-15](#), [L-16](#), [L-17](#), [L-26](#), [L-27](#).
- **A person with a copy of the relay's database** but no live access to it.
 - Limitations whose entries name this adversary, or something it holds whole, as able to do what the entry states: none; [section 5.6](#) says what it can test offline.

Not in scope: a compromised device while unlocked, a person who knows the master password, and a malicious build of the application. No design of this kind defends against those, and this one does not claim to.

3.3 What crosses each boundary

From a device to the relay:

- an authentication hash at login, made for that relay's address ([section 5.2](#)),
- opaque session tokens on every request made under a session,
- sealed objects with their coordinates in the clear,
- version vectors and sizes,

- the write capability of the generation a request writes into ([section 5.3](#)),
- the account capability on every request that writes the account's keys, its vaults' identities or its device slots without proving the password or the mailbox,
- the account capability's SHA-256 at sign-in, at the completion of a reset and in a password change's final request ([section 5.2](#)),
- the application's version and platform,
- the name the person gave the device.

Never the master password, never a wrapping key, never a generation key, never plaintext content, never a file or folder name.

From the relay to a device:

- sealed objects at the coordinates the device asked for,
- the account's wrapped keys,
- hand-over records and lineage certificates,
- the account's identifier ([section 10.7](#)),
- bookkeeping.

The device opens every sealed object before believing it, as sections [6](#) to [9](#) describe. Much of the bookkeeping, such as which devices the account has, how far each has synced, and whether a root has been retired, is the relay's word, with no seal behind it. [Section 11.3](#) says what a relay can do with it. The cut that travels beside a page's compaction is not among that bookkeeping: it is in the clear, and the compaction's seal is bound to it ([section 6.2](#)).

4. Primitives and parameters

Purpose	Primitive	Parameters
Password stretching	Argon2id, version 0x13	memory 19 MiB (19,456 KiB), 2 iterations, 1 lane, 32-byte output; the password in Unicode Normalization Form C; salt is a 32-byte SHA-256 digest of the normalised email under a domain prefix
Key separation	HKDF-SHA256, expand only	32-byte pseudorandom key in, 32-byte key out, ASCII <code>info</code> label per branch
Sealing	XChaCha20-Poly1305	32-byte key, 24-byte random nonce drawn fresh for every seal but one (section 6.4), 16-byte tag, associated data per object
Digests	SHA-256	the email salt, the recovery-kit checksum, the Merkle forest, the registry and page-map digests of section 8.3 , the digest of a compaction's cut in its associated data, an image's content hash inside its sealed entry, the relay's record of each session token, of each generation's write capability and of the account capability each session was opened under

Purpose	Primitive	Parameters
Randomness	the operating system's cryptographically secure generator	vault, page, asset and device identifiers, root keys, kit seeds, nonces, session tokens, and, on the relay, the account's identifier
At-rest hash of the login credential, on the relay	Argon2id with a random 16-byte salt per hash	the library's defaults, which are the same 19 MiB, 2 iterations, 1 lane (L-36 <i>The relay's own credential hashing uses library defaults rather than parameters chosen for it</i>)

The parameters are versioned. The sealing protocol version, currently 2, is bound into every seal's associated data, so an object sealed under an earlier version cannot be replayed into a later one. Every wrapped-key record carries its own format version byte, and a record with an unknown version is refused before any key is used. Each small versioned structure that [section 6.3](#) names has its version checked before the open, and an unknown value is refused: a wrapped key, the wrapped kit seed, a sealed label, a hand-over record and a lineage certificate. The Argon2id parameters are versioned on the device, not in the labels: the application carries a list of parameter sets, newest first, and today the list holds the one set above. Where a password is proved to the relay, the sets are tried newest first. The device records which set the account's login credential was made under; nothing on the wire names it. The Argon2id parameters can be raised for an existing account without a new password, by a release that adds a stronger set. A sign-in that succeeds under an older set moves the login credential to the newest under the same password. This move is the password change of [section 10.3](#) with the password kept, called a re-tune. [Section 10.3](#) says what a re-tune does to the account's other devices. The `v1` and `v2` in an HKDF label version that label's own layout, not the Argon2id parameters.

Why these. Argon2id is the password-hashing competition winner and the memory-hard function current guidance recommends against GPU guessing. The parameters are the widely published minimum-recommended set. XChaCha20-Poly1305 is chosen over AES-GCM for its 24-byte nonce, which is long enough to draw at random without any nonce-management state, and over the 12-byte-nonce ChaCha20-Poly1305 for the same reason. HKDF-SHA256 gives many independent keys from one, with a label per branch, so no two branches yield the same key. All four are implemented by the RustCrypto project's crates, exact-pinned, pure Rust, with no C toolchain, which is what lets one build run on Linux, macOS, iPhone and in the browser-hosted simulator.

5. Keys

5.1 The hierarchy at a glance

```
master password + email, each in Unicode Normalization Form C
  | Argon2id(salt = SHA-256("slate:kdf:v1:email-salt:" || email))
  v
master key (32 bytes, never stored)
  +- HKDF "slate:kdf:v2:auth-hash" || relay's origin -> auth-hash: the login credential
  +- HKDF "slate:kdf:v1:wrapping-key"                -> wrapping key: stays on the device
  +- HKDF "slate:kdf:v1:attest"                      -> attest key: seals heartbeats

wrapping key
  +- wraps each vault's root key                      (random 32 bytes, permanent)
  +- wraps the account's recovery-kit seed            (random 32 bytes, as 24 words)
  +- HKDF "slate:kdf:v1:account-write" || relay's origin -> account capability

root key K_R (one per vault identity, for its life)
  +- HKDF "slate:kdf:v1:generation" || generation-id -> generation key: seals its objects
  +- HKDF "slate:kdf:v1:handover"                    -> hand-over key: seals hand-overs
  +- HKDF "slate:kdf:v1:lineage"                     -> lineage key: seals certificates

generation key
  +- HKDF "slate:kdf:v1:write-capability"             -> write capability: for the relay

recovery-kit seed
  +- HKDF "slate:kdf:v1:recovery-kit"                 -> kit key: second wrap of each root
```

Two things in this picture are worth reading twice. First, every key in it either derives from something the person can re-enter, or is a random key reached only by opening a wrap under such a key. What the person can re-enter is the password and the email at the top, or the twenty-four words at the bottom. No secret has to be synchronised for a new device to derive all of it. What it fetches from the relay is wrapped records, which reveal nothing without the key above them. Second, every arrow that says "wraps" is the same operation as sealing a note, with a different key and a different domain string, so the argument about transplanting a ciphertext in [section 6](#) covers keys as well as content.

5.2 The account keys

The master password is put in Unicode Normalization Form C before it is stretched. So an accented letter typed as one character, or as a letter followed by a combining mark, derives one key. It is Form C and not Form KC, so a ligature and the letters it joins stay two different passwords. The email address is trimmed, lower-cased and put in the same form. Both forms are fixed for every character Unicode has assigned ([L-74](#) *A password or email that contains a character Unicode has not yet assigned may derive a different key after an update*). The Argon2id salt is `SHA-256("slate:kdf:v1:email-salt:" || email)`, so a short address is still

above Argon2's salt floor and the salt is fixed-width. Argon2id over the master password and that salt produces the 32-byte master key. HKDF-SHA256 expands it under three labels into three independent keys:

- the **auth-hash**, which is what the device presents to the relay as its password. Its label is followed, with no separator, by the relay's **canonical origin**: the relay's address as `scheme://host:port`, with the scheme and host lower-cased and the port always written out. So the same password gives a different credential at every relay origin, and the same wrapping and attest keys at all of them. A credential verifies only at relays of the origin it was made for ([L-18](#) *The login credential is bound to an origin, with two exceptions*). The origin is the address's scheme, host and port and nothing after them. One trailing dot is dropped from a host name; otherwise the host is taken as written. So two spellings of one IPv6 address are two origins, and give two credentials. The relay hashes the auth-hash again with Argon2id and a random per-hash salt before storing it, so its database holds a hash of a hash. Neither derivation can be run backwards to the wrapping key, because HKDF branches are independent;
- the **wrapping key**, which never leaves the device. It wraps the account's root keys and the recovery-kit seed, and the account capability below is derived from it;
- the **attest key**, which seals the device heartbeats of [section 8.3](#). It is account-wide rather than per-vault so that any signed-in device can read every sibling's heartbeat.

The wrapping key derives one more value, the **account capability**: `HKDF(wrapping key, "slate:kdf:v1:account-write" || canonical origin)`, 32 bytes. Its label is followed by the relay's canonical origin with no separator, as the auth-hash's is. It is not a fourth branch of the master key, so the three keys above stay independent. Like the write capability of [section 5.3](#), it is not a key: it seals nothing, opens nothing, and no device is ever shown it or trusts it. A device derives it from the origin it dials for one operation that needs it, such as one write, one sync round, one heartbeat publish or one password change, drops it when that operation ends, and never stores it.

It travels as a request header on every request that writes the account's keys, its vaults' identities or its device slots. Those requests are:

- the write of a root's wrapped key, in each of its three forms;
- a hand-over;
- a retire;
- the pointing of a superseded generation at its replacement, another vault ([section 7.2](#));
- the write of the recovery-kit record;
- the request that binds a generation to its root;
- publishing roots and founding generations;
- registering a device, re-enrolling one, writing its heartbeat slot and removing one.

A session token does not carry the account capability. Against an honest relay, someone who holds a session and not the password's wrapping key cannot write the account's keys, its vaults' identities or its device slots ([L-10](#) *The account capability is a bearer value too, and proves only*

what the session's opener held). So a refresh token copied from a device, with nothing else, is refused at each request that list names.

These do not carry it:

- reads;
- signing out;
- refreshing a session;
- the requests that prove the password again: a password change, moving the credential to a relay's new address, deleting the account and declaring a change;
- writes into a vault's content, which carry the write capability instead.

A few requests write what the capability guards without it. None of them is open to a session alone:

- a password change's final request writes the kit seed, on a proof of the login credential;
- deleting the account removes everything, on a proof of the login credential;
- the undo or the cancel of a mailed reset restores the root wraps and the kit seed and purges what its window made, on the previous login credential or the mailed cancel code, with no session at all ([section 10.4](#)).

The relay keeps the capability's SHA-256 on the session, and never serves it:

- A sign-in sends the digest, and the relay stores what it is sent on the session it opens.
- The completion of a mailed reset does the same.
- A refresh keeps it.
- A password change's final request replaces it on the caller's session; a re-tune replaces the caller's and clears every other session's ([section 10.3](#)).
- Moving the credential to a relay's new address changes none, because the key does not move: the device that moved signs in afresh at the new address, and the account's other devices go on at the old one ([L-71 A device left at a relay's old address by another device's move is stranded by its next sign-in](#)).

The relay admits a write the account capability gates only when the capability presented hashes to the digest on the calling session. The comparison is made in constant time, before any answer that depends on what the relay holds for the account. Only the session's own refusals and the request's own checks, such as its sizes or a missing certificate, come before it.

Any of these earns one refusal, which a device can tell apart from every other:

- a missing, malformed or wrong capability;
- a session whose digest is cleared;
- a request with no capability and no session, which earns that refusal and not the answer an expired session gets, because the header is checked first.

An application that holds no wrapping key, locked or with its cached key expired, refuses such a write itself, before it dials. A device that meets the refusal takes it to mean that the key it holds is not the one its session was opened under. It ends its session, on the relay first if the relay answers. So, where the relay does not answer, the relay's session outlives the device's. It drops the access token, the session, every unlocked vault key, and the wrapping and attest keys with their entry in the secret store; that entry stays where the store refuses the deletion. It keeps its account state, records why, and asks for the master password. Where the key it holds is the new key of a password change still open on it, it asks instead for the password from before that change. The record survives a restart. Only a sign-in with the password clears it. An unlock does not, and nothing dials until that sign-in. What the capability proves and what it does not is stated in [L-10](#).

[Section 10.3](#) says which roots a master-password change reaches, and which keep the wrap they had.

5.3 Root keys and generation keys

A **vault** is a folder that syncs. When sync is first turned on for it, the device mints a random 16-byte **root identifier** and a random 32-byte **root key** `K_R`. Both are permanent for the identity they name: they name and unlock the vault until a restart gives it a new root ([section 7.2](#)). The root key is wrapped under the wrapping key and stored on the relay in a slot named by the account and the root, and on the device.

What the relay actually stores sealed objects under is not the root but a **generation**: a random 16-byte identifier for one lifetime of the vault's history. A restart of the history, which [section 7](#) describes, mints a fresh generation. It is under the same root unless the restart cannot, or should not, go on with the old root key. Each generation's sealing key is derived, not minted: `HKDF(K_R, "slate:kdf:v1:generation" || generation-id)`. A device that holds the root key can therefore open any generation of that vault, including one minted after it last synced, with no new key material from anyone. The relay keeps no separate wrap of a generation's key: a device reaches a generation's key only through the root, by deriving it, and keeps it, wrapped under the wrapping key, on the device alone ([section 5.5](#)). Every generation's key is derived: a new root's first, each one a restart hands over to, and one the relay holds no record of the key's source for. No generation's sealing key is its root key.

Each generation's key derives one more value, the generation's **write capability**:

`HKDF(generation key, "slate:kdf:v1:write-capability")`, 32 bytes. It is not a key: it seals nothing, opens nothing, and no device is ever shown it or trusts it. It is a bearer value for the relay. A device sends it, as a request header, with every request that writes into the generation's content:

- a sync request;
- an upload of a page's or an image's bytes;
- the vault's label;
- the request that binds a generation to its root;
- the purge of a page or of the vault.

A vault's operations are fetched only with the write capability. The request that makes the relay's record of a generation fixes the capability. Three requests make one: the binding of a generation to its root, a hand-over, and the publish that links a new identity. Each presents the account capability of [section 5.2](#) as well. A request into a generation the relay holds no record of makes none, and is refused. The relay stores its SHA-256 there, in a field no later request rewrites. From then on it admits a write into that generation only from a caller whose capability hashes to the stored digest, compared in constant time. Any other is refused, and nothing is written. A request that writes and carries no well-formed capability is refused. So, against an honest relay, a caller that holds a session and not the vault's key writes, overwrites and deletes nothing inside the vault: not a note's operation, a registry entry, a snapshot, a compaction, the label, an image's register entry or its bytes ([L-9](#) *The write capability is a bearer value, and is not wiped everywhere*). It cannot purge a page, or the vault by the request that names it. A session alone cannot change a vault from the inside. The whole of a generation is still discarded by a retire or a hand-over, which carry no write capability and present the account capability instead (below). The write capability derives from the generation key and not from the root key, so that a device holding a generation's key alone can still write. A device derives it when it needs it and never stores it.

What the relay enforces is exact, and narrower than "only a key holder can write": [L-8](#) *Whoever holds the account capability with a session can publish roots and found generations of their own, with write capabilities of their choosing, up to the account's limit on vaults*. Three kinds of request carry no write capability: a hand-over, a retire, and the pointing of a superseded generation at its replacement ([section 7.2](#)). The write of a root's wrapped key carries none either, except the publish that links a new identity, which carries its first generation's ([section 7.2](#)). Each presents the account capability instead. An honest caller of a retire or of that pointing may hold no key for the generation it names: a device that cannot open the root, or one starting afresh after a forgotten password. A hand-over's caller does hold the root key, and could derive the parent generation's capability. It presents none because an interrupted hand-over is sent again from a record kept on the device, which holds no write capability. A session alone therefore cannot hand a vault over or retire it, and whoever holds the account capability with a session can. [Section 7.4](#) states the cost. A hand-over carries, in its body, the digest for the generation it hands over to. [Section 5.6](#) says what each capability is worth to each adversary. What a session alone can still do is stated in [L-14](#) *A session alone can read, end things, and hold things; whoever holds the account capability with a session can destroy the relay's copy of a vault whole*.

5.4 The recovery kit

An account has one recovery kit. The device makes it:

- at registration;
- at the first sign-in of an account the relay records as still owing one (a relay that requires address verification records that on every account it creates);
- or, failing both, when sync is enabled or a vault rebuilt on a device that holds no kit and finds the relay's slot empty.

A device that finds the account's kit already on the relay adopts it instead, once it opens under the wrapping key. Making it means minting a random 32-byte **kit seed** and showing it as twenty-four words and offering it as a small file ([L-63 The recovery kit has rough edges when a publish fails or two devices race](#)). That device refuses to enable sync until the person has acknowledged the words, and any unlocked device can show them again. The kit key is `HKDF(seed, "slate:kdf:v1:recovery-kit")`, and under it every root key is wrapped a second time, beside its wrapping-key wrap. Unwrapping either wrap yields the same `K_R`, so the kit reaches a vault by exactly the route the master password does and by no other.

The seed is also kept, wrapped under the wrapping key, on the device and in one account-level slot on the relay. That is what lets a root minted long after the words were shown still get its second wrap: any unlocked device unwraps the seed, derives the kit key, wraps the new root, and drops both. Neither the seed nor the kit key is ever at rest unwrapped, and neither is held in session state; each costs one unwrap to recompute. The relay refuses to store a root's wrapped key that arrives without its kit wrap from an account that holds a kit seed or still owes its first kit. It refuses a kit wrap from an account whose seed it does not hold. The application, for its part, makes or adopts the account's kit before it enables sync for a vault or rebuilds one. On a relay that does not require address verification no account is recorded as owing a kit, so there, until the kit's seed reaches the relay, the application's rule stands alone ([L-62 On a relay that does not require address verification, until the kit's seed reaches the relay, only the application keeps a vault from being published without its kit wrap](#)). What the relay checks is presence: it cannot tell whether the kit wrap opens under the kit the owner saved, and does not try. It takes a write of the seed's slot only with the account capability or in a password change's final request. A reset's undo or cancel puts back the record the reset replaced, at its generation before the reset ([section 5.2](#)). Otherwise it moves the slot's generation, which [section 10.6](#) describes, one step at a time:

- while it holds a seed, only to one above the stored generation, or at the stored one with the same bytes, which is a retry;
- while it holds none, to any generation up to half the signed 64-bit range;
- and never above the signed 64-bit maximum, so no write it admits leaves the record unable to move.

The word encoding is the BIP-39 construction, re-implemented over the vendored English word list: `bits = seed (256) || SHA-256(seed)[0..8]`, split into twenty-four 11-bit indices. The twenty-fourth word therefore carries three bits of the seed and the eight checksum bits, and a mistyped word fails the checksum with probability 255 in 256. The remaining case derives a seed that opens nothing. The file rendering is a one-line header, `slate-recovery-kit v1`, followed by the words one per line. After trimming surrounding whitespace and blank lines, the header is compared byte-for-byte, not parsed, so a future layout is a refusal rather than a misread.

5.5 Where each secret lives, and for how long

Secret	Where	Lifetime
Master password	nowhere at rest; typed	from the keystroke to the end of the request that carries it
Master key	process memory during derivation	the derivation
Wrapping key	process memory; one copy in the operating system's secret store (Secret Service on Linux, the Keychain on macOS and iPhone)	the session; 30 days after the password was last typed the application treats the stored copy as expired and deletes it the next time it reads it
Attest key	process memory; stored beside the wrapping key	the same
Auth-hash	sent at login to the one relay whose origin it is made for; that relay stores an Argon2id hash of it	until the password, the relay's origin or the Argon2id parameter set changes
Root key	process memory while unlocked; wrapped on the device and on the relay	the life of the vault identity it names; re-wrapped on a password change, kept by a hand-over, and replaced only by a restart that mints a new root (section 7.2)
Generation key	process memory while unlocked; wrapped under the wrapping key on the device alone; never written to disk unwrapped. The relay keeps no wrap of it	in memory, until lock, sign-out or exit; the cached wrap, until the generation is replaced
Write capability	process memory, derived from the generation key for one sync round or one write, and the header of each request that carries it; never at rest on the device	on the device, the round or the write that derived it; the relay's digest, the generation's life
Account capability	process memory, derived from the wrapping key and the relay's origin for one operation (one write, one sync round, one heartbeat publish or one password change), and the header of each request that carries it; never at rest on the device	on the device, the operation that derived it; the relay's digest, the session's life
Kit seed	wrapped on the device and in one relay slot; the words wherever the owner put them	until the kit is replaced
Kit key	process memory for the length of one wrap, or of one password change's or kit reset's pass over every root it re-wraps	the operation that derived it
Hand-over key, lineage key	process memory for the length of one seal or open	the same
Session tokens	access token in memory only, 15 minutes; refresh token, 30 days, rotated on use	as stated

- **Master password.** It is typed, then held as an ordinary string in the web view, and in the application's own handling of the request that carries it until that request has finished. It is not wiped.
- **Wrapping key** (and the attest key, whose lifetime is the same). What its Keychain entry allows on a Mac and on iPhone is [L-37](#) *The Keychain entry on a Mac carries the platform's defaults*. The application deletes the stored copy on sign-out and when the relay refuses the account capability ([section 5.2](#)), unless the configuration that names the entry cannot be read or the store refuses the deletion. 30 days is a default the configuration can raise without ceiling. The in-process copy is not subject to it. The store itself enforces no expiry, so an entry the application never reads again stays.
- **Auth-hash.** A relay whose address changes can have the login credential moved to the new origin over a live session, with no key moving.
- **Generation key.** Its wrap under the wrapping key is kept in the device's sync cache. While a password change is under way, it is also kept wrapped under the new wrapping key in that change's journal in the account store. The journal's copy lasts from the start of the change until the change finishes on this device, which for an interrupted change is a later attempt, or until the device's account state is cleared.
- **Write capability.** The relay keeps its SHA-256 beside the generation's record. While a hand-over is under way, the device's account store holds the digest for the generation being handed over to, never the capability.
- **Account capability.** A sync round and the heartbeat are handed the capability, never the wrapping key. The relay keeps its SHA-256 on the session. A password change presents its starting key's capability until its final request is acknowledged. The relay's digest is replaced on the caller's session by a password change, and cleared on every other session by a re-tune.
- **Session tokens.** The refresh token is kept in the operating system's keyring where the device has one that accepted the write. Otherwise it is kept in a column of the local account database, a file created owner-only and kept so on every open ([L-22](#) *The refresh token's protection depends on whether the keyring accepted it, and a lost device holds a live session, whatever the state of its cached key, until the master password is changed from another device, the account is reset, or, unless it was lost in the middle of a sign-in, it is retired*). Both tokens end on the device when the relay refuses the account capability ([section 5.2](#)). They end on the relay only if the relay answers the device's request to end them.

Key material in memory is held in types that are wiped on drop and print as redacted in any diagnostic output. The exceptions are stated rather than hidden:

- The master password and the twenty-four words: [L-25](#) *The master password and the kit words exist as ordinary strings, which nothing wipes.*
- The two capabilities on their way out: [L-9](#) [L-10](#).
- The auth-hash's hexadecimal form: [L-25](#).

5.6 Security considerations for the keys

Offline guessing (the curious relay operator, a person with a copy of the relay's database, a person holding a session token, the mailbox). A guess costs one 19 MiB Argon2id run per candidate password, and that is the whole cost. A session is enough to be in this position, because the relay serves any session of the account its roots' wrapped keys, their kit wraps and the recovery-kit record. So a person holding a session token, and whoever completed a mailed reset from the mailbox, holds for that account what a copy of the database holds. Whoever holds a root's wrapped key tests a guess by deriving the wrapping key and trying to open the wrap. Whoever observed the login credential at login compares the derived auth-hash directly. Whoever holds an account capability's digest, from the relay's database or from a sign-in's body, or the capability itself from a request's header, compares the derived one the same way. The capability derives from the password, so each session's digest in a copy of the relay's database is a verifier for password guesses, and a right guess yields the capability. The relay's own second hash of the login credential protects nothing against any of these. It only stops a database copy from being replayed as a login. The email is public and is the salt, so an attacker can precompute per address but not across addresses. The strength of the master password is therefore the whole of the account's strength against anyone who holds the relay's database. [L-33](#) *The email address is the salt, so precomputation per address is possible, and the relay learns the address.* [L-34](#) *The application checks a master password's length and nothing else: at least 12 characters and at most 128 bytes.* A copy of a wrap taken before the parameters are raised ([section 4](#)) can still be tested at the old cost. The address cannot be changed: a new address is a new account, with new keys.

Replay at another relay (a hostile relay). The login credential is a bearer credential, but it is made for one relay's canonical origin ([section 5.2](#)). A relay operator who has seen it cannot sign in with it at a relay of another origin that the person uses with the same password, because that relay expects the credential made for its own origin. There are two exceptions ([L-18](#)). What remains otherwise is the offline guessing above: the operator can test candidate passwords against the hash they saw.

Why the email is the salt (the design's own trade). A stored random salt would have to be synchronised to every new device before it could derive anything, and a lost salt would be a lost account. Making the email the salt means a blank device needs exactly two things a person can type, beside the relay's address. The cost is that a password change re-wraps each root key it reaches ([section 10.3](#)). The cost is also that two accounts with the same email and password on two different relays derive the same wrapping and attest keys, though not, at two different origins, the same login credential. That is accepted: the relay is not a party to the derivation.

Nothing binds a key or a seal to a relay (a hostile relay holding another relay's copy). The wrapping and attest keys are the same at every relay for one email and password, and no associated data names a relay or an account. [L-19](#) *Nothing binds a key to a relay.* A device that joins fresh adopting such a copy is the fresh device's trust in the relay's current state ([L-5](#) *A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup*), carried across relays. Binding a relay into the wraps would tie every key to an address and was not done.

Domain separation. Every HKDF label and every associated-data domain string is distinct, and the pseudorandom keys they expand from are themselves distinct (the master key, the wrapping key, a root key, a generation key, the kit seed), so no two HKDF branches yield the same key. One key still serves several purposes. The wrapping key wraps root keys, the kit seed and, on the device, generation keys, and is also the input from which the account capability is expanded. A generation key seals every kind of content object of its generation ([section 6.2](#)), and is also the input from which that generation's write capability is expanded. So in each the same 32 bytes are used as a cipher key and as an HKDF key. The two uses are of different primitives, and each capability's label is its own. The design relies on HKDF's output revealing nothing of its input; that is argued, not tested ([section 12](#)). The cipher does not commit to its key, and the design does not rely on it. What keeps a ciphertext sealed for one purpose from opening as another is the domain string and coordinate in its associated data, which would refuse the open even if two keys' bytes coincided.

The write capability (a person holding a session token, the relay, a person with a copy of the relay's database). The capability exists to answer one adversary: someone who holds a session and not the vault's key. It answers no one else, and claims nothing against them. The relay sees the capability in every request that writes, since the relay is who it is for. A hostile relay gains nothing from it, because the capability gates only the relay's own storage, which a hostile relay controls anyway, and because no device accepts anything on its strength. What a device accepts is still only what opens under the key. A copy of the relay's database holds the capability's SHA-256, from which a 256-bit random-looking value cannot be recovered. The capability is a bearer value ([L-9](#)).

The account capability (a person holding a session token, the mailbox, the holder of the login credential, the relay, a person with a copy of the relay's database). It answers the same adversary one plane up: someone who holds a session and not the password's wrapping key. What the relay's check proves is narrower than it reads ([L-10](#)). Whoever completes a mailed reset chooses the digest the reset's session is opened under, because the relay stores the digest it is sent ([section 3.2](#)). That the digest on every session is of the account's current key rests on every move of the credential that changes the key replacing or clearing the digests it leaves, or deleting their sessions ([section 10.3](#)). A hostile relay gains nothing from the capability, for the write capability's reasons: it gates only the relay's own storage, and no device accepts anything on its strength. Its label carries the relay's canonical origin, so a capability one relay is shown is good at no other, with the first of the two exceptions in [L-18](#).

The cached wrapping key (the finder). The operating system's secret store holds the wrapping key so that a launch does not prompt. [L-24](#) *A retired device's cached wrapping key is not wipeable from outside.* A device compromised while the entry is there, by someone who can read that store as the user, can open every root key the device has, and the wrapped kit seed beside them, and therefore the kit key. [L-23](#) *A lost device whose cached key is still live is one of the owner's devices to the finder who launches it.* [L-12](#) *Root keys are permanent, so there is no cryptographic revocation.* This exposure is inherent in not asking for the password every time, and is bounded by the application's expiry when it next runs, by sign-out clearing the entry, and

by the operating system's own disk encryption. Retiring the device from the account ([section 10.5](#)), which presents the account capability like every write to a device slot, stops it reaching the relay unless it was lost in the middle of a sign-in.

6. Sealing

6.1 The operation

Every object that leaves a device is sealed the same way:

```
nonce      = 24 random bytes, fresh for this seal (one exception, section 6.4)
ad         = domain || protocol-version (1) || coordinate fields
ciphertext = AEAD(key, nonce, ad, plaintext)      // tag appended
```

The nonce and the coordinate travel in the clear. One case reuses a nonce, safely ([section 6.4](#)). The key is one of the keys of [section 5](#). The plaintext is content, a key, a label or a record, depending on the domain. Opening recomputes the associated data from the coordinate the receiver *expected*, never from any field inside the record. Opening fails as one opaque error for every cause the cipher can see: wrong key, wrong nonce, altered ciphertext, truncation, or a mismatched coordinate. Distinguishing those would be an oracle, so the design does not. Three other failures exist: a record that does not decode and a record whose format version is unknown, each refused before any key is used ([section 4](#)), and a plaintext that opens but does not decode, refused after.

Whether one of them is reported apart from a failed open differs from record to record:

- An undecodable record or an unknown version is named for a root's wrapped key and a wrapped generation key.
- An undecodable record is named for the wrapped kit seed, whose unknown version is not.
- A plaintext that opens and does not decode is named for a note's operation, a page registry entry, a compaction and an image's register entry.
- For the root's kit wrap, the label, the hand-over record and the lineage certificate they return the same opaque failure as a failed open.
- A heartbeat that opens and does not decode is not heard, exactly as one that does not open.

6.2 The associated-data register

Every domain, with its coordinate, in full. Widths are bytes.

Domain	Coordinate after the version byte	Key	Seals
"slate:op:"	generation (16) page (16) actor (8) seq (8)	generation key	one merge operation

Domain	Coordinate after the version byte	Key	Seals
"slate:register:"	the same	generation key	one entry of the vault's page registry
"slate:snapshot:"	generation (16) page (16) 0 (8) 0 (8)	generation key	the text the page's history was seeded from, the same on every device
"slate:compaction:"	generation (16) page (16) SHA-256 of the cut (32)	generation key	the page's merged state up to a cut
"slate:asset-register:"	generation (16) asset (16) revision counter (8) revision actor (8)	generation key	what a binary file is: its path, hash, the hash of the version it replaces, size, and whether it is a deletion
"slate:asset-bytes:"	the same	generation key	the binary file's bytes
"slate:label:"	generation (16)	generation key	the vault's display name
"slate:dek:"	generation (16)	wrapping key	a generation's key, on the device only: in its sync cache, and in a password change's journal while the change is under way (section 5.5)
"slate:root:"	root (16)	wrapping key	a root key
"slate:recovery-root:"	root (16)	kit key	a root key, second wrap
"slate:recovery-seed:"	kit generation (8)	wrapping key	the kit seed
"slate:handover:"	root (16) epoch (8)	hand-over key	a hand-over record
"slate:lineage-root:"	root (16), the retired root	the successor's lineage key	a lineage certificate on a retired root's link to its successor
"slate:lineage-vault:"	generation (16), the superseded one	the successor's lineage key	a lineage certificate on a superseded generation's link to the vault that replaced it
"slate:attest:"	actor (8)	attest key	a device heartbeat

The field named *generation* is the identifier of [section 5.3](#), which the code calls the vault id. No content coordinate names the root. Two generations of one root are kept apart twice: by their different derived keys, and by the generation identifier in every content coordinate. The version byte after every domain is the sealing protocol version, currently 2. Every field is fixed-width, so no two distinct coordinates can serialise to the same bytes, and the domain string comes first, so two objects of different kinds differ before any coordinate byte is compared.

A snapshot and a compaction are not stream objects. Each is one slot per page, overwritten when a newer one is written. The snapshot is sealed at the reserved coordinate whose actor and seq are both zero, which no device can register as. The compaction is sealed under a domain and a coordinate of its own. Each is authenticated by its seal alone, and the Merkle forest of [section 8.1](#) does not cover either. A compaction travels with its **cut**: for each device, the position in that device's stream up to which the compaction includes every operation. The cut travels in the clear, because the relay needs it to decide which history it may discard. It is bound into the compaction's seal: the associated data ends in a SHA-256 of the cut in one canonical form. In that form the cut's entries are in ascending order of actor, each as `actor (8) || seq (8)`, and entries at position zero are left out. A device opens a compaction under the cut the relay served beside it. So a compaction served under any cut but the one it was sealed with does not open, and the device moves nothing on its strength: no frontier, no stored history, no checkpoint. Both places that act on a served compaction open it this way: the device that adopts one and the device that merges one with its own. The application's sealing interface has no way to seal a compaction without its cut. A compaction that does not open is a failure of its page ([section 6.4](#)).

What is deliberately *absent* from every coordinate: a path, a file name, a label, a hash of content. Those are content and live only inside a seal. A vault, page, asset or device identifier is a random value with no relation to what it names.

6.3 Record formats

A wrapped key, the wrapped kit seed, a sealed label, a hand-over record and a lineage certificate are each a small versioned structure of three fields, `version (1) || nonce (24) || ciphertext`. Each is encoded with a compact binary serialiser (postcard), which writes the ciphertext's length before it. The tuple sealed inside a hand-over record or a lineage certificate is encoded the same way, so its integers are variable-width; the associated data, which is what is authenticated, keeps the fixed-width layout of [section 6.2](#). The relay refuses a lineage certificate longer than 2,048 bytes, and a device refuses one whose chain is empty, longer than 64 roots, or names a root twice. A note's operation and a page registry entry each ride an envelope, `(vault, page, actor, seq, kind, nonce, ciphertext)`, which is the unit the relay stores; the relay checks the envelope's account, vault, page, size and sequence and never looks inside. The relay binds the device an envelope names to the session that sends it, as it does the device named in an image's register entry and in an image's upload: one that names a device other than the one the session is bound to is refused. A compaction's cut names other devices' positions and is not bound this way ([section 6.2](#)). A snapshot or a compaction travels as the nonce and ciphertext of its page's slot, a compaction with its cut beside them.

6.4 Security considerations for sealing

Transplanting (a hostile relay). The relay's storage key is the relay's, and a hostile relay can serve any bytes at any coordinate. Binding the coordinate into the associated data means the receiver's expectation is what is authenticated: bytes served at a coordinate other than the one they were sealed at do not open. That covers another page, another vault, another generation,

another asset and an older revision of the same asset. It also covers every cross-kind case, including a file's bytes served where its register entry was asked for, and a generation key served where a root key was asked for.

Nonce reuse. Nonces are 24 bytes drawn from the operating system's generator for every seal, with the one exception below. At that size the birthday bound for a repeat under one key is far beyond any plausible volume, and no counter or state is needed. Nonces are never derived from content. The exception: an image upload resumed after an interruption seals the image again under the nonce it drew the first time, at the same coordinate and under the same key. It does so only when the image's content hash shows its bytes are unchanged. The two ciphertexts are then identical, so the repeat reveals nothing.

The oracle. One opaque failure for every cause the cipher can see, at the cryptographic boundary and at the user-facing one, where a root's wrapped key that does not open under the typed password reads as "wrong master password" and nothing more.

What a failed open does. It differs by plane, and the difference is stated because [section 2](#) could be read as promising one behaviour. An operation on a note's page that fails to open, or that differs from what the device already holds at that coordinate, is a hard error for that page. The page stops syncing and shows as needing attention, and the vault's other pages go on ([L-39 A failed open or a divergence on a note's page raises no alert](#)). A page's compaction that fails to open, which includes one served under a cut it was not sealed with, fails that page the same way. The device takes nothing from it, and keeps the operations, the checkpoint and the file it has. The vault does not read as synced while the page is failed. An entry of the page registry that fails to open, beside others that do, ends the vault's sync round with an error. There is one exception: where the relay cut a round's list of registry changes short and the device fetches the rest in the same round, a failure in that rest is logged. In that case the round goes on with what it had already read, and those entries wait for a later round. A heartbeat that fails to open is treated by the device that reads it as not heard, which reads there as staleness. On the device whose slot it is, a sibling that read the slot after the publish and did not hear it is the evidence the hidden-heartbeat check acts on, and raises the alert ([section 8.3](#)). An image's register entry that fails to open is logged, dropped, and skipped over, and the device carries on. A lineage certificate that fails to open, or opens and does not qualify, is treated as no certificate: the device adopts nothing and the vault waits on the person's decision ([section 7.3](#)). Only the note streams' failures are the hard error the principle in [section 2](#) describes. One more case opens and is still refused: a page registry entry whose clock lies outside the range every device accepts is stored, has no effect, and is reported on that device as a sync failure. What a device holding the vault's key can do with that is stated at [L-7 A device that holds the vault's key can stop an honest device's changes reaching anyone](#).

Replay. On a note's stream and on the image plane, a sealed object can be re-served at its own coordinate, and that is not an attack, because the coordinate names one thing that never changes. Other coordinates name a slot that is overwritten: a page's snapshot and compaction, the vault's label, a root's two wraps, the wrapped kit seed, and each device's heartbeat. There, serving an older object is a rollback, which can be any of these:

- an old name for the vault ([L-55 A vault's name can be served old](#));

- a wrap under an old password or an old kit ([L-29 Kit replacement and password change](#) replace what an honest relay serves; they cannot un-publish bytes a dishonest one kept);
- an older heartbeat, heard as fresh for up to a day ([section 8.3](#));
- an old compaction under its own old cut.

That last one opens, because it is authentic. A device whose own checkpoint for the page the old cut does not cover refuses it. Any other device takes it and applies over it every operation it holds, so nothing the device had is lost, and what the relay withheld by serving it reads as lag. The wrapped kit seed binds its generation into its associated data against exactly that ([section 10.6](#)), within the limit stated at [L-5 A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup](#). An old snapshot is the one harmless case, because a page's seed text never changes and every device derives the same seed text for a page. Freshness, where it matters, is a separate mechanism ([section 8](#)).

Length. Ciphertext length equals plaintext length plus the tag, and nothing is compressed. What the relay learns from lengths is recorded as limitations, not hidden: [L-30](#) and [L-31](#).

7. Vault identity and lineage

7.1 Coordinates are immutable

A note's operation, or an entry of the page registry, lives at `(vault, page, actor, seq)`. The actor is a device's stable random identifier, and `seq` counts that device's objects for that page, gap-free from 1. A page's snapshot and compaction slots sit apart from the streams ([section 6.2](#)). A device that has seen a given object at a coordinate will treat different bytes at that coordinate as a forged history and stop syncing that page, hard. So an honest device must never write two different things at one coordinate, which forces the rule below.

7.2 Restarts mint fresh identity

When a history has to restart, because a device is recovering into a vault whose relay copy it cannot reconcile with, or because the person chose to start over, the restarted history is written under a **fresh generation identifier**. In a hand-over or a retire, described below, the old generation is purged on the relay and **tombstoned**: its owner's requests for it answer "gone" for ever, and the identifier can never be claimed again. After the reset of [section 10.4](#) the old generation is kept instead, pointed at the vault that replaced it, until it is deleted deliberately. Either way a device that returns later holding the old generation gets a clean signal that the lineage was replaced, and rejoins onto the new one, rather than a fork error.

A restart that can open the root key keeps it, unless it is the deliberate "start a new vault identity" action (below). It mints the fresh generation, derives that generation's key from the root, and hands the root's pointer over in one relay transaction. That transaction is a compare-and-set from the parent generation to the new one. It tombstones the parent and claims the successor

together, and fixes the new generation's write capability from the digest the request carries ([section 5.3](#)). That compare-and-set, and the epoch of [section 7.3](#), each keep a root to at most one live generation at every instant ([section 12](#)). No key material is published by a restart at all.

A restart mints a *new root* only where it cannot, or should not, go on with the old root key, and it does so by one of two routes.

The deliberate "start a new vault identity" action, and the replacing of the synced copy of a vault whose root key the device cannot open with the device's own, take a **retire** followed by the publishing of a successor ([L-68 Starting a new vault identity on an account at its limit of vaults does not finish](#)):

1. The retire clears the root's pointer and bumps its epoch ([section 7.3](#)), naming no successor.
2. The successor names what it replaces when it is published. That one request also founds the successor's first generation. It carries the generation and its write capability. The relay, in one transaction, links the old root to the new one, stores the new root and makes the generation's record under it, or does none of the three. Every refusal comes before the link is made, because a link is made once. So the new root is never visible on the relay without its first generation. A retry whose answer was lost reads its own link back as success. Two devices racing to do this cannot both win: the loser reads the winner's link, before any limit on the account's roots or vaults is asked, and rejoins the winner.

Were the new root visible on the relay without its first generation, whoever held the account capability with a session could read the new root's identifier from the old root and found a generation under it with a capability no device holds. That would lock every device out of the new identity for good. A session alone could not, since every request that founds a generation presents the account capability ([section 5.3](#)). That closes one road to that end, not every road: [L-16 Whoever holds the account capability with a session can spend a retired root's one successor link](#).

The reset of a vault's relay copy from the files takes the other route ([L-69 An interrupted vault reset cannot finish once another device's reset of the same vault has won](#)). A device offers that reset only where nothing it holds for the vault opens under its wrapping key, as after a forgotten password on an account with no usable recovery kit ([section 10.4](#)).

1. It publishes the new root and its first generation beside the old generation. A vault's first enable, and that reset, still publish the new root and bind its first generation in two requests. Both present the account capability, so a session alone can neither publish such a root nor found a generation under it between the two ([section 12](#)). Nothing on the relay names such a root to any other session before its generation exists, since it replaces nothing by a link. So the window above does not open there for anyone but whoever read the publish itself in transit, who holds the root's identifier, the account capability and a session together ([L-8](#), [L-10](#)); that is argued, not tested ([section 12](#)).
2. Then it **supersedes** the old generation by pointing it at the new vault, and deletes nothing ([section 10.4](#)). Both pointers, the successor link and the supersede, carry a lineage certificate ([section 7.3](#)).

7.3 The sealed hand-over record and the epoch

The relay holds no key, and must not be given a verifier of anything a device trusts, because a symmetric verifier is a forger. The two verifiers it does hold, the digest of a write capability (section 5.3) and the digest of the account capability on each session (section 5.2), verify nothing a device relies on. So the relay verifies nothing about a hand-over beyond the account capability, which it checks before anything it holds for the account, and the compare-and-set. What tells a *peer* that a transition was authored by someone who could open the vault, rather than invented by the relay, is a **hand-over record**. The record is the tuple (root, parent generation, new generation, epoch, key source) sealed under HKDF(K_R, "slate:kdf:v1:handover"), with the root and the epoch in its associated data. The relay stores it opaquely beside the clear-text pointer and serves both. A peer opens the record, compares the two halves field for field, and refuses any mismatch.

The **epoch** is a per-root counter the relay keeps monotone and every device that has committed one keeps durably. A hand-over posts `stored + 1`. A device refuses a record sealed below its own committed epoch, and accepts one at the same epoch only when it names the transition that device committed itself. That is what still lets it re-read its own hand-over, while refusing the other half of a race it did not win. That is the anti-rollback property in one line: a relay restored from a backup cannot walk a device backwards or sideways, because the epoch is inside the seal's associated data, not merely beside it in a column.

Every lineage change is written on the device as a durable **transition marker** before the request it describes, carrying everything needed to replay the request byte-for-byte. If the request is interrupted, it is replayed byte-for-byte from the marker. The relay reads the same tuple with different bytes as a different operation, so a resume that re-sealed would lose its own compare-and-set. The marker is cleared only when the device's two local stores agree.

A hand-over record cannot vouch for a new identity, whose successor is a different root with a different key; a **lineage certificate** does. Every pointer that changes a vault's identity carries one: a retired root's link to its successor, and a superseded generation's link to the vault that replaced it.

The certificate is the tuple (predecessor, successor, chain) sealed under HKDF(K_R(successor), "slate:kdf:v1:lineage"). Its associated data is the predecessor, a root identifier or a generation identifier, under a domain string of its own for each kind. `chain` is the ordered list of roots the successor descends from, ending in the successor. So a device still on A of A -> B -> C can check the step to C without opening B, whose key a second forgotten-password reset may have put beyond it. That holds only where the device that minted C knew B's ancestry, and a device that joined the vault fresh at B never held A. It learns the ancestry from the relay, and checks it. With a root's record the relay serves the predecessor whose link names that root, and that link's certificate. At a fresh join, or when it holds a generation's key and not yet its root, a device that adopts a root it holds no chain for verifies that certificate under the lineage key of the root it has just opened. It verifies, under the same key, the certificate on any supersede pointer onto the generation it adopts. A certificate that opens must name the adopted root as its successor and the served predecessor as its last ancestor, and two that open must state the same chain, or the adoption is refused. The device records the chain no later than the

root it belongs to, so a certificate it later mints for **C** lists **A**, **B** and **C**. A certificate that does not open, or that opens and names another successor, is treated as one the relay withheld. At these two adoptions a certificate decides only which chain is recorded, never where the device goes, so skipping it follows no pointer. What follows when a relay withholds the certificate is stated at [L-67](#) *A device can be left unable to follow its vault to a new identity.*

The device that mints the successor seals the certificate once, carries it in its transition marker, and replays the same bytes if interrupted. The relay requires the account capability and a certificate on every retire, successor link and supersede before it changes anything. It stores the certificate beside the pointer and serves it back. It holds no root key, so it checks only the certificate's length. A device requires the certificate before it follows a successor link or a supersede to another vault. A peer checks it once, where it would adopt the successor: the certificate must open under the lineage key of the successor root the peer has just opened with its own wrapping key, name that root as its last entry, and list the root the peer holds strictly before it. Where there is no certificate, or it does not open or does not qualify, or the device holds no root, the device adopts nothing: the vault waits on the person's decision, and no file is touched.

7.4 Security considerations for lineage

Coordinate reuse (a hostile relay). Refused at every level: a page restart gets a fresh page identifier, a vault restart a fresh generation, and a device that sees two byte sequences at one coordinate stops.

An invented hand-over (a hostile relay). The clear pointer can be forged; the sealed record cannot, and a peer adopts nothing until the two agree.

Rollback (a hostile relay restoring a backup). Blocked by the epoch for every device that holds one. A device keeps its committed epoch in its local account store and in its sync cache, which refuses to hold a root without one, so the fence survives losing either store. For a fresh device, which has none, see [L-5](#) *A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup.* The associated data binds the root and the epoch and not the generations the record names, so a relay that kept the losing record of a raced hand-over holds two authentic records at one epoch. It cannot make the winner adopt the loser's: the winner refuses a different generation at an epoch it has already committed. Nor can it hand the losing record back to the device that sealed it: that device records what it lost before it joins the winner, and refuses it if it is ever served back. What such a relay can still do is stated at [L-6](#) *When two devices race to hand a vault over or give it a new identity, a relay that kept both records can split other devices between two generations, with no alarm in band.*

A false retirement or a redirected successor (a hostile relay, whoever holds the account capability with a session). A hostile relay can tell a device that its root was retired. Anyone holding a session and the account capability can retire a root, since the relay can check no more than a certificate's length. A session alone cannot, because a retire presents the capability. The devices on that root then stop syncing the vault and wait for a successor, which is destruction of the kind stated at [L-14](#) *A session alone can read, end things, and hold things; whoever holds the*

account capability with a session can destroy the relay's copy of a vault whole. Such a caller can go one step further and publish a root of its own as the retired root's successor ([L-16](#)). Neither can point a device at another vault of the account: a successor is followed only under a lineage certificate sealed with the successor's root key, and neither holds a root key. What the certificate proves is that someone who holds the account's keys authored the succession. Its limit is stated at [L-4](#) *A lineage certificate proves authorship by someone who holds the account's keys, not by a particular device.* Two devices that race to give one vault a new identity are covered by [L-6](#).

Concurrent transitions (a hostile relay). The compare-and-set that serialises hand-overs is the relay's. A hostile relay can acknowledge two competing hand-overs from one parent, one to each of two devices, which is the split view of [section 8.2](#) applied to lineage rather than a new class. The relay also enforces a set of non-cryptographic rules on a hand-over, such as the successor being unclaimed and the record being of bounded size. Each rule is a refusal a hostile relay can apply selectively to one device without forging anything.

Destruction on the strength of a session and the account capability (whoever holds the account capability with a session). A hand-over purges the parent generation inside the relay's transaction, and a retire purges the generation the root named and clears the pointer, on the authority of a session and the account capability. A session alone is refused ([section 5.2](#)). A retire's certificate is checked only for its length. A hand-over made by whoever holds the account capability with a session and does not hold the root key also fixes, for the generation it hands over to, a write capability of the caller's choosing. No device follows it, because the caller cannot seal the hand-over record ([section 7.3](#)). What it achieves is the loss of the relay's copy, and a vault that stops syncing in front of the person. Who can still do it is stated at [L-14](#).

Loss of the parent's key. After a hand-over the device drops the parent generation's derived key; the root key stays. Nothing about a restart changes what the relay could decrypt, which is nothing.

8. Multi-device consistency

8.1 Append-only streams and the Merkle forest

Each device's objects for each page form a stream numbered from 1 with no gaps. Beside the per-page version vectors that say how much of each stream a device has, every device keeps one append-only binary Merkle tree per actor whose leaves hash 64-object ranges of the sealed envelope bytes. The per-actor roots are a hash-compressed version vector. Two honest devices holding the same objects compute identical roots whatever order the objects arrived in and whatever the relay asserts, because the address space the tree is built over is the authenticated one. Where each tree starts is authenticated too: once a device adopts a page's compaction, its trees for that page start from the compaction's cut, and a compaction opens only under the cut it was sealed with ([section 6.2](#)). Hashes are over ciphertext, so the relay may keep the same forest to answer catch-up queries without learning anything it does not already store.

The forest covers the objects that ride envelopes: a note's operations and the page registry's entries. It does not cover a page's snapshot and compaction slots, the asset plane, sealed labels, wrapped keys or heartbeats, each of which is authenticated by its seal alone. A divergence between a device's forest and what the relay serves localises to a 64-object range, and reads as "fork, corruption or omission". On a note's page, it is a hard error for that page, which stops syncing and shows as needing attention while the vault's other pages go on ([L-39](#) *A failed open or a divergence on a note's page raises no alert*). Omission of a *suffix* reads as stale, never wrong ([L-53](#) *Withholding and old views of notes read as lag*), because the registry stream that names pages is itself append-only and a missing tail is indistinguishable from lag.

8.2 The remaining gap: a relay that lies to different devices differently

Everything above makes each device's view internally consistent. What none of it can detect on its own is a relay that serves device A one consistent prefix of the honest history and device B a different consistent prefix, for ever. Because forging is impossible, such a fork is purely omission-shaped. This is the known limit of any design where clients share only a symmetric key and the server is the only channel between them; the published result is that freshness requires an out-of-band comparison.

8.3 Sealed heartbeats and the two-screen check

Slate's out-of-band channel is the owner. Each device publishes one small **heartbeat** to a per-device overwrite slot on the relay, on sync rounds where what it attests has moved and at least every six hours while unlocked. The relay takes a write of that slot only with the account capability, which the device is handed for each publish. A device whose publish is refused is signed out, as [section 5.2](#) says, and publishes nothing more until a sign-in. A heartbeat is sealed under the account's attest key, with the publishing device's actor in the associated data. Every signed-in device reads every sibling's heartbeat; one that does not open is treated as not heard. A heartbeat carries the device's wall-clock time. It also carries the relay's stamp on the device-list read that fed it, with the siblings whose heartbeats the device actually opened in that read and the time each of them claimed. For each vault whose latest round succeeded, it also carries:

- its per-actor frontier totals over the vault's live pages;
- the point it has reached in the vault's page registry, with a digest of the registry as folded at that point;
- a digest of its map from each live page to that page's frontier;
- whether the map was taken straight after a check that the device held exactly the relay's frontier on every live page, with the relay's change counter at that instant.

Frontier totals are chosen for three properties that carry the design.

- They are independent of how much history a device has compacted away, because frontiers are logical positions.
- They are monotone, and withholding cannot raise them, because the relay cannot mint objects, so any withholding strictly lowers what a device can attest. A compaction lifts a device's frontier only to a cut that a holder of the vault's key sealed ([section 6.2](#)).

- One field serves both directions, because my total for *your* actor is at once my claim about my frontier and my acknowledgement of how much of you I have received.

My total for my own actor is what the relay has acknowledged receiving from me, not what I wrote, so a write the relay refused is never claimed.

Freshness is the reader's comparison of the heartbeat's wall-clock claim with its own. A heartbeat is heard only while its wall-clock claim is within 24 hours of the reader's clock, either way. Outside that window it makes no claim on any check, though it still counts as a sibling heard, which is what the clearing of a sync integrity alert asks (below). A relay can therefore replay a heartbeat as fresh for up to a day, or withhold it for ever, at the cost of staleness: "last heard through sync at T". On hearing a sibling, a device runs four checks for the vault the round drove.

- **Withheld objects.** Where the sibling stands at the same point of the page registry as this device, so that both totals are sums over the same live pages, and it attests objects this device lacks, the device demands them. Where the sibling's registry is ahead of this device's, the device demands the registry's missing tail instead. A standing demand makes every following round run the full repair sweep. A demand still unmet after three completed rounds raises the **sync integrity alert**. Each round is judged against the demand that was made, not against the sibling's newest heartbeat, because an active sibling honestly moves on while the relay answers.
- **The page registry.** Where the sibling stands at the same point of the page registry, the two registry digests must be equal, because both devices folded the same entries. A difference that survives three rounds that each completed a repair pass of the registry raises the alert.
- **A split by page.** Totals are sums, and a sum loses which page each object belongs to. As a result, with three or more devices, a relay can deliver one author's objects for page A to one sibling and the same author's objects for page B to another, so that both attest the same total. This check compares page maps, but only two maps each taken straight after a passing frontier check, at the same registry point. Two such maps are readings of one relay's frontier, which only grows. So equal sums with different maps, a later reading with a smaller sum, or two readings at one relay instant with different sums cannot come from an honest relay. Three such contradictions raise the alert. The frontier check runs only while some demand stands against a sibling, and a sibling whose total equals this device's over a different map arms a demand of its own.
- **A hidden heartbeat.** The relay stamps every heartbeat it stores, and every device list it serves, from one counter of its own. A sibling that attests a device-list read stamped at or after this device's last publish, and does not list this device's current heartbeat among those it heard, was served a list without it, which contradicts the relay's own stamps. One such observation raises the alert, and it stays raised until the same sibling attests a later read in which it did hear this device's current heartbeat. The check tells one publish from another by its wall-clock time in whole seconds ([L-48](#) *The hidden-heartbeat check tells publishes apart by the second*). It also takes it that only the device writes its own slot ([L-17](#) *Whoever holds the account capability with a session can raise the sync integrity alert against an honest relay*).

A round counts toward the two demands for withheld objects only when its final attempt ran the repair sweep, drained it to the end, and returned success. Every attempt clears that record first. The device takes the record once per tick, whatever the round returned. So an attempt abandoned and retried inside the tick, or a tick in which no attempt ran, counts for nothing. The bound is therefore three **completed** rounds. Lag, compaction and a change of lineage are absorbed before the alert can fire: a round that a lineage, root or key refusal ends before its drain does not count, and neither does a round that fails. The trade is stated plainly: [L-40](#) *A relay that refuses every sync round is never accused.* A relay that refuses only some rounds is still caught, because the count carries across the rounds that do not count.

The sync integrity alert is raised on the device, and it latches. It clears after a round in which all of the following hold:

- at least one sibling's heartbeat opened;
- no demand lapsed on an expired heartbeat;
- no accusation was held back;
- no check with a question outstanding went unanswered;
- after the round, no demand of any check stands.

That is weaker than it reads. A demand for withheld objects, or for a registry that differs, no longer stands once it has raised the alert. The split-by-page and hidden-heartbeat checks keep a latch of their own, and an alert either of those raised is not cleared this way. And a sibling's heartbeat counts as heard as soon as it opens, before its freshness is read and whether or not it carries anything for this vault. So an alert raised by one of the first two checks can be cleared with the objects still withheld ([L-47](#) *A raised sync integrity alert can be cleared by a heartbeat that says nothing of the vault*). [L-51](#) *The sync integrity alert is held in memory.*

What the checks do not catch is stated in [section 13](#):

- [L-47](#).
- [L-48](#).
- [L-42](#) *A relay that serves exactly what is demanded while staying ahead is never alarmed on, and a device with a durably failed page of its own accuses nobody on the withheld-objects or split-by-page checks.*
- [L-43](#) *The split-by-page check never speaks at an author that never converges.*
- [L-44](#) *The split-by-page check waits while the page list keeps changing.*
- [L-45](#) *Rounds that end early can make the split-by-page check stop asking.*
- [L-46](#) *A heartbeat hidden by a frozen device list reads as staleness.*
- [L-49](#) *The relay can clear a raised sync integrity alert by dropping the device it accuses.*
- [L-50](#) *A password change blinds the check until the other devices return.*
- [L-41](#) *A relay that refuses the account capability signs every device out.*

The one case that stays undetectable in band is the eternal total partition ([L-54](#) *The eternal total partition is undetectable in band by construction*). The whole ritual is as follows:

1. Press sync on two devices.
2. Read both screens.

One screen saying "published and acknowledged just now" beside another saying "stale" is a contradiction the relay cannot fake.

8.4 Security considerations for consistency

Why not per-device signing keys? Signal-style identity verification defends key distribution, where a server hands Alice something claimed to be Bob's key. Slate has no key-distribution step: the shared secret reaches every device through the owner's fingers. A shared symmetric key cannot serve as a device identity, since every device could impersonate every sibling, and it does not need to, because the devices are mutually trusted and the adversary lacks the key entirely. The whole problem reduces to freshness, which symmetric cryptography cannot manufacture. The trade is recorded as [L-11](#) *With shared symmetric keys only, a compromised device while unlocked can publish anything a sibling could, and nothing distinguishes the two.*

A compromised device (out of scope, stated anyway). Every device holds the same attest key, so a heartbeat proves possession of the account's keys, not which device wrote it. A device compromised while unlocked can publish any totals as itself, raise false sync integrity alerts, or help a relay suppress real ones.

Replay of heartbeats. Allowed by construction; the associated data binds only domain, protocol version and author. A heartbeat whose claim falls outside the freshness window of [section 8.3](#) makes no claim on any check, so a device whose clock is wrong by more than a day takes no part in them, which reads as staleness.

9. Binary files

9.1 The binary-file plane

Binary files, which today means images alone (the rules that admit them are in [section 9.2](#)), sync on their own plane with their own coordinate. The coordinate is a random 16-byte **asset** identifier assigned at first sight of the file, never derived from its content, and a **revision** that orders whole-file overwrites. A revision is a per-vault counter with the writing device as tie-break. Two objects are sealed at that coordinate and only two: the file's bytes, verbatim, and a small **register entry**. A register entry carries the path, the content hash, the hash of the version the write replaces when there is one, the size, and whether the write is a deletion. The bytes and the register entry take separate domain strings, for the reason [section 6.4](#) gives.

9.2 Validation and revisions

A superseded upload is not merely refused by policy; bytes that lost the ordering race are bound to a revision no device will ever ask for again. The relay enforces the same ordering independently, refusing to store bytes whose revision the register no longer names, so the two guards agree without either trusting the other.

A decrypted register entry is not yet a trusted one. The open proves it was sealed by a device holding the vault's key; it says nothing about whether the path inside is safe to write to disk. So an entry is validated where it leaves the seal, by the same rules the local file scan applies. The rules ask for all of the following:

- an image extension from a fixed list (png, jpg, jpeg, gif, webp and svg);
- no hidden path segment;
- no staging name;
- a vault-relative spelling;
- a bounded length.

[L-35](#) *SVG is an admitted attachment format, and a peer's SVG is admitted exactly as this device's own.* A peer's entry is admitted exactly as this device's own would be.

A device never seals two different register entries at one revision while its own record survives. It draws each new revision one past the greater of everything it has seen of the register and the highest revision it has drawn. It keeps that highest revision in its account store, and raises it before the write leaves, so that it never draws one revision twice while that record survives. A device that already holds an entry at a revision compares any other entry served there by path, content hash, size and deletion. It keeps its own, and logs and drops one that differs. Where that stops is stated in [L-52](#) *Two different image entries at one revision are caught only where a device already holds one.* The relay announces an image again when its bytes are stored. So a device that took the entry before the upload finished learns that the bytes are there and fetches them: from an honest relay, an image whose upload finishes reaches every device that syncs. The relay orders an image's register entries without opening them. When it stores an entry, it discards the bytes stored for that image at every other revision. So a write of a register entry, like an upload of an image's bytes, is admitted only from a caller that presents the generation's write capability ([section 5.3](#)). A session without the vault's key can therefore neither write an entry nor displace an image's bytes.

9.3 What "synced" counts

A vault reads as synced on a device when, as of that device's last sync round, it has nothing left to send and nothing left to fetch. That means all of the following:

- no change to a note waiting to go;
- no image it wrote whose bytes it has not uploaded;
- no image in the folder it has not yet announced;
- no image on the relay that it should hold and does not.

An image kept on the device on purpose, because it is too large to sync or its path is too long, is not counted. Each device answers for its own part: while the device that wrote an image has yet to upload it, that device reads as not synced, and another device may read as synced. So, once every device of a vault reads as synced, any one of them, or the relay, can be lost without losing anything.

Three exceptions are known and stated in [section 13.5](#):

- [L-58](#) *"Synced" speaks for the last sync round.*
- [L-59](#) *An image a device can no longer upload can read as synced.*
- [L-60](#) *An image changed in place with its size and modification time kept is not noticed.*

The promise also assumes a relay that answers: a relay that withholds can leave a device reading as synced over an old view, which is the third tier of [section 1.1](#) ([L-53](#), [L-56](#)).

9.4 Security considerations for binary files

A content-addressed design was rejected because it would tell the relay which devices hold identical bytes, and when.

[L-56](#) *Images have no freshness signal.* [L-57](#) *An image edit that never finishes uploading destroys the image.*

10. Accounts, sessions and recovery

10.1 Registration and login

A relay runs in one of three registration modes: open, invitation, or closed. Registration sends the email and the auth-hash made for that relay's origin ([section 5.2](#)). Login sends the same pair, with the SHA-256 of the account capability made for that relay's origin ([section 5.2](#)), and receives a session, with the account's identifier ([section 10.7](#)). The relay stores that digest on the session it opens.

A registration is first put to the relay's mode. Two of the modes refuse there, each without regard to whether the address has an account. A closed relay admits a registration only while it holds no account and, where verification is required, no registration still waiting on its code. It refuses every other registration, for any address. An invitation-only relay admits a registration only with the address's live invitation, which it checks before it reads anything about the address's account. The registration that creates the address's account, or starts its wait for a code, spends the invitation, so a second registration with the same code is refused. A wait for a code that lapses unanswered frees the invitation it spent, unless the invitation's own seven days have passed. A registration for an address that already has an account creates nothing and spends nothing. Only whoever holds that address's invitation can see that difference.

Where address verification is required, every registration the relay admits answers the same success whether the address is new, pending or already taken. Where verification is required, the account becomes real only when the mailed code is presented. The code is entered together with the master password, and the device sends the auth-hash derived from it with the code. For a registration still waiting on its code, that auth-hash is the one the account is created with, not the one sent at registration. So the person who proves the mailbox chooses the credential, whoever registered the address first. For an account that already exists and is only waiting to have its address verified, the code verifies it and the auth-hash sent with it changes nothing.

A request whose auth-hash is missing, empty or over-long is refused before any hashing. A registration is refused the same way, before any hashing, when its address is empty and, on an invitation-only relay, when it carries no well-formed invitation. Presenting a code carries no such address check. Otherwise the relay hashes the offered auth-hash once, before it reads anything about the address, both at registration and when a code is presented. When a code is presented, every well-formed attempt, accepted or refused, pays exactly that one hash ([section 12](#)). Asking for a fresh code carries no credential, and pays one hash of a fixed dummy value instead. A registration the relay admits, and every request for a fresh code, each answer byte-for-byte the same whatever the relay knows about the address ([section 12](#)). The request for a fresh code consults no registration mode. That their timing matches is argued, not measured ([section 12](#)).

A session is two opaque random 32-byte tokens, presented as hex. The access token lives in memory only and expires after 15 minutes. The device refreshes it a little ahead of expiry over a known-good connection. The refresh token's home depends on whether the device's keyring accepts the write. Where the device has one that accepts the write, the refresh token lives in the operating system's keyring (Linux's Secret Service over D-Bus, the Keychain on macOS and iPhone). [L-37](#) *The Keychain entry on a Mac carries the platform's defaults*. Where it does not, it lives instead in a column of the device's own account store. A desktop session with no keyring service running is the ordinary case of a keyring that does not accept the write. On iPhone the application tries at each launch to move a token in the account store into the Keychain. A write the Keychain refuses leaves the token in the account store for a later launch. The account store is a file created owner-only, and never opened if it cannot be made so. The refresh token expires after 30 days and is rotated on every use. Presenting an already-rotated refresh token deletes the whole session family, on the reading that it was stolen, with one bounded exception. Inside a 60-second window after rotation, the re-presentation is treated as a lost response and answered with a freshly minted pair, because that is the benign cause. [L-21](#) *A refresh token replayed inside the grace window yields a second live branch of the session that reuse detection does not see*. A password change has a similar 180-second window for the same reason. Tokens are stored as SHA-256 digests, looked up by digest, and never logged. The relay makes no constant-time comparison of session tokens. The refresh token is kept under the account's address and the relay's origin. A session issued by a relay other than the one the device is set to use is cleared before it is presented.

Signing out asks the relay to delete the session, allowing it a few seconds, and signs the device out whether or not the relay answered. Retiring a device deletes every session bound to it ([section 10.5](#)). A master-password change deletes every session of the account but the one that made it, and replaces that session's account capability's digest. A re-tune, which keeps the password, deletes none, and clears the digest on each of the others ([section 10.3](#)). A device whose account capability the relay refuses ends its own session ([section 5.2](#)). A mailed reset deletes them all ([section 10.4](#)).

[L-32](#) *Under the default configuration a registration reveals whether an address has an account.*

A per-client-address token bucket and a per-account one bound every authentication route. Both are held in memory and reset by a restart; only the mail ceiling is durable. The mail ceiling allows at most ten messages a day to one address, and counts only these:

- the message a reset request mails, with its token and cancel code, or, while a reset's window stands, the notice that one is under way;
- an address-verification code, or, to an address that already has an account, a notice saying so;
- an operator's invitation.

The alerts that report a change to the account are sent outside the ceiling, whatever the count:

- that a mailed reset has completed;
- that a reset was undone;
- that the master password was changed or its protection upgraded;
- that the recovery material changed ([section 10.6](#)).

Each alert is written in the same transaction as the change it reports, so that whoever spends the day's allowance first cannot suppress it. The alert that a mailed reset has completed is written inside the transaction that moves the credential, and goes to the same address as the reset request's own message, which the ceiling counts.

The requests that prove the password over a live session are declaring a password change, completing one, moving the credential to a relay's new address, and deleting the account. They draw on a limit of their own, kept per session under a looser ceiling per account, so that one session's wrong proofs cannot use up what another session needs. A copied token, being its device's session, shares that device's limit ([L-14 A session alone can read, end things, and hold things; whoever holds the account capability with a session can destroy the relay's copy of a vault whole](#)).

A vault is bound to an account by the request that makes the relay's record of it: the binding of a generation to its root, a hand-over, or the publish that links a new identity, each under the account capability. The account that made the record owns it, and every later request for it is checked against that owner.

10.2 Transport

Devices talk to a relay over TLS 1.2 or 1.3, through a pure-Rust TLS implementation validating against a bundled copy of Mozilla's root store. There is no certificate pinning. The application refuses a relay address that starts with `http://`, with one exception: an address whose host is a literal loopback address, which is a relay on the device itself ([L-20 A relay on the device itself is dialled without TLS](#)). A literal loopback address is any of these: an IPv4 address in the loopback range (`127/8`), the IPv6 loopback address `::1`, or the IPv4-mapped form of an address in that range. The name `localhost` is not such an address, and neither is any other name: a host is read as a literal address or not at all. The transport enforces the same rule on every request, whatever address was saved, so a request to any other host goes out over TLS or not at all. It follows no redirect, because a redirect would carry the session's tokens and both capabilities to wherever it pointed. A request to a loopback address takes no proxy from the environment, so it does not leave the device by that road. An `http://` address already saved for a host on the network turns

sync off at launch, with the reason, and signs the session out. The rule is the application's: a relay may itself listen on plain HTTP behind a proxy that terminates TLS, as the hosted relay does.

TLS protects the session tokens, the login credential, the two capabilities and the metadata from a network attacker. A break of TLS would expose them ([L-38](#) *There is no post-quantum consideration of the design's own cryptography, because it uses no public-key cryptography*). From the relay itself it protects nothing that this document's other sections do not already protect, because the relay terminates it. The relay stamps its API version on every reply, and the device reads that stamp before it uses a reply. The device refuses a relay whose version it does not read, with a message that says which side to update. That is a guard against a mismatched relay, not against a hostile one, which can stamp what it likes.

10.3 Changing the master password

1. Before it declares the change, the device settles the kit with the relay under the current password ([L-65](#) *A kit replaced on another device just as a password change starts makes the change fail and, unless this device signs out, leaves it declared for up to 24 hours*). A device holding no kit fetches the account's, and keeps it only if it opens. One whose own kit never reached the relay publishes it, or, where another device's kit has taken the account's slot meanwhile, adopts that one once it opens and re-wraps it with the change. Where the adopted kit's words differ from the device's own, the device shows them once the change has finished. One that holds the kit compares it with the relay's, and adopts a newer one only if it opens.
2. The device then opens the kit under the current password, so a mistyped current password is refused before anything is declared.
3. In no stated order among themselves, the device does each of these; of them, only the declaration is stated to follow steps 1 and 2:
 - It declares the change. Declaring a change needs the password as well as a session: the declaration carries the current auth-hash, and the relay verifies it as it verifies the one that completes a change. A wrong one is refused, declares nothing and moves no epoch. The change is declared in flight on the relay for up to 24 hours, and a second change is refused while one is. The relay has no verb to cancel the declaration, but ending the session that made it frees it. The application ends its session on the relay when the person signs out or starts sync over. Because a declaration needs the password, a session alone cannot declare a change, and cannot hold the owner's own off with one. Nor can a session alone delete the account or change its password: each needs a proof derived from the password itself, which a session token does not carry.
 - It derives the new key triple, and re-wraps the root keys it reaches (below) under the new wrapping key, on the device and on the relay.
 - It re-wraps the kit seed on the device.
4. Only after the device has re-wrapped the root keys does it move the login credential. The kit seed, re-wrapped beforehand, travels in the request that moves the credential, and the relay stores both in one step or neither. Until this final request is acknowledged, each of the

device's writes to the account's keys, the kit's settling among them, presents the account capability of the wrapping key it started with. The final request carries the digest of the new key's capability, which the relay stores on the caller's session in the transaction that moves the credential. This order is what makes an interruption safe: at every point each root the change re-wraps opens under either the old password or the new one. Until the change finishes, though, some roots may need the one and some the other. So an interrupted change never leaves a root it re-wraps, or the kit seed, wrapped under a password that no longer works. In the transaction that moves the credential, the relay deletes every other session of the account, so every other device is signed out and must sign in with the new password; and it mails the owner.

5. The device treats the change as finished only when the relay serves back the kit seed the change carried, which is the relay's own sign that the credential moved.

The one change that signs no one out keeps the password and moves the credential onto a newer Argon2id parameter set ([section 4](#)). The relay cannot check that the password was kept, so it mails the owner then as well. That change, a re-tune, keeps every other session but clears the account capability's digest on each, because the key moved. Each other device is refused at its next write to the account's keys, its heartbeat at the latest, signs itself out, and asks for the master password once. A password change started on such a device declares itself before that refusal comes. No release yet ships a second parameter set, so no device meets a re-tune.

An account-wide **key epoch**, declared by every write of wrapped material, moves once per credential change. A device that still holds a superseded wrapping key has its wrap writes refused rather than accepted. The refusal names the current epoch, but the device persists an epoch only from a reply that proves it holds the current password, never from a refusal. Otherwise a stale device could turn the fence's own refusal into an accepted write. A change abandoned after it is declared, on a device that does not sign out, refuses every other session's wrap write until the declaration expires ([L-66 A master-password change abandoned part-way has rough edges](#)).

A session alone cannot stop the change through the recovery-kit record either, whose write presents the account capability ([section 5.4](#)). What a session alone can still do to a change is in [L-14](#). A mailed reset ([section 10.4](#)) makes no declaration and is not fenced by one. Notes are not re-encrypted: the root keys are unchanged, only their wraps move. The relay keeps no separate wrap of a generation's key, so what it serves under the wrapping key is the roots' wrapped keys and the kit seed.

What a change reaches is all of the following:

- the kit seed;
- the root of each vault this device has registered and for which this device already has a vault identity, when that root opens under the old wrapping key;
- every root that the relay's list of the account's vaults names.

Each listed root is tried under the old wrapping key, then under the new one, which means the change has already moved it, then under the recovery kit. Through the recovery kit's key, the change can re-wrap under the new wrapping key a root that a change interrupted on another

device left under a password the account never took. What it does not reach, and a listed root none of the three opens, keeps the wrap it had, and an honest relay goes on serving that wrap. The list names only vault generations that have not been tombstoned. So a root none of whose generations is live, a retired root or one whose live generation was purged, is not reached through it. Whoever holds the account's email, the password such a wrap was made under and a copy of the wrap can open that root key.

Other devices learn of a change by being signed out. When one signs in again, with the new password, it derives the new wrapping key. For each vault whose cached wrap does not open under it, it fetches the root's wrapped key from the relay and adopts it if it opens there. That is the same shape whether the account's password was changed on another device or reset there with the kit. The device does not tell the two apart: what it checks is that the root opens under the wrapping key of the password the relay has just accepted. A device that cannot adopt that way, because the relay does not answer or its root does not open either, keeps what it had and tries again at its next unlock.

Only a vault for which nothing the device has cached, neither its root's wrapped key nor its generation-key wrap, opens under the wrapping key it holds is offered a reset of its relay copy from the files ([section 10.4](#)). That judgement is the device's own and asks nothing of the relay. The prompt that says the password was changed elsewhere asks for the new password. An unlock or a sign-in that adopts the relay's wrap ends it. It is raised whenever a device's cached wrap opens under the key it holds and the relay's decodes but does not. That no longer follows a change that completes, since that signs the device out. The prompt follows each of these:

- a change in flight or abandoned on another device, which re-wraps roots on the relay before the credential moves ([L-66](#));
- a well-formed overwrite of a root's wrapped key by whoever holds the account capability with a session;
- a relay serving an old wrap;
- a re-tune, after which it lasts at most until the device's next write to the account's keys, its heartbeat at the latest, where the relay refuses it and the device signs itself out (above).

The same words have a second road: a wrap write refused on a stale key epoch says that the password was changed on another device ([L-66](#)).

10.4 Forgotten password: the mailed reset

The reset is the standard mailed flow over Slate's encrypted data model.

1. A request answers with the same body for a known and an unknown address, at a timing that differs by the millisecond-scale database work the known arm does. For a known one, it mails a single-use token valid for one hour together with a cancel code.
2. Presenting the token with a new login credential sets the new credential **immediately**, bumps the key epoch, deletes every session, records the previous credential, and mails an alert. The alert names the original message, which carries the cancel code. The alert itself does not carry the cancel code.

3. After that, in no stated order:

- The same request carries the digest of an account capability, in the application the one the new password's wrapping key derives. The relay opens the reset's session under it as sent ([section 3.2](#)), in a transaction after the one that moved the credential. A failure between the two leaves no session, and the new password signs in.
- What the new password can open depends on the kit. **With the kit**, the device derives the kit key, opens each root's kit wrap, re-wraps the root under the new wrapping key, and re-wraps the seed. Nothing about the vaults changes; the words were the second route to the same keys. A root the kit cannot open is counted and left byte-for-byte. **Without a usable kit**, nothing on the relay can be opened: the old wraps stay sealed under a wrapping key nobody holds. A fresh kit is minted, and each vault starts afresh under a new root beside the old generation, which is kept and pointed at its successor under a lineage certificate ([section 7.2](#); [L-73](#) *A forgotten password on an account with no kit starts every vault's relay copy afresh*). The files on disk were never encrypted, so nothing is lost; only the relay's copy is superseded.
- For 24 hours the previous password's login credential, or the cancel code, **undoes** the reset. The undo restores the credential and every root's wrapped key from their snapshots, and it restores the kit seed and its generation along with the credential. It bumps the epoch again, deletes every session again, and purges any vault generation or root created after the reset, inside the window. It mails the owner, naming which of the two proofs was used. It undoes the account's credential and keys, not its content. A reset made within seven days of the account's last undo is closed to the previous credential: only its cancel code undoes it, for the whole of its window. Inside that window the relay also refuses to delete anything the undo would need.

A session signed in since the reset writes nothing into a vault that already existed unless its holder can derive that vault's key, because every such write needs the write capability ([section 5.3](#)). A mailed reset without the kit opens no existing vault, so its session writes into none. What a session that does hold the key wrote meanwhile stays. A reset made with the kit opens every root the kit wraps, and the note operations, page registry entries, snapshots, labels and image entries it then writes to those vaults are not undone.

Inside the 24-hour window the relay refuses the reset's session these: a retire, a hand-over, the deletion of a vault, the purge of a page, moving the credential to a relay's new address, and deleting the account. Inside that window the relay also stores no compaction, answering one as already superseded rather than with the window's refusal. It admits the rest. Pointing a superseded generation at another vault is admitted and marked as written in the window, and the undo clears it. The writes of a root's wrapped key and of the recovery-kit record are admitted, and the undo restores what they replaced. Roots and generations the reset's session publishes are purged by the undo. Registering a device, re-enrolling an existing one, writing a heartbeat slot and removing a device are admitted as well, and the undo restores none of them. A device the session removed stays removed, with its frontier reports, its heartbeat slot and its unfinished uploads deleted. A heartbeat the session wrote into a device's slot, after re-enrolling that device on its own session, stays there until the device publishes again. So whoever holds the reset's session can remove the owner's devices and fill the account's 32 places with devices of its own.

After the undo each removed device is refused when it signs in and registers again, and the application, which drops a sign-in whose registration is refused, offers no way to remove one. After the window every write is admitted, a retire and a hand-over included, and nothing is undone. That holder, who holds the credential they set, can then delete the account.

10.5 A lost device

A replacement device signs in with the email and the master password, opens each root key under the re-derived wrapping key, and rebuilds every vault from the relay's copy. The owner then **retires** the lost device from the account. Retirement deletes every session bound to the device, so that it can no longer reach the relay unless it was lost in the middle of a sign-in (below). It also removes the device from the bookkeeping that decides which old history the relay may discard. Retirement is an explicit, owner-driven action; there is no automatic retirement.

What helps after a device is lost is four steps, in this order.

1. Retire the lost device first, from another device. Removing a device presents the account capability, so an application that holds no wrapping key, locked or with its cached key expired, asks for the master password first and tries once more after the unlock. A removal still refused then is not asked for again. The retirement deletes every session bound to the device, a replayed refresh token's branch among them. The finder cannot open another: a sign-in needs the login credential, which the device does not keep, and a session bound to one device cannot register a second. A device lost in the middle of a sign-in, before it registered, keeps a session bound to no device, which the retirement does not delete and a completed password change or a mailed reset does. Retirement comes before the password change, because the cached key derives the account capability. With that key the finder can write a recovery-kit record the password does not open, which makes the change refuse, and can end a change in flight by removing the device that runs it. The finder can race the retirement the same way, by removing the owner's device first; but a retirement is one request, where a change is many, and needs no kit record. A mailed reset, which needs no session, ends every session whatever the finder does.
2. Then change the master password ([section 10.3](#)): until it changes, the finder's cached wrapping key is the account's wrapping key, and opens any new root and any new kit seed that a copy of the relay's database holds. Where the finder has already written a recovery-kit record the password does not open, the change is refused until a mailed reset repairs it ([L-15 Whoever holds the account capability with a session can, with one write, stop every step of the remedy after a lost device but the retirement, the password change among them](#)).
3. Then replace the kit ([section 10.6](#)), because the finder holds the wrapped seed too, and a new root is wrapped under whatever kit the account has when it is published.
4. Then start a new vault identity ([section 7.2](#)) for each vault, which mints a new root ([L-12 Root keys are permanent, so there is no cryptographic revocation](#)).

10.6 Replacing the recovery kit

A kit can be replaced.

1. A fresh seed reaches the account's one relay slot before anything local moves. The relay takes the new record only one generation above the one it holds ([section 5.4](#)); a replacement and a password change post this device's own generation plus one, and only a reset posts the relay's generation plus one. On a device whose generation is above the relay's, a kit replacement is refused until that device resets ([L-64 A kit record that moves backwards on the relay leaves a device ahead of it](#)).
2. Every root the device can reach is re-wrapped under the new kit key.

A recovery-kit record the current password does not open fails a kit replacement ([L-15](#)).

The seed slot's own monotone generation, bound into its associated data, means a relay cannot serve a replaced kit's seed as the current one to a device whose own kit record is newer. A device refuses a generation below its own record's. At its own record's generation, it refuses a seed other than its own, where it published its own record and that record still opens under its wrapping key. It takes a newer record that opens under its wrapping key, a replaced kit's among them.

Whenever the relay takes a record written with the account capability that replaces the one it holds with different bytes, it mails the owner that the account's recovery material changed, outside the daily mail ceiling ([section 10.1](#)). The record a password change's final request carries is covered by that change's own mail. A reset's undo or cancel, which puts back the record the reset replaced, is covered by the mail saying the reset was undone. A record written into an empty slot, and a retry of the record held, mail no one.

The word the interface uses is **replaced**, deliberately: replacement changes what an honest relay serves from now on. [L-29 Kit replacement and password change replace what an honest relay serves; they cannot un-publish bytes a dishonest one kept](#). A root this device cannot open keeps its old-kit wrap until a device that can open it runs the sweep, and a retired root is not reached at all. [L-5 A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup](#). [L-72 An interrupted kit replacement leaves two live kits until a later sweep converges them](#).

An account's first kit, wherever [section 5.4](#) makes it, is written on the device before it is published, the reverse of a replacement's order.

10.7 The account's identifier

The **identifier** is 16 random bytes the relay mints when it creates the account and returns with every reply that carries session tokens: a sign-in, a completed reset, a refresh. It is the relay's word for the account, not a credential and not a secret: it proves nothing, and a relay can put any value in it. The relay never logs it and never labels a metric with it. A device records the identifier from its last sign-in; signing out keeps it, and starting sync over or deleting the account clears it. It could say that a relay is not the one a device knew; in this release nothing compares it. What it is for is to let a later release notice that a relay is not the one a device's local state belongs to — another relay, a relay started over, an account deleted and made again — which an address cannot tell apart. If that release makes a device forget the account's local state when the

identifier changes, its recovery kit among it, then a relay that lies about the identifier could make a device forget its kit and accept the one the relay serves. That is what a device new to the account already does ([section 10.6](#); L-5).

10.8 Security considerations for accounts, sessions and recovery

What a session alone can still do is stated in L-14.

The mailbox is the boundary here, and the design says so. Anyone who controls the account's mailbox and does not hold the master password can complete a mailed reset ([section 10.4](#)).

What they gain, without the kit, is an account whose vaults they cannot open and whose old ciphertext is sealed under a key they do not have, and a session opened under the account capability whose digest they sent. Unless they sent one they cannot present, that is not a session alone. It can change nothing inside a vault, but it can make every write [section 5.2](#) says the account capability gates, within the window's refusals. The session also reads the account's wrapped keys, so the thief can test guesses at the old password offline, at the cost [section 5.6](#) states, and a weak old password then gives up the notes. The owner sees the alert only if the thief leaves it in the mailbox. A thief who deletes both messages, the original message and the alert, and waits out the window leaves an owner who still knows the old password able to undo nothing, and an owner who had forgotten it with no way back at all.

A consequence of the undo's shape ([section 10.4](#)): L-27 *A reset can be undone by the previous password's login credential for 24 hours, unless it was made within seven days of the account's last undo.* An owner who still knows a password they believe has leaked is better served by changing it, which signs every other session out and leaves nothing to undo ([section 10.3](#)). Deleting the reset's alert from the mailbox remains the way it goes unseen. L-26 *A mailed reset is an account takeover, bounded only by an alert the thief can delete and a window the thief can outwait; with the kit as well it is unbounded.*

What retirement ([section 10.5](#)) cannot do is reach the lost device, which keeps two things. The first is its refresh token, valid for up to 30 days and renewed on use. It is the one relay credential a lost device keeps after the wrapping key has expired. It is a live session with everything a session alone can do: it reads, it ends things and it holds resources (L-14). Without the wrapping key, it writes nothing the account capability gates. It lasts until the owner retires the device, changes the master password from another device, or resets the account (L-22 *The refresh token's protection depends on whether the keyring accepted it, and a lost device holds a live session, whatever the state of its cached key, until the master password is changed from another device, the account is reset, or, unless it was lost in the middle of a sign-in, it is retired*). Each of these deletes it on the relay, the retirement only where the session is bound to the device ([section 10.5](#)). While the cached key below lasts, though, the finder can stop the change. The second is the cached wrapping key, which none of those touches (L-24 *A retired device's cached wrapping key is not wipeable from outside*). While the entry is still in the device's secret store ([section 5.6](#)) and the device's own disk protection does not stop the finder, the finder can open the root keys the device holds and, with a copy of the relay's ciphertext, read the history. Together the two make the finder more than a reader: L-23 *A lost device whose cached key is still live is one of the owner's devices to the finder who launches it.* A password change afterwards re-wraps the root keys and does not replace them (L-12).

11. What the relay can see, and what it cannot do

11.1 Visible to a curious operator

Item	What the relay sees or holds
The email address	The email address on each account, and when it registered, logged in, and synced
The account's identifier	The identifier it minted for the account (section 10.7)
Random identifiers	Random identifiers for vaults, pages, assets and devices, and which belong to which account
Device names	The device registry, with the name the person gave each device, up to 64 characters, in the clear
Coordinates	Every coordinate: which device wrote how many objects to which page, and when
Sizes	Sizes: of every note's operations, every snapshot, every file, with no padding
Lengths of files and paths	The exact length of every file and of every file's path
Lineage	Which generation identifiers share a root, which generation a root currently names, and when restarts happened; when a vault takes a new identity, which root or generation it replaced; and the hand-over records and lineage certificates beside the account's roots' and generations' pointers
The root each session asks about	Which session asks about which root
Progress	Version vectors and frontier reports, which say how far each device has caught up, and the cut of every page's compaction
The write capability	The write capability of a generation, in each request that writes into it; the relay keeps only its digest (section 5.3)
The account capability	The account capability, in each request that writes the account's keys, its vaults' identities or its device slots without proving the password or the mailbox; the relay keeps only its digest: on each session, the digest of the account capability it was opened under, which it never serves (section 5.2)
The application	The version and platform of the application on each device, which it sends with its requests
Credential hashes	The Argon2id hash of the auth-hash, and session token hashes
The account's key and address state	The key epoch and whether a change is in flight, whether the address is verified and whether the account still owes its first kit, and the kit generation
Wrapped keys	The wrapped roots with their kit wraps, and the recovery-kit record
Heartbeats	One sealed heartbeat per device

Item	What the relay sees or holds
Account events	When a reset of the account was last undone, and, if an operator has suspended the account, when and why

- Lengths of files and paths: because they are exact, a file the operator already knows can be recognised by its size, and a file uploaded from two devices by two equal sizes.
- The root each session asks about: the relay sees it because every generation-addressed request carries the root it believes in, so one vault's activity can be followed across every restart it survives.
- The account capability: the requests that carry it include the heartbeat, at least every six hours.

For 24 hours after a reset the relay holds the previous login credential's hash, the previous kit seed with its generation, and the cancel code's hash. These roll the account back: the cancel code always, and the previous login credential unless the reset was made within seven days of the account's last undo ([section 10.4](#)). Until each is delivered, the relay also holds the body of every queued mail, reset tokens and cancel codes in the clear included. A delivered body is blanked in the same write that marks it sent. An undelivered one is kept for retry, and blanked once the retry limit is reached, or once the mail is older than the longest-lived token a mail can carry. The mail's record, its recipient, subject, attempts and error, is kept without the body.

Deleting an account removes all the relay holds per account ([L-28 A self-hosted relay left on its default mail transport prints every mail to its log](#)):

- the rows "The account's identifier", "Credential hashes", "The account's key and address state", "Wrapped keys", "Heartbeats" and "Account events";
- in the row "The email address", the email;
- in the row "Device names", the device registry with the name each device was given;
- in the row "Lineage", the hand-over records and lineage certificates;
- in the row "Progress", each device's frontier reports;
- in the row "The write capability", the digest of each generation's write capability;
- in the row "The account capability", on each session, the digest of the account capability it was opened under;
- what it holds after a reset, and every queued mail with its record, above.

11.2 Not visible to anyone but the owner's devices

The text of any note. The name of any file or folder. Any key. The vault's display name. Whether two files are the same file, other than by length.

11.3 What a hostile relay can attempt

Each outcome names its tier from [section 1.1](#).

Attempt	Outcome
Read or alter a sealed object	Prevented. It does not open. On a note's page that is a hard error that stops that page syncing, with no alert (L-39 <i>A failed open or a divergence on a note's page raises no alert</i>). A heartbeat that does not open is treated as not heard, and an image record that does not open is logged and skipped (section 6.4)
Serve an object at another coordinate, kind, vault, generation or revision	Prevented. The associated data does not match, and it does not open
Serve different bytes at one coordinate	Detected, on a note's page: the Merkle forest diverges, and a hard error stops that page syncing, which shows as needing attention. No sync integrity alert is raised (L-39). On the image plane a device that already holds an entry at that revision compares path, content hash, size and deletion, keeps its own, and logs and drops a contradicting one. On the image plane a device that holds none takes whichever it is served first (section 9.2 , L-52 <i>Two different image entries at one revision are caught only where a device already holds one</i>)
Drop the tail of a page's history	Neither. It reads as stale, never wrong, and heals on honest contact (L-53 <i>Withholding and old views of notes read as lag</i>)
Serve a page's compaction under a cut other than the one it was sealed with	Prevented. It does not open, and the device takes nothing from it: no frontier, stored history or checkpoint moves. The page fails and shows as needing attention, with no alert, and stays so while the relay goes on serving it that way (section 6.2 , L-39)
Serve an older compaction of a page under its own older cut	Neither. It is authentic and opens. A device whose checkpoint the old cut does not cover refuses it; any other applies over it every operation it holds and loses nothing, and what is withheld reads as lag (section 6.4)
Ignore the write capability, or store a write that carries none	Neither, and nothing gained. The capability gates only what the relay itself stores; a device accepts only what opens under the key (section 5.6)
Ignore the account capability, or admit a write that carries none	Neither, and nothing gained that a relay refusing everything lacks. The capability gates only what the relay itself stores, which a hostile relay can rewrite anyway. A device still follows a new identity only under a certificate, and takes a kit or a hand-over only where it opens under the account's keys (sections 5.6 and 7.3)
Answer a device's writes with the account capability's refusal	Neither, and nothing gained that a relay refusing everything lacks. Every device it answers so signs itself out, drops its keys and asks for the master password, and the detectors of section 8.3 are dark until it signs in again. No file and no account state is lost (section 5.2 , L-41 <i>A relay that refuses the account capability signs every device out</i>)
Withhold everything, or withhold every heartbeat	Neither. Indistinguishable from a slow network; stale (L-53)
Refuse or fail every sync round	Neither. Never accused; the device shows an error or a decision to make, and the vault visibly stops syncing (L-40 <i>A relay that refuses every sync round is never accused</i>)

Attempt	Outcome
Serve what was demanded while staying one step ahead, keep one of the device's own pages failing, or answer a demand with other objects of the same author	Neither. Never alarmed on (L-42 , L-43)
Withhold from one device objects that another device's heartbeat shows exist	Detected , within three completed rounds of the demand, while the devices hear each other's heartbeats within a day, with the exceptions section 8.3 states
Serve two device groups different consistent histories	Detected , as the row above, whenever a fresh heartbeat crosses between the groups, including a split by page between converged readings, with the exceptions section 8.3 states. The eternal total partition is neither : the owner's two-screen check catches it (L-54 <i>The eternal total partition is undetectable in band by construction</i>)
Hide one device's heartbeat from a sibling	Detected on one observation the relay itself stamped. A sibling kept on a device list read from before the heartbeat sees the device only as stale (L-46 <i>A heartbeat hidden by a frozen device list reads as staleness</i>)
Serve, after an alert for withheld objects or a differing registry is raised, a sibling's heartbeat that carries nothing for the vault	Neither. The alert clears, with the objects still withheld, where no other demand or latch stands (L-47 <i>A raised sync integrity alert can be cleared by a heartbeat that says nothing of the vault</i>)
Serve the earlier of two heartbeats a device published within one second, stamped as read after the later	Neither. The hidden-heartbeat check accepts it, until the device publishes again (L-48 <i>The hidden-heartbeat check tells publishes apart by the second</i>)
Drop the accused device from the device list	Neither. A raised alert can be cleared this way, once a round hears another sibling in agreement (L-49 <i>The relay can clear a raised sync integrity alert by dropping the device it accuses</i>)
Invent a hand-over or serve an old one	Prevented. An invented record does not open or does not match the clear pointer, and an old one is below the epoch the device committed; both are refused. A device that has committed no epoch for the root reads the relay's current hand-over as the truth (L-5 <i>A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup</i>)
Mark a root retired	Neither. Believed, since the relay keeps a root's state and can check no more than a certificate's length. The devices on that root stop syncing the vault and wait for a successor; no file is touched (section 7.4)
Point a retired root, or a superseded generation, at another of the account's vaults	Prevented. A device follows the pointer only under a lineage certificate that opens under the successor's root key, ends in that successor and lists the root the device holds before it. Otherwise it adopts nothing, and the vault waits on the person's decision (section 7.3)

Attempt	Outcome
Withhold a root's predecessor, or the certificate on its link, from a device that joins	Neither. The device joins with a shorter chain, and nothing shows. A new identity it later gives the vault cannot be followed by a device still on the ancestor that was withheld (L-67 <i>A device can be left unable to follow its vault to a new identity</i>)
Serve an old image revision, or omit an image	Neither. No image state crosses between devices, so both screens can read as synced; comparing the two devices' files is the check (L-56 <i>Images have no freshness signal</i>)
Serve a replaced kit seed as current	Prevented for a device whose own kit record is newer than the replaced one: a seed record served under another generation does not open, and the device refuses a generation below its own record's, or a different seed at that same generation where it published its own record and that record still opens under its wrapping key. A device that has committed none accepts what it is served, and one whose own record is older takes a newer record that opens under its wrapping key, a replaced kit's included (L-5)
Drop a root's kit wrap, or take a root published without one	Neither. The kit then does not open that root, which shows when the kit is used to reset the password: the reset reports the vaults it could not open (L-63). An honest relay refuses a root without its kit wrap for an account that holds a kit or still owes its first one, and the app makes or adopts the account's kit before it enables sync for a vault or rebuilds one. A relay that does not require address verification never records that an account owes a kit, so there only the app's check stands (L-62 <i>On a relay that does not require address verification, until the kit's seed reaches the relay, only the application keeps a vault from being published without its kit wrap</i>). A root taken without its kit wrap can get one later (L-62)
Keep a copy of old ciphertext after a kit replacement or password change	Neither. It keeps it; the old wraps stay openable to whoever held the old kit or wrapping key (L-29 <i>Kit replacement and password change replace what an honest relay serves; they cannot un-publish bytes a dishonest one kept</i>)

12. Verification

What stands behind the claims above, in words. None of it substitutes for an outside reading, which is why this document exists.

- **How this text was read.** Before this release the text was read whole, with the code open, by four readers, each a frontier AI model working alone. This release corrects the sentences the code or its history contradicted, and restructures the text; every rewritten sentence was checked against the passage it replaces, and every new sentence against the passages it rests on. Every defect that the reads of this text before this release found is fixed, or accepted as a limitation and stated in [section 13](#); the rest of what they found is stated there as a limitation, or was a wrong sentence and is corrected. [L-1](#) *No external review has taken place.*
- **Byte-pinned vectors.** Every associated-data layout is pinned to its exact bytes. Under fixed keys, nonces and plaintexts, so are the sealed output for a note's operation, a page registry entry and the vault's label, and so are the master key and the three account keys. So, under

fixed keys, nonces and plaintexts, is a hand-over record, both the tuple sealed inside it and the sealed record, under a hand-over key derived from a fixed root key, so that key's derivation is pinned as well. So, under fixed keys, nonces and plaintexts, are a compaction's associated data, with the digest of its cut, and a compaction's sealed output. So, under fixed keys, nonces and plaintexts, are the write capability derived from a fixed generation key and the account capability derived from a fixed wrapping key and origin, each a vector computed outside the application's code. An implementation change that alters any of those by a single byte fails. The rest is not pinned to bytes. The generation, kit and lineage keys are tested for being deterministic and distinct from one another. The wrapped root, the root's kit wrap, the wrapped kit seed, the wrapped generation key, the lineage certificate, the heartbeat and both image seals are tested for opening where they were sealed and nowhere else.

- **The cross-open matrix.** Every sealed kind in the protocol is sealed and then opened in every other kind's context, at the sealing layer over constructed coordinates, and the test asserts refusal everywhere but the diagonal. This replaced a hand-written list of cases, which is how the asset plane's two objects came to be built under one domain; they were caught before any release carried them. It does not cover each caller's choice of coordinate; those are pinned by the tests of the flows that call it.
- **The misuse family.** A property test generates seals of content objects and, in each generated case, applies one misuse drawn at random from a fixed list. Each misuse on the list is a change to one field of the associated data, the kind, the generation, the page, the actor or the seq; an open under the wrong key or the wrong nonce; or a sealed content object moved into the slot of a wrapped generation key, or back. It asserts that the misuse fails to open. A second flips bits in the ciphertext and tag, truncates and extends it, with the same result. It does not take a field's value from another valid seal, and it does not reach the associated data of the image plane, the wraps, the lineage certificate or the kit, which the cross-open matrix above covers across kinds. Sealing and then opening returns the payload, over generated payloads.
- **Keys never touch disk unwrapped.** A test enables sync for a vault over a real temporary store, then stores a recovery kit for the account and wraps the vault's root under the kit key through the application's own wrap function. It then asserts that no bytes of the generation key, the root key, the kit seed or the kit key appear in any file of the device's two stores. It does not run unlock. The kit record and the root's kit wrap it searches were written by the test, not by enabling sync. It does not look for the wrapping key or the attest key, which by design are kept in the operating system's secret store.
- **A cache hit performs no network I/O.** Unlock is cache-first, and a test pins that a hit makes no request, which is what forces every key change onto a fresh generation rather than an in-place migration.
- **The state matrix.** For one credential, four combinations of session state, seated wrapping key, cached wrap and vault ownership, signed in and signed out, are driven through the real derivation path and their status asserted. The matrix records why three others cannot be reached. An expired session, and the state a refusal of the account capability leaves, are not rows of it; the second is pinned by the tests of the account capability below. When a reset is

offered, and that a password changed on another device is recognised as that rather than offered a reset, are pinned by the tests beside it. A password change is outside the matrix.

- **The lying relay.** A deterministic simulator runs the same merge engine and the same detection kernel as the application against a relay that withholds, replays, rolls back, omits the middle of a history, serves a compaction under a raised cut, splits one author's pages between two siblings, and hides one device's heartbeat from another. It asserts the outcome [section 11.3](#) gives for each, with honest controls that must raise nothing. These are hand-built, bounded scenarios, not a fuzz corpus. The simulator has no vault root, no restart of a vault's identity, no password and no recovery kit. So the rows of [section 11.3](#) about hand-overs, new identities and the kit are pinned instead by the application's own tests, against a real relay whose stored state or answers a test can alter.
- **Fork detection.** The heartbeat kernel's monotonicity and compaction-independence are property-tested, and the simulator above carries a counterexample for each check of [section 8.3](#). In the application, a test drives real sweeping rounds against a withholding relay until the sync integrity alert fires on the third completed round. In the application, another test shows that an honest relay is never accused after a vault's lineage restarts on three devices. Further tests pin that a round ended early by a root or key refusal, or inside its drain, counts toward nothing.
- **Lineage.** The relay refuses a retire and a successor link that carry no certificate, before it changes anything. The device's check admits the strict ancestors a certificate lists and nothing else. It refuses a certificate that is absent, read at another coordinate, ending in a root other than the one opened, sealed under another root's key, or not a certificate at all. A device offered an uncertified successor lands in a decision rather than an adoption. Against a real relay, a device two identities behind a device that joined fresh follows the new identity that device mints. Against a real relay, a join is refused when its two proofs of ancestry disagree or when the served predecessor is not the chain's last ancestor, and is not refused by a certificate that does not open. Against a real relay, the chain is on disk before the root at a crash between any two of the writes. A session that tries to found a generation under a newly linked successor is refused, a successor published without its first generation is refused, a refused founding links nothing and stores nothing, and a founding sent again reads its own link back and cannot change the write capability. A retire, a successor's publish, a hand-over and a supersede are each refused without the account capability.
- **A compaction's cut.** In the application, against a real relay, a device served a compaction under a raised cut refuses it at both places that act on one, the adoption and the merge. In the same tests, the page's operations, checkpoint and file are intact after a restart. Each test failed before the cut was bound into the seal. Removing the cut from the associated data fails every one of these tests.
- **The write capability.** Against a real relay, five things a session without the vault's key could do before are each attempted and refused. They are an image register entry at the top of the revision counter, an overwrite of a stored image, a compaction uploaded again under a raised cut, the purge of a page and of a vault, and a write by a mailed reset's session into a vault from before the reset. Each test failed before the capability existed. Other tests pin which requests ask for the capability and which do not. They pin that each request which founds a

generation writes the digest once and nothing rewrites it, an undone reset included. They also pin that the application turns a refused capability on each of its ten write requests into one decision put to the person: that the synced copy cannot be read with this device's key.

- **The account capability.** Against a real relay, every write it gates is refused without it and under a wrong one, before any answer that depends on what the relay holds for the account. The session's own refusals and the request's own checks of its sizes and form come first, and the tests do not reach them. Against a real relay, a census of every route the relay mounts that changes anything fails when one is added without being classed as gated or not. Against a real relay, no request makes the relay's record of a vault without the capability, and the digest is never served. A vault's first enable, and the reset of a vault's relay copy from the files, each publish the new root and bind its first generation in two requests ([section 7.2](#)). Both present the account capability, so a session alone can neither publish such a root nor found a generation under it between the two, and that is tested. Each of the application's ten requests that carry it is refused under another key and under another relay's capability, and admitted under its own. A scheduled sync round and the heartbeat present the capability of the key that is seated. Through the application's own work against a real relay, a device whose heartbeat or whose removal of another device is refused is signed out, keeps its account state, stays so across a refresh, a restart and an unlock, and resumes at a sign-in with the right password. Through the same work, a password change, run whole, cut before its final request, or with that request's answer lost, presents the capability it started with until its final request and is admitted when run again.
- **The transport.** An `http://` address is accepted only at a literal loopback host, and a request to any other host over plain HTTP is refused before it leaves. A redirect is not followed and its target receives nothing. A request to a loopback address ignores a proxy set in the environment.
- **SVG.** An SVG reaches the screen only as the `src` of an `<img>` element, which does not run script an image carries. No file that can touch asset bytes carries a markup sink that could route it anywhere else ([L-35 SVG is an admitted attachment format, and a peer's SVG is admitted exactly as this device's own](#)). Both halves are pinned. One test asserts the sink-free property over every such file and fails on an unpinned new consumer of the asset-URL seam. Another asserts that the image viewer puts the URL in an `<img> src` and in no other attribute anywhere.
- **Model checking.** Six of Slate's own coordination rules are written a second time as TLA+ models and checked exhaustively over small instances, by hand, with every configuration's result recorded beside its model. The first is the agreement on which old history may be discarded, including a relay that lies about a compaction's cut. The second is the binary-file register. The third is the vault identity hand-over, including the lineage certificate's check, a device that joins and records the chain it is served, a new identity's first generation founded in the request that links it, and a thief who holds a session and not the password and tries to retire, link a successor, hand over and found, each admitted only with the account capability (the ordering of the relay's pointer, its epoch and the two stores on a device, and not the cryptography). The fourth is the master-password change, including a thief who holds a copy of a session and not the password. The fifth is which sync rounds count toward the withholding alert. The sixth is the life of the account capability's digest on the session, across

sign-ins, a password change, a re-tune, a mailed reset and its undo, moving the credential to a relay's new address, a mistyped unlock, a change cut before its final request and a copied token. It is modelled with the rule that no write the capability gates is admitted from a copied token alone, or on a session whose digest is of a key the account has left. That last model is looser than the application in four places. First, its device presents the capability a change started with on every write while the change is open, where the application's rounds and heartbeat present the seated key's, and are refused where that is the change's new key. Second, its copied token is admitted to nothing by the guard on its own action, so the rule holds by construction, and a thief presenting a capability read in transit is not modelled. Third, its change's final request requires the capability, where the relay's asks for the password instead. Fourth, its reset is the honest device's, which derives the login credential and the digest from one password ([section 3.2](#)). Five configurations across three of the models are recorded as failing, and left so on purpose. One is of the hand-over model, pinning one of its findings. Three are of the password-change model, which pin one finding between them. One is of the withholding model, which states the trade of [section 8.3](#) that a relay refusing every round is never accused. The hand-over model has been brought level with the durable transition marker the application gives an interrupted rejoin, and the configuration that pinned the rejoin without one now passes. The four that are not the withholding model's pin two findings, one of each of those two models, that are not yet closed and not yet fixed. Each model also carries configurations that remove a safeguard and must fail, to show that it is needed. Among them, the hand-over model without the account capability lets the thief spend a retired root's link. Among them too, the account-capability model fails without the capability, with a re-tune or a password change that leaves the other sessions' digests standing, and with a move that replaces the moving device's. Two of the hand-over model's do not fail, and are recorded as findings of the model. In the first, without the compare-and-set on a root's pointer, the epoch alone still keeps one live generation per root ([section 7.2](#)). The model finds that removing either the compare-and-set or the epoch of [section 7.3](#) alone leaves the property holding that a root has at most one live generation at every instant, and that removing both breaks it. In the second, one clause of the rule that stops a retired root being bound again has no case the model can reach. No model's own configurations are checked by the model checker in continuous integration or in the nightly run, and the model checker itself is not committed with the models. What runs every night is narrower: recorded simulator traces are checked against two of the models, as a referee. The history-discard model is checked against a batch of traces the simulator generates that night, on every night whose simulator run passes, and against a committed set of traces on nights the code has changed. The binary-file register is checked against committed traces only, on nights the code has changed, because some traces in a generated batch take longer to check than the referee allows. The other four models have no trace check, because the simulator has no vault root, no restart of a vault's identity and no password. So the implementation as a whole is not checked against the models: two of them referee the simulator's traces, and the rest of the code is held to them by tests.

- **Tested, not modelled.** Several of Slate's own rules are pinned by tests and have no model yet, which falls short of the standard the project sets for its own coordination rules, and is recorded as such:

- the rest of the withholding detection of [section 8.3](#), that is the registry, split-by-page and hidden-heartbeat checks, the heartbeat's freshness and the bar a device's own failures raise;
 - which vault root a relay may store without a kit wrap;
 - which requests the relay admits on a write capability, of which only the founding of a new identity's first generation is modelled;
 - which clock a device gives a page's registry entry when it creates, renames, deletes or restores the page;
 - what "synced" counts for images;
 - a password change's adoption of a kit another device published;
 - the single send of a registry entry every other device refuses;
 - the mailed reset's window;
 - session rotation;
 - the binding of the device an envelope, an image's register entry or an image's upload names to the session that sends it;
 - how work a device holds back is counted;
 - which account a pending registration may verify;
 - the account capability's derivation and the relay's comparison of its digest, which the hand-over and account-capability models take as given, and whose running in constant time is by construction, read in the code and not tested;
 - the recovery-kit record's single step under a ceiling;
 - and the sign-out of a device whose account capability is refused.
- **Argued, not tested.** In the wrapping key and in a generation key, the same 32 bytes are used as a cipher key and as an HKDF key ([section 5.6](#)). The design relies on HKDF's output revealing nothing of its input; that is argued, not tested. Nothing on the relay names the new root of a vault's first enable, or of the reset of a vault's relay copy from the files, to any other session before its generation exists, since it replaces nothing by a link ([section 7.2](#)). So the window in which such a root stands on the relay without its first generation does not open there for anyone but whoever read the publish itself in transit, who holds the root's identifier, the account capability and a session together; that is argued, not tested. For a registration the relay admits, and every request for a fresh code ([section 10.1](#)), that their timing matches whatever the relay knows about the address is argued from the single hash, not measured.
 - **The recovery kit.** A round-trip property over generated seeds, a pinned vector per rendering, and a checksum-refusal case, for the words and the file. Against a real relay, a first sign-in whose read of the kit fails still makes the kit before any vault is published, and the relay refuses a root with no kit wrap from an account that owes its kit until the seed lands.
 - **Registration.** When a code is presented, the relay hashes the offered auth-hash once, and every well-formed attempt, accepted or refused, pays exactly that one hash, which a test counts ([section 10.1](#)). Tests compare both on an open relay: a registration the relay admits,

and every request for a fresh code, each answer byte-for-byte the same whatever the relay knows about the address.

- **Sessions.** The lost-response recovery inside the grace window, family deletion past it, reuse detection and proactive refresh are each pinned, some against a network proxy that drops the exact response. So is a password change keeping the session that made it and signing every other session out. A session without the password cannot declare a password change, the owner's change lands while such a session tries, and one session's flood of password proofs does not refuse another session's. The account capability's digest is pinned through a sign-in, a password change, a re-tune, moving the credential to a relay's new address and a refresh, and a completed reset opens its session under the digest it sent.

What none of this proves: that the composition is sound, that the primitives are used as their authors intended, or that the threat model is complete. Those are what an outside reading is for. A reader doing one would still need what this document does not carry:

- the source at the named commit;
- the exact wire encodings and parser limits;
- independent test vectors;
- the model files with their configurations and the commands that reproduce their results;
- a password-strength assumption and measured guessing costs;
- and the state machines for concurrent transitions, crashes and cloned device stores.

13. Known limitations and residual risks

The limitations are stated in the words the project records them in, so that a reader can decide for themselves. They are grouped by the tiers of [section 1.1](#): what prevention does not cover, where detection stops, what is left to the owner's own comparison, and what ordinary use of the application can still get wrong. Each limitation has its one entry in section 13, under a number it keeps, L-1 to L-73. The other sections cite it by its number, and at a first citation in a section outside a run of citations by its headline. Where a limitation has an actor, its entry says who, holding what, can do what, and, where the person sees something, what they see; an entry that is accepted for this release says so, and one ticketed for a fix says when, or its group's opening sentence does. What is listed here is not only what the design cannot do: it includes faults the project means to fix and has ticketed, and an entry that is ticketed says so.

13.1 The project

- **L-1. No external review has taken place.** The residual risk was accepted on the record on 5 July 2026. The review becomes a priority as Slate approaches a wider public release. Nothing on the website calls the encryption examined, and this document does not either.
- **L-2. One person builds Slate, with an AI assistant, and the source is not published.** Every claim here is prose to be taken on faith or tested from outside. The verification apparatus in [section 12](#) is the project's answer to the first half of that; this document is its answer to the second.

- **L-3. The hosted relay is by invitation and has not been operated under attack.** Its rate limits and admission controls are engineering, not cryptography, and are outside this document.

13.2 What prevention does not cover

- **L-4. A lineage certificate proves authorship by someone who holds the account's keys, not by a particular device.** Someone who knows the master password can mint a root, certify it as the successor of any vault of the account and have it linked. The owner's devices follow it, as they would follow one of their own. The certificate excludes the relay and a session alone. A person who knows the master password is outside the threat model of [section 3.2](#), and this is the design's stated limit.
- **L-5. A device with nothing committed, or whose kit record is older than a served record that opens under its wrapping key, believes a relay restored from a backup.** A device that has never committed an epoch for a root reads the relay's current hand-over as the truth, exactly as a fresh device reads everything else when it joins. One that has never committed a kit generation accepts whichever consistent recovery-kit record it is served. A device that has committed either refuses a record below it, or a different one at the same point, which for the kit it refuses only where it published its own record there and that record still opens under its wrapping key. Above its own kit record a device has committed nothing, and takes any record there that opens under its wrapping key. A kit replaced without a password change leaves the old kit's record sealed under the same wrapping key. So a relay that kept that record can serve it to a new device, or to one whose own kit record is older than it, where it opens. That device then wraps under the old kit every vault it creates. And one that held an older kit re-wraps under it every root its sweep reaches. So those vaults open under the replaced kit's words and not under the current kit's. A kit replaced because its words leaked therefore goes on opening them until a device that holds the current kit re-wraps them in its sweep over the account's roots. Being served the current kit does not re-wrap anything by itself.
- **L-6. When two devices race to hand a vault over or give it a new identity, a relay that kept both records can split other devices between two generations, with no alarm in band.** Each seals an authentic record. One wins the relay's compare-and-set; the loser records what it lost and refuses it if it is ever served back, and a device that committed the winner refuses the other. Such a relay can still serve one to one other device and the other to another, splitting them between generations of one vault. The heartbeat checks compare only devices on the same generation, so nothing in band raises an alarm ([section 7.4](#)).
- **L-7. A device that holds the vault's key can stop an honest device's changes reaching anyone.** A registry entry whose clock sits exactly at the top of the allowed range is accepted. The receiving device's next rename, deletion or restoration of a page then carries a clock above the range, which every other device refuses and reports as a sync failure. The honest device sends each such entry once and is told nothing, and its changes to which pages exist stop reaching the others. A note change at the top of its clock's range does the same to that device's later note changes. An image write at the top of the revision counter gets every later image write in the vault refused by the relay. Each needs a device that holds the vault's key and misbehaves, which [section 3.2](#) puts out of scope. Each is ticketed for after the beta opens. No rule on the writing side alone can close them, because every range has a top.

- **L-8. Whoever holds the account capability with a session can publish roots and found generations of their own, with write capabilities of their choosing, up to the account's limit on vaults.** The write capability is whatever the founder of a generation presented. The relay holds no key, so it cannot check that the capability fixed when a generation's record is made was derived from that generation's key. It enforces only that every later write presents the same one ([section 5.3](#)). An honest device derives it from the key. Every request that makes such a record presents the account capability as well, so a session alone cannot publish roots and found generations of its own. Nothing in such roots and generations opens under any key of the account. But the owner's application offers no action that removes them, a purge by name needs the write capability the founder chose, and nothing frees their roots. And they count against limits that are the account's, not a vault's. Three limits can be spent this way. The number of vaults: a new vault is then refused. The stored bytes, which are summed over every vault of the account: a caller that fills a generation of their own leaves the owner's vaults refused for space. And the number of roots: a caller that spends them leaves the account unable, for good, to enable a new vault, give a vault a new identity or restart one after a reset without the kit. The number of roots is 64 for the account's life, across all its vaults, because neither a retire nor a purge deletes a root's row, and only the undo of a mailed reset removes the roots made inside its window. Ordinary use spends the same 64, one for each vault enabled, each new identity and each vault restarted by such a reset. This limitation needs the account capability as well as a session, and is ticketed for after the beta opens.
- **L-9. The write capability is a bearer value, and is not wiped everywhere.** It is the same for every request into a generation for that generation's life. Nothing rotates it. A restart of the vault's history gives the new generation a new capability. Whoever reads a request in the clear holds it. A hostile relay reads requests in the clear, and so does anything that terminates TLS in front of an honest one. With a session as well, that party can write into the generation on the relay and purge it. A device accepts what that party writes only if it opens under the key. In memory, the application wipes the buffer it writes the capability into, but not the copy the HTTP library makes for each request. The HTTP library's copy of each capability's header is an ordinary string. The relay does not wipe the header it received ([section 5.5](#)). Wiping the request's copy is ticketed for after the beta opens.
- **L-10. The account capability is a bearer value too, and proves only what the session's opener held.** It is the same on every request it gates for the life of one password under one Argon2id parameter set at one relay's origin, which a password change or a re-tune ends. The heartbeat sends it at least every six hours. So whoever reads one such request in the clear, and holds a session, can write the account's keys, its vaults' identities and its device slots. A hostile relay reads such requests in the clear, and so does anything that terminates TLS in front of an honest one. A password change would end that, but the holder can prevent one. It can write a recovery-kit record the password does not open, which stops the change ([L-15](#) *Whoever holds the account capability with a session can, with one write, stop every step of the remedy after a lost device but the retirement, the password change among them*). It can also end a change in flight by removing the device whose session runs it. What ends its power is the session it uses losing the matching digest: by its end, in retiring its device or signing it out, by a password change or a re-tune that completes, or by a mailed reset. The

relay compares only a digest, and cannot check that the digest a sign-in sent was derived from the wrapping key. So whoever holds the login credential, without the password, can sign in under a digest of their choosing and write the same, which that holder could already reach by changing the password. No session's end bounds them ([section 5.6](#)). Whoever reads one sign-in in the clear holds that credential, and needs neither a session nor a captured capability. In memory, the application wipes the buffer it writes the account capability into. Neither the copy the HTTP library makes of its header for each request nor the relay's copy of the header it received is wiped. These are accepted for this release, and wiping the request's copy is ticketed with the write capability's.

- **L-11. With shared symmetric keys only, a compromised device while unlocked can publish anything a sibling could, and nothing distinguishes the two.** There is no per-device identity. This is the trade [section 8.4](#) makes.
- **L-12. Root keys are permanent, so there is no cryptographic revocation.** A password change, a device retirement and a kit replacement each change what an honest relay serves from now on. None replaces a root key. Whoever extracted a root key, from a lost device's cache, from an old kit, or from an old wrap and an old password, opens every future generation of that vault; for a key from a lost device's cache, that is not only the history on the device. The remedy is to replace the kit, because the wrapped seed sits beside the roots, and then to start a new vault identity. They come in that order because a new root is wrapped under the kit the account has when it is published. Where the wrapping key itself was taken, as from a lost device's cache, the remedy is also to change the master password before either, or the new seed and the new root are wrapped under the key that was taken. For a lost device, it is also to retire the device before that password change ([section 10.5](#)).
- **L-13. The recovery kit is a complete credential, not a second factor.** A kit together with a copy of the relay's database, or with any session of the account, opens every vault of the account with no password, because the relay serves a session the kit wraps. The mailbox yields such a session through a reset. [Section 3.2](#) names this adversary; this is the answer.
- **L-14. A session alone can read, end things, and hold things; whoever holds the account capability with a session can destroy the relay's copy of a vault whole.** A session alone is not a destruction capability; what it can still do is read, end things, and hold things. Retiring a root, handing one over, pointing a superseded generation at another vault, removing another device, and overwriting a root's wrapped key, the recovery-kit record or a device's heartbeat each present the account capability ([section 5.2](#)). A copied refresh token carries none. A copied refresh token alone cannot spend the account's roots and limits on roots and generations of its own, stop the owner's password change through the recovery-kit record, spend a retired root's successor link, or raise the sync integrity alert by overwriting a heartbeat. No password prompt comes with the account capability but the one before a locked application removes a device ([section 10.5](#)). A session alone cannot write, overwrite or remove anything inside a vault while the vault stands, which needs the write capability ([section 5.3](#)). It cannot delete the account, change its password or declare a password change, which need the password. A session alone reads what the relay serves it: the account's list of vaults, the wrapped keys of the account's roots, their kit wraps, the recovery-kit record, every device's heartbeat, the devices' names, the account's key epoch, and any stored object whose coordinate it can name. That is the position [section 5.6](#)

describes for guessing the password offline, and it can guess the password offline against what it reads. It does not read a vault's operations, which are fetched only with the write capability. It can end its own session. By presenting a refresh token that has already been rotated, it can end the session family it shares with the device it was copied from, and so a password change in flight on that device. The owner signs in and runs the change again. It can hold the account's four request slots and two transfer slots with slow requests, writes that carry a wrong account capability among them, for as long as it keeps at it. During that time the owner can neither sync nor change the password. Nothing reserves a slot for a password change, so this lasts as long as the holder sustains it. A copied token is the same session as its device, so it shares that device's limit on password proofs. It can therefore keep that device's password proofs refused, and keep that device's password change, declaration, move and deletion refused. It can spend up to half of the account's ceiling on password proofs. It can grow the relay's tables of tokens. The way out is that removing a device, signing in and signing out take no slot, so the owner can remove the device the token was copied from. Removing that one ends the token's session and all of it, unless the token's session is bound to no device ([section 10.5](#)). The files on the devices are untouched by any of this. The last four of these are ticketed for after the beta opens: holding the slots, keeping that device's password proofs refused, spending up to half of the account's ceiling on password proofs, and growing the relay's tables of tokens. A copied session also outlives a password change made on the device it was copied from, because a change keeps the session that made it. That session then holds the new key's digest, which the copier cannot present. Whoever holds the account capability with a session can make every write that presents it, removing every device and overwriting a root's two wraps among them. Such a holder can still destroy the relay's copy of a vault whole, by a retire or a hand-over, and the vault's devices then stop syncing it in front of the person. The notes on the owner's own devices are untouched. Those holders are each of the owner's devices, a lost device's finder while its cached key lasts ([L-23 A lost device whose cached key is still live is one of the owner's devices to the finder who launches it](#)), and whoever read a request that carried the capability. They are also the holder of the login credential ([L-18 The login credential is bound to an origin, with two exceptions](#)), and whoever completes a mailed reset, after its window ([section 10.4](#)). The same holders, and a mailed reset's session within its window too, can remove every device of the account, which signs each out, as often as they like. They can also overwrite a root's wrapped key and its kit wrap with bytes that open under nothing. After that, no new device can join that vault and the kit no longer opens it at a later reset. A well-formed overwrite of a root's wrapped key raises the changed-elsewhere prompt on the devices that hold the root ([section 10.3](#)). Bytes that do not decode, and a spoilt kit wrap, show nothing until a join or a reset fails. Nothing rewrites such a root unasked but a password change or a kit replacement on a device that still holds it. A recovery-kit record the password does not open stops both. An undo restores wraps changed inside its window. These are accepted for this release.

- **L-15. Whoever holds the account capability with a session can, with one write, stop every step of the remedy after a lost device but the retirement, the password change among them.** A recovery-kit record the current password does not open makes a password change refuse before anything is declared. Such a record also fails a kit replacement and every

publish of a root. It comes only from a hostile relay, a defect, a mailed reset, or whoever holds the account capability with a session. A mailed reset, which posts a record above it, repairs it. So one write by whoever holds the account capability with a session stops every step of the remedy after a lost device but the retirement, and keeps the password change that would end that holder's power from completing ([section 10.5](#)). The relay takes a write of the account's recovery-kit record only with the account capability or in a password change's final request, and only one generation above the stored one or as an identical retry. A reset's undo or cancel puts back the record the reset replaced ([section 5.2](#)). So a session alone cannot write it, and no write can leave it at the top of its range ([section 5.4](#)). A password change that replaces a relay record it cannot open was not built. This is ticketed for after the beta opens.

- **L-16. Whoever holds the account capability with a session can spend a retired root's one successor link.** A vault's new identity is a retire followed by a successor's publish. The relay links a retired root to a successor once. It checks that the request presents the account capability and that the certificate has the right length, and nothing else. Whoever holds the account capability with a session can therefore publish a root of their own, with a junk certificate, as the successor of a root that is retired. That root can be one they retired themselves, or one the owner's device has retired a moment before it publishes. The certificate does not open, so no device follows the link. But the link is spent, for good unless it was made inside a mailed reset's window and the reset is undone. The owner's device is told another successor won, and cannot follow it either. Every device of the vault lands in the state described at [L-67](#) *A device can be left unable to follow its vault to a new identity*. Each has to be set up again against a new vault by hand. No note is lost. The pointer that marks a generation as superseded is written once in the same way, under the same capability. So whoever sets it first leaves a later reset without the kit unable to point that generation at the vault that replaced it. So the account's other devices, which can no longer open the old root, find no successor they can follow. What this entry describes needs the account capability as well as a session, and is accepted for this release.
- **L-17. Whoever holds the account capability with a session can raise the sync integrity alert against an honest relay.** A device's heartbeat slot is written only by a session bound to that device that presents the account capability. So a copied refresh token alone cannot write it. A session opened under the capability can bind itself to an existing device by re-enrolling it. That is how whoever completes a mailed reset reaches a device's slot, inside the reset's window or after it. An undo of the reset does not put the slot back ([section 10.4](#)). Such a caller can overwrite the slot with bytes that do not open, or with the device's own older heartbeat. A sibling then reports that it did not hear the device's current heartbeat. The device reads that as the relay hiding it, on one observation ([section 8.3](#)). The alert names the relay. The relay's stamps were honest throughout. It needs the account capability as well as a session. It is accepted for this release.
- **L-18. The login credential is bound to an origin, with two exceptions.** First, two relays served at two paths of one origin share an origin, so the operator of one can sign in at the other as a person who uses the same email and password at both. Second, moving the credential to a relay's new address sends the new origin's credential to the relay at the old one, over the live session ([section 5.5](#)). That is harmless where the two addresses are one relay, which is what the move is for. Pointed at a different relay where the person uses the same email and

password, it hands the old relay's operator a credential for it. The application does not warn in either case. What is handed over is the account's credential at the other relay: a session there, and the proof of the password that relay asks for wherever it asks for it, as before it deletes the account or changes its password. So its holder can delete the account at that relay, or change its password and sign every device out. That holder can also sign in under an account capability's digest of their choosing, because the relay stores the digest it is sent and cannot check how it was derived. They can then write the account's keys, its vaults' identities and its device slots at that relay. The sign-in mails no one and signs no one out. Of the writes, only one that replaces the recovery-kit record the relay holds with different bytes mails the owner ([section 10.6](#)). Of the writes, only removing a device signs one out, the device removed. That holder can replace the credential as well. Moving the credential to a relay's new address needs only the credential. It sends no mail and signs no one out, so the owner's devices go on until one has to sign in and cannot. A change marked as a re-tune, a mark the relay takes on the request's word, keeps every session but clears their digests. So each other device signs itself out by its next heartbeat, and its password no longer signs in. The mail says the password did not change if this was an ordinary update. It tells an owner who did not start it to change the password. Because a reset's undo is proved by the previous login credential, the credential handed over can undo the reset the owner makes to be rid of it ([section 10.4](#)). The same holds for whoever reads one sign-in in the clear. The move itself carries no account capability and no digest to either relay. The capability's label carries the canonical origin as the credential's does, so two relays at two paths of one origin share the account capability as well. No key is handed over, and no note opens. It is ticketed for after the beta opens.

- **L-19. Nothing binds a key to a relay.** A relay that holds another relay's copy of the same person's account, its wrapped roots and sealed objects with their identifiers kept, can serve it to a device that joins fresh, and everything opens ([section 5.6](#)). The device adopts the person's own vault from the other relay, as it adopts whatever vault a relay lists. The copy is the person's own content, old or from elsewhere. A device that already holds the vault's identifiers and epochs is not moved by it. This is a trade of the design, not a fault awaiting a fix.
- **L-20. A relay on the device itself is dialled without TLS.** The one `http://` address the application accepts is a literal loopback address ([section 10.2](#)). A request to it does not leave the device. Whatever else runs on the device as the same user can read it. That party is the compromised device [section 3.2](#) puts out of scope.
- **L-21. A refresh token replayed inside the grace window yields a second live branch of the session that reuse detection does not see.** The branch has its own 30-day refresh token, and lives as long as it keeps refreshing. The branch belongs to the same session. So a master-password change made from that session keeps it, where the change signs every other session out. Signing that session out, retiring the device it is bound to or a mailed reset ends it. The branch shares the session's account capability digest, which its holder cannot present. So it reads, ends things and holds resources, and writes nothing the capability gates. After a change made from that session the digest is the new key's, which the holder cannot present either. The owner sees nothing. The grace window is an accepted trade for

the lost response it recovers from. This consequence is accepted for this release. A fix is ticketed for after the beta opens.

- **L-22. The refresh token's protection depends on whether the keyring accepted it, and a lost device holds a live session, whatever the state of its cached key, until the master password is changed from another device, the account is reset, or, unless it was lost in the middle of a sign-in, it is retired.** Where the keyring accepted it, it lives in the operating system's own keyring, behind whatever the user's session or login unlocks. On iPhone the keyring is the Keychain ([L-37](#) *The Keychain entry on a Mac carries the platform's defaults*). Where it did not, it lives instead in a column of the device's own account store, a file the app creates owner-only and refuses to open if it cannot be made so. Those cases are a desktop session with no keyring service running, and an iPhone whose Keychain refused the token at sign-in. There, on desktop, anything running as the same user reads it. On iPhone, the app's own sandbox is what stands between it and anything else. The phone's backups hold it too. A later launch tries to move it into the Keychain ([section 10.1](#)). Separately and inherently, a lost device holds a live session until it is retired, the master password is changed from another device, or the account is reset, whatever the state of its cached key. A device lost in the middle of a sign-in keeps one that only the last two end. While the cached key lasts, its finder can stop the change ([L-23](#)).
- **L-23. A lost device whose cached key is still live is one of the owner's devices to the finder who launches it.** A launch seats the cached key without asking for the password, and the device's session is live. So until the device is retired or the account is reset by mail, the finder receives every new note, and writes notes and images that the owner's other devices accept as the owner's. Because the cached key derives the account capability, until then the finder can also hand a vault over or give it a new identity that the other devices follow, as any of the owner's devices can ([section 10.5](#)). Until then, with the device's session, the finder can also write the account's keys, its vaults' identities and its device slots. Retiring the device ends every session bound to it, unless the finder removes the owner's device first. A reset by mail ends every session. So does a password change made from another device, unless the finder has stopped it. A device lost in the middle of a sign-in, before it registered, keeps a session bound to no device, which retirement does not end ([section 10.5](#)). The cached key is the operating system's to protect. It is accepted for this release.
- **L-24. A retired device's cached wrapping key is not wipeable from outside.** The application stops using the key 30 days after the password was last typed, judged by the device's own clock, and deletes it the next time it reads it after that. So someone holding the device who sets the clock back keeps it live. The store itself keeps no clock, so on a device where the application is not run again, the entry stays ([section 5.6](#)).
- **L-25. The master password and the kit words exist as ordinary strings, which nothing wipes.** They exist as such in the web view while they are typed or displayed, and in the application's own handling of each request that carries one, including a request queued to run later, until that request has finished. The login credential's derived value is held in a wiped type. Its hexadecimal form is an ordinary string too, in the body of each request that carries it and in the structures that build that request. Nothing wipes that string, and a diagnostic would print it whole ([section 5.5](#)).

- **L-26. A mailed reset is an account takeover, bounded only by an alert the thief can delete and a window the thief can outwait; with the kit as well it is unbounded.** The kit is the secret that must be kept apart from the mailbox. An undelivered mail keeps its token in the relay's database for as long as that token could still be redeemed. A row that the relay gave up on, and any undelivered row older than the longest token lifetime, keeps its audit shape — recipient, subject, attempts, error — and has its body blanked. A blanked row is never delivered.
- **L-27. A reset can be undone by the previous password's login credential for 24 hours, unless it was made within seven days of the account's last undo.** The undo is open to whoever leaked the password or holds that credential alone. Undoing the reset with that credential signs every device out. A reset made within seven days of the account's last undo is undone only by its cancel code. The reset's alert is mailed to the compromised mailbox, where the thief can delete it. The alert is exempt from the daily mail ceiling, so it cannot be suppressed by exhausting the day's allowance first. The undo restores the account's credential and keys, not content and not devices. What a session that held a vault's key wrote to that vault since the reset stays. So do devices registered in the window, which can fill the account's places and leave the owner's removed devices unable to register again. After the window the reset's holder can delete the account. That the devices survive is ticketed for after the beta opens. A reset made with the kit yields a session that holds a vault's key. One made without it does not, and writes into no vault that existed before it ([section 10.4](#)).
- **L-28. A self-hosted relay left on its default mail transport prints every mail to its log.** That transport sends nothing. It writes each message whole, recipient and body, to the relay's own log, for its operator to read. So a reset token, a cancel code or a verification code is there for whoever can read that log. That reader then holds what [section 10.8](#) says the mailbox holds. Deleting the account does not reach that log. The hosted relay sends mail instead. It is accepted for this release.
- **L-29. Kit replacement and password change replace what an honest relay serves; they cannot un-publish bytes a dishonest one kept.** Whoever held the old kit or the old wrapping key can still open the ciphertext they kept. A password change replaces only the wraps it reaches ([section 10.3](#)). An honest relay goes on serving a root the change did not reach under the wrapping key that root had. A listed root that none of the old wrapping key, the new one and the recovery kit opens likewise keeps the wrap it had. A root left on the wrap it had opens to whoever holds the account's email, the password that wrap was made under and a copy of the wrap. The remedy for a key believed compromised is to start a new vault identity, which mints a new root.
- **L-30. Metadata is not hidden.** The relay learns sizes (among them the size of every note, every operation and every file), timing, cadence, the shape of activity, and the correlations of [section 11.1](#). It holds in the clear the name each device was given. There is no padding and no timing obfuscation.
- **L-31. Register entries leak path length, and no object is padded or compressed.** Because an image's register entry is a fixed set of fields, a hash of the version it replaces when there is one, and the path, the relay learns the length of every file's path to the byte. This is an accepted property of the design rather than an outstanding defect. Nothing on any plane is

padded. Padding one field of one plane would close roughly ten bits on the shortest object in the set. The entry's format is fixed from this release, so padding it later would take a second version of the format that every app then reads beside the first.

- **L-32. Under the default configuration a registration reveals whether an address has an account.** Under the shipped default, in which registration is open and verification is off, a registration for an address that already has an account is refused with a distinct status. So the address's existence can be tested. That is an accepted limitation.
- **L-33. The email address is the salt, so precomputation per address is possible, and the relay learns the address.** A targeted attacker can compute a table for one address before obtaining anything.
- **L-34. The application checks a master password's length and nothing else: at least 12 characters and at most 128 bytes.** It makes no estimate of strength and refuses no common password. So what Argon2id's cost multiplies is whatever the person chose ([section 5.6](#)).
- **L-35. SVG is an admitted attachment format, and a peer's SVG is admitted exactly as this device's own.** SVG is an active-content format. It reaches the screen only as the `src` of an `<img>` element, which does not run script an image carries. No file that can touch asset bytes carries a markup sink that could route it anywhere else ([section 12](#)).
- **L-36. The relay's own credential hashing uses library defaults rather than parameters chosen for it.** They are the same as the client's. The credential they protect is itself a 256-bit derived value.
- **L-37. The Keychain entry on a Mac carries the platform's defaults.** The application sets nothing on the entry but its name. What those defaults allow has not been confirmed on a device ([section 5.5](#)). On iPhone the application sets the entry's class, the one Apple names "after first unlock, this device only". Such an entry can be read once the phone has been unlocked after it started. A backup restores it only to the same phone. It never transfers to a new one. When the application reads an entry of another class, such as one an earlier version wrote, it tries to replace it with one of this class. A replacement that fails can leave the old entry in place, or no entry. That an entry carries the class has not yet been read back on a phone.
- **L-38. There is no post-quantum consideration of the design's own cryptography, because it uses no public-key cryptography.** Every key is symmetric and every derived key is 256 bits, though a password-derived key is only as strong as the password. TLS does use public-key cryptography. A break of it would expose the metadata, the session tokens, the login credential and the two capabilities it carries, not content. The session tokens with the account capability are enough to write the account's keys, its vaults' identities and its device slots. The login credential alone is enough to sign in, change the password and delete the account.

13.3 Where detection stops

Most of these are described in [section 8.3](#); the first also in sections [6.2](#), [6.4](#) and [8.1](#), and the last in [section 9.2](#). Unless it says otherwise, each is accepted for this release.

- **L-39. A failed open or a divergence on a note's page raises no alert.** An operation that does not open, bytes that differ from what the device holds at a coordinate, or a compaction that does not open, which includes one served under a cut it was not sealed with, fail that page alone. The page stops syncing and shows as needing attention, and the vault's other pages go on. Nothing raises the sync integrity alert for such a failure, so a relay that tampers with one page reads as a sync failure of that page, not as tampering. A page that has failed this way also stops the device accusing the relay on two of the checks ([L-42](#) *A relay that serves exactly what is demanded while staying ahead is never alarmed on, and a device with a durably failed page of its own accuses nobody on the withheld-objects or split-by-page checks*). A relay can use a compaction this way to hold a page failed: it can discard its own copy of the page's history below a cut of its choosing and serve a compaction that does not open under it. The device keeps everything it holds and fetches the compaction again in every round. The page stays failed for as long as the relay serves it so. Where the device met the lie while merging the served compaction with its own, the page stays failed until the relay serves a cut beyond the one it lied with. Raising an alert for these failures, and the repeated fetch, are ticketed for after the beta opens.
- **L-40. A relay that refuses every sync round is never accused.** Only a round that asked the relay and drained its answer counts toward the alert. So a relay that answers every round with a lineage, root or key refusal, or makes it fail, raises no alert. The person sees an error or a decision to make, and the vault visibly stops syncing. This is the trade that stops an honest relay being accused after a vault's lineage restarts. The alert is bounded in completed rounds, not in time.
- **L-41. A relay that refuses the account capability signs every device out.** A relay can answer a device's writes with the account capability's refusal at will, and each device then signs itself out and asks for the master password ([section 5.2](#)). The detectors of [section 8.3](#) are dark on every device until it signs in again. An alert already raised stays raised, because the sign-out does not clear it. No file and no account state is lost. The relay gains nothing a relay refusing everything lacks.
- **L-42. A relay that serves exactly what is demanded while staying ahead is never alarmed on, and a device with a durably failed page of its own accuses nobody on the withheld-objects or split-by-page checks.** The sync integrity alert has a recorded false negative and a suppression. The recorded false negative is the accepted residual: a relay that always serves exactly what was demanded, while staying one step ahead, is never alarmed on. The suppression is that a device with a page of its own that has durably failed to sync accuses no relay on the withheld-objects or split-by-page checks, because it has a local cause for being short. It logs the accusation it held back, and the vault shows that page as needing attention instead. A relay that can keep one such page failing keeps those checks from accusing it.
- **L-43. The split-by-page check never speaks at an author that never converges.** A relay that withholds a page from a sibling, and answers the sibling's demand with other objects of the same author, enough to cover the count demanded, satisfies the demand. The substitution goes uncaught, because the frontier check runs only while a demand stands, and the author of the withheld page is itself served honestly, so it never takes the converged reading the split-by-page check needs.

- **L-44. The split-by-page check waits while the page list keeps changing.** The check compares only devices at exactly the same registry point. So, while the vault's pages keep being created, renamed or deleted, a relay that serves two devices points one apart delays the comparison until the changes pause. This costs detection time, not soundness.
- **L-45. Rounds that end early can make the split-by-page check stop asking.** The check drops a standing question after three readings in which the two devices had no converged pair to compare. A round that a key or lineage decision ends before the check runs counts as one such reading, although the check never ran in it, so three such rounds make the check drop its question. A relay that ends rounds that way can make a device stop asking. The error runs toward a missed alert, never a false one. It is ticketed for after the beta opens.
- **L-46. A heartbeat hidden by a frozen device list reads as staleness.** The hidden-heartbeat check needs a sibling's read to be stamped at or after this device's publish. A relay that stamps this device's publishes honestly, and goes on serving a sibling a device list read from before them, is not caught by that check. The sibling sees this device only as not heard from lately.
- **L-47. A raised sync integrity alert can be cleared by a heartbeat that says nothing of the vault.** The alert clears after a round that heard a sibling and left no demand or latch standing. A demand for withheld objects, or for a differing registry, no longer stands once it has raised the alert. A sibling's heartbeat that opens and carries no entry for this vault counts as heard and asks nothing. Such a heartbeat is an old one from before the sibling joined the vault, expired or not, or one from a sibling whose latest round for the vault failed. So an alert those two checks raised, for withheld objects or for a registry that differs, clears in the next round in which such a heartbeat opens, where no other demand or latch stands, with the objects still withheld. An alert the split-by-page or hidden-heartbeat check raised keeps its own latch and is not cleared this way. The relay chooses which heartbeat it serves. The fix is in the application alone, and is ticketed for after the beta opens.
- **L-48. The hidden-heartbeat check tells publishes apart by the second.** A device can publish twice within one second, and the check knows a publish only by its second. A relay that serves a sibling the first of the two heartbeats, under a stamp later than the second, is not caught. The sibling attests a heartbeat from that second, and the check accepts it as the current one, until the device publishes again. The fix, each publish carrying a time later than the device's last, is in the application alone, and is ticketed for after the beta opens.
- **L-49. The relay can clear a raised sync integrity alert by dropping the device it accuses.** The device list is the relay's word. A device withdraws every demand, and every latch, that it holds against a sibling the relay's device list no longer names, which is how a retired device's demand is withdrawn. So a relay that removes the accused sibling from the list, and lets the device hear another sibling in agreement, clears the alert. The fix is ticketed for after the beta opens.
- **L-50. A password change blinds the check until the other devices return.** A master-password change rotates the attest key, so no sibling's heartbeat opens until that sibling publishes under the new key. The change also signs every other session out, so each sibling stays unheard until it is signed in again with the new password and syncs. Fork detection is dark for that interval, which is when an owner reacting to a suspected compromise most

wants it. A re-tune signs no one out, but it rotates the attest key too. It also clears every other session's account capability, so each sibling is refused at its next heartbeat and stays unheard until it signs in again ([section 10.3](#)). The fix is ticketed for after the beta opens.

- **L-51. The sync integrity alert is held in memory.** Quitting the application clears a raised alert and every count toward one. Against a relay that goes on withholding, the counts start again from nothing.
- **L-52. Two different image entries at one revision are caught only where a device already holds one.** A device never seals two different entries at one revision while its record of the highest revision it has drawn survives. That record lives in its account store, so an account store restored from a file-level backup, copied to a second machine, or caught by a power cut before the record was permanently written, can draw a revision already spent. A device that already holds an entry at that revision keeps it, drops the other and logs it. A device that holds neither takes whichever it is served first, and nothing is shown. These residuals are stated, not closed.

13.4 Left to the owner's comparison

- **L-53. Withholding and old views of notes read as lag.** A relay can withhold new objects from every device, or serve every device an old view, and each screen reads it as a slow network; it heals when an honest relay answers. A relay that withholds can leave a device reading as synced over an old view. Omission of a *suffix*, which is what withholding looks like, reads as stale, never wrong. Comparing two screens, as [section 8.3](#) describes, is how the owner notices.
- **L-54. The eternal total partition is undetectable in band by construction.** It is the case where two groups of devices never see each other again. It freezes each screen's view of the other at the fork date. The owner's own attention is its detector: the memory of having used both devices since.
- **L-55. A vault's name can be served old.** The sealed label is one slot, overwritten when the vault's name changes. A relay can go on serving an earlier name: it opens, and nothing notices.
- **L-56. Images have no freshness signal.** Heartbeats carry no image state. So a relay can serve a device an older revision of an image whose successor that device has not seen, or omit an image altogether, and both screens can read as synced, and nothing in band notices. Comparing the two devices' files directly is the check. A relay that tells a device an image's bytes are there, and then withholds them, does show: the device is owed them, and reads as not synced. Building freshness for images was weighed and declined for this release, in favour of reliability, and is to be revisited after the beta opens.

13.5 What ordinary use can still get wrong

These need no hostile party: an interruption, an unlucky order of events, or a file restored from a backup.

- **L-57. An image edit that never finishes uploading destroys the image.** When an image is edited in place, the relay drops the old version's bytes as soon as the edit is announced.

Every other device removes its copy of the old version on hearing of it. Until the upload finishes, the new version exists only on the editing device. If that device is lost, wiped or never finishes the upload, the image is gone everywhere. The editing device reads as not synced for the whole of that window, so this is a loss before the devices agree, not after. It is a stated limitation of this release, and first in the queue after the beta opens.

- **L-58. "Synced" speaks for the last sync round.** An image put in the vault's folder between rounds is not counted until a round finds it, so a device can read as synced for that interval. This is stated in the ruling on what "synced" means, and not closed.
- **L-59. An image a device can no longer upload can read as synced.** When a damaged account store is set aside and rebuilt, the device takes a new identity. An image it announced under the old one and never uploaded can then no longer be uploaded, because the relay accepts an upload only from the identity that wrote the revision. The device reads as not owing it. It needs a damaged account store, and is ticketed for after the beta opens.
- **L-60. An image changed in place with its size and modification time kept is not noticed.** The application re-reads a file only when its size or modification time moves. So an image already on the relay that is rewritten by a tool that keeps both stays changed on that one device while it reads as synced. A copy that preserves times, or a restore from a backup, is such a tool. Ordinary editing moves the modification time. It is ticketed for after the beta opens.
- **L-61. A sync store restored from an older copy can rank a page change below one already sent.** Every registry entry a device writes takes a clock above every entry it has written or received. But the clock is kept in the vault's sync store. A sync store restored from a copy taken earlier can give the device's next change to which pages exist a clock below one it sent after the copy was taken, and that change then loses to the older one. The residual is stated, not closed.
- **L-62. On a relay that does not require address verification, until the kit's seed reaches the relay, only the application keeps a vault from being published without its kit wrap.** The relay records that an account owes its first kit only when a verified registration creates it. It refuses a root without a kit wrap only from an account that owes a kit or holds one. A relay with verification off, which is how the relay ships, never records the debt. There, a vault published without a kit wrap is taken. A device that holds the kit and opens its root tries to publish the wrap after a later sign-in or unlock. Until one does, the kit does not open it. The hosted relay requires verification. Recording the debt for every account is ticketed for after the beta opens, before the self-hosted relay is released.
- **L-63. The recovery kit has rough edges when a publish fails or two devices race.** A device whose kit's seed did not reach the relay can show its words before it learns that another device's kit has taken the account's slot. So the person may save words that are not the account's. The device raises the gate again over the account's words before any root is wrapped under a kit. A reset made with words that open none of the account's vaults still makes those words the account's kit, and reports the vaults it could not open. Two devices recovering with the same kit at the same moment can leave one of them failing every key write until its local account state is cleared. Each is ticketed, and none is fixed in this release.

- L-64. A kit record that moves backwards on the relay leaves a device ahead of it.** The undo of a mailed reset puts the relay's kit record back to its generation before the reset. A relay restored from a backup, or a hostile one, can serve an older generation too. Only a reset posts the relay's generation plus one. A kit replacement and a password change post the device's own generation plus one. So, on a device whose generation is above the relay's, both are refused until that device resets. A reset can also post the relay's generation plus one at or below the generation of another device's own kit record. When that device next compares the two, it refuses a lower record as a rollback, after which its writes of a root's wrapped key fail. At an equal generation, that device takes the reset's record where its own does not open under its wrapping key, as after a reset to a different password, or was never published. Its writes then go on under the reset's kit. Where its own was published and still opens, it refuses a different seed as a contradiction. After that, its writes of a root's wrapped key are taken, wrapped under the kit it kept, which the account's kit on the relay does not open. Either refusal lets its sign-in carry on. No note is lost. It needs an undone reset, a restored relay or a hostile one, and is ticketed for after the beta opens.
- L-65. A kit replaced on another device just as a password change starts makes the change fail and, unless this device signs out, leaves it declared for up to 24 hours.** The change fails with the credential and the kit as they were. While it stays declared, it refuses other devices' key writes. It is ticketed for after the beta opens.
- L-66. A master-password change abandoned part-way has rough edges.** A device that still holds such a change, after the account has moved past it, recovers when the person signs out and in again. Until then, an unlock does not recover it, and running a change again there is refused. Where a vault the change re-wrapped has no root record on the device and a sync store it cannot read, the sign-in leaves the change in place. Unlocking with the abandoned change's new password adopts a vault key under a password the account never took, there or on any other of the account's devices holding a root the change re-wrapped on the relay. That unlock is the one the prompt that says the password was changed elsewhere asks for ([section 10.3](#)). The account's real password then fails to unlock those vaults until the change is run again to completion. Such a device is refused at its next write to the account's keys and signed out. The device that holds the change is then asked for the password from before it. Where the change was declared and nothing of it was yet staged on that device, it asks for the master password instead. Any other device asks for the master password, which the abandoned change never moved. After that sign-in, a vault the unlock re-seated stays locked until the change is run again. If the recovery kit is replaced between the abandoned change and a later change to a different password, some vault keys end up openable only with the replaced kit's words. That is why the application tells the person to keep an old kit as well. If those words are discarded, neither the account's password nor its current kit opens those vaults' copies on the relay, and no new device can join them. A change abandoned after it was declared also leaves the account's other devices told, once the declaration has ended, that the master password was changed on another device, which is not so, until each signs in again. This is because the key epoch moved when the change was declared, and the password did not. No note is lost in any of these, because each device keeps its files. They are ticketed for after the beta opens.

- **L-67. A device can be left unable to follow its vault to a new identity.** A device follows a new identity only under a certificate that lists the identity it holds ([section 7.3](#)). Three things can leave it without one. A vault that was superseded by a reset after a forgotten password with no usable kit points at the generation that replaced it. If the replacement's root is later retired for a new identity, that generation is gone, and a device still on the superseded vault finds nothing to walk to. A relay that withholds a vault's ancestry from a device that joins leaves that device with a chain that starts where it joined, and a new identity it later mints does not list the earlier ones. And a certificate's chain keeps only its newest 64 roots, so a device more than 64 identities behind is not listed in it. In each case the device adopts nothing, as [section 7.3](#) requires, and tells the person that the synced copy of the vault is gone, which is not so. Of the remedies it names, replacing the synced copy with this device's own reports success and changes nothing, and starting a new vault identity is refused. The way out, which the panel does not offer, is to recover the vault into a new, empty folder. No note is lost, because each device keeps its files. With an honest relay it takes a forgotten password, a lost recovery kit and a later change of identity. The 64-root chain is not reached there, because an honest relay refuses an account's 65th root first ([L-8](#)). It is ticketed for after the beta opens.
- **L-68. Starting a new vault identity on an account at its limit of vaults does not finish.** The old identity is retired first, and the request that publishes the new one and founds its first generation is then refused at the limit. The device keeps the unfinished change and the vault waits, syncing nothing, until the account has room. The message it shows names the limit on vault identities, where the limit met is the one on vaults. No note is lost. It is ticketed for after the beta opens.
- **L-69. An interrupted vault reset cannot finish once another device's reset of the same vault has won.** Retrying the reset and a manual rejoin both meet the same refusal. No note is lost, because the files stay on the device. It is ticketed for after the beta opens.
- **L-70. A relay that has lost a generation's record puts a decision to the device.** A relay restored from a backup, or the undo of a reset inside whose window a generation was made, can leave a device holding a generation the relay no longer has a record of. Only the three requests of [section 5.3](#) make such a record. So the device's next request into it is refused as naming the wrong root, and the device puts the decision to the person. No note is lost. It is stated, not closed.
- **L-71. A device left at a relay's old address by another device's move is stranded by its next sign-in.** Moving the credential to a relay's new address changes no session, so the account's other devices go on working at the old address. But any refusal that ends in a sign-in, the account capability's among them, leaves such a device unable to sign in there until a person points it at the new address. No note is lost. It is ticketed for after the beta opens.
- **L-72. An interrupted kit replacement leaves two live kits until a later sweep converges them.** Each kit opens some of the roots, with no durable marker, until the next sweep converges them.
- **L-73. A forgotten password on an account with no kit starts every vault's relay copy afresh.** The files are untouched. An account with no kit is one whose first kit failed to settle.

- **L-74. A password or email that contains a character Unicode has not yet assigned may derive a different key after an update.** Normalisation and lower-casing are fixed for every character Unicode has assigned. A code point unassigned in the Unicode version the application was built with is left as it is, and a later version can give it a decomposition or a case mapping. The same keystrokes would then derive a different key, and the owner would be told the password is wrong. No such character is on an ordinary keyboard. Refusing such characters when a password is set is ticketed for after the beta opens.

14. Reporting a weakness

Write to security@slateoutliner.com. Say what you did and what you saw; a proof of concept helps. The address is also published at the website's `/well-known/security.txt`. There is no reward programme.

Appendix A. Glossary

Term	Meaning
Prevented, detected, neither	The three tiers of section 1.1 . Prevented : made impossible by cryptography for anyone without the account's keys, the relay included. Detected : attempted by a hostile relay and noticed by a device under the conditions of section 8.3 . Neither : indistinguishable from a slow network, and left to the owner's own comparison.
Synced	A vault's state on a device when, as of that device's last sync round, it has nothing left to send and nothing left to fetch, for notes and images alike, images kept on the device on purpose aside. Section 9.3 gives the whole promise and its exceptions.
Master password	The one secret you type. Never stored, never sent.
Master key	The 32-byte Argon2id output of the password and the email salt. The parent of the three account keys.
Argon2id parameter set	The memory, iterations and lanes a credential is stretched under (section 4).
Re-tune	A sign-in that succeeds under an older Argon2id parameter set, and moves the credential to the newest under the same password (section 4 , section 10.3).
Canonical origin	The spelling of a relay's address that the login credential is made for: <code>scheme://host:port</code> , the scheme and host lower-cased and the port always written out, and nothing of the path (section 5.2).
Auth-hash	The HKDF branch presented to the relay as the login credential (section 5.2). L-18 <i>The login credential is bound to an origin, with two exceptions.</i>
Wrapping key	The HKDF branch that never leaves the device (section 5.2). Elsewhere called a key-encryption key.
Attest key	The HKDF branch that seals device heartbeats.
Account identifier	Sixteen random bytes the relay mints for an account (section 10.7). The relay's word, not a credential.

Term	Meaning
Vault	A folder that syncs.
Root, root key	A vault's permanent random identifier and its permanent random 32-byte key, minted once, at a first enable or at a restart that gives the vault a new root (section 7.2).
Generation	One lifetime of a vault's history under its root, with its own random identifier and a key derived from the root key (section 5.3).
Generation key	The key that seals every content object of one generation. Elsewhere called a data-encryption key.
Write capability	Thirty-two bytes derived from a generation's key under a label of its own (section 5.3). A bearer value for the relay, not a key: it opens nothing, and no device trusts it.
Account capability	Thirty-two bytes derived from the wrapping key under a label of its own followed by the relay's canonical origin (section 5.2). A bearer value for the relay, not a key: it opens nothing, and no device trusts it.
Account capability's digest	The SHA-256 of the account capability, which the relay keeps on each session (section 5.2).
Refused state	A device's state after the relay refused its account capability: signed out, its keys dropped, its account state kept and the reason recorded, until a sign-in with the password (section 5.2).
Hand-over key, lineage key	Two keys derived from a root key under labels of their own: the first seals that root's hand-over records, the second the lineage certificates that name that root as a successor.
Wrap, seal	The same AEAD operation; a wrap encrypts a key with a key, a seal encrypts data with a key.
Associated data	Cleartext bound into a seal: a domain string and a coordinate. Authenticated, not encrypted.
Coordinate	Where a sealed object lives: for content, (generation, page, actor, seq), the first field being what the code calls the vault id.
Actor	A device's random 64-bit identifier, stable until the person starts sync over on that device.
Envelope	The relay-visible unit of a note's operation or a page registry entry: a coordinate, the kind of object, a nonce and a ciphertext.
Page, registry	A page is one markdown file's history; the registry is the vault's append-only list of pages and their paths, itself sealed.
Snapshot, compaction, cut	A page's two overwrite slots, outside the streams and outside the Merkle forest. The snapshot holds the text the page's history was seeded from; the compaction holds the page's merged state up to its cut, the position in each device's stream it covers (section 6.2).
Registry point	The position a device has reached in a vault's page registry (section 8.3).
Register clock	The clock a page registry entry carries, which orders two entries for one page (section 6.4 and L-61 <i>A sync store restored from an older copy can rank a page change below one already sent</i>).

Term	Meaning
Epoch	A per-root counter that moves on every lineage change and is bound into the hand-over record. Distinct from the per-account key epoch , which moves on every credential change.
Hand-over	A restart that keeps the root and moves its pointer to a fresh generation in one relay transaction, under a hand-over record sealed with the root's hand-over key.
Retire, successor	The two halves of giving a vault a new root: clearing the old root's pointer, then publishing the root that replaces it (section 7.2).
Supersede	Pointing a generation that is kept, after the reset of section 10.4 , at the vault that replaced it, under a lineage certificate.
Lineage certificate	The tuple (predecessor, successor, chain) sealed under the successor's lineage key (section 7.3).
Tombstone	A relay-side mark on a purged generation that answers "gone" to its owner for ever.
Transition marker	A durable local record of a lineage change in progress (section 7.3).
Recovery kit, kit seed, kit key	Twenty-four words or a file, both renderings of one random 32-byte seed; the kit key derives from the seed and wraps each root key a second time.
Kit generation	A per-account counter bound into the wrapped seed's associated data; moves whenever the recovery-kit record's bytes move (section 5.4).
Heartbeat	A device's sealed attestation, in an overwrite slot the relay serves to siblings (section 8.3).
Frontier, version vector	Per actor, how far a device has applied that actor's stream; a version vector is the set of those positions for one page.
Page map	A device's map from each live page of a vault to that page's frontier.
Frontier check, converged reading	The frontier check asks whether a device holds exactly the relay's frontier on every live page; a converged reading is a page map taken straight after one that passed.
Demand	A device's request that the relay produce objects, or registry entries, a sibling's heartbeat shows exist and this device lacks.
Completed round	A sync round whose final attempt ran the full repair sweep, drained it to the end and succeeded (section 8.3). Also called a counted round.
Sync integrity alert	The latched alert the checks of section 8.3 raise, also called the tamper alarm.
Device list	The relay's list of an account's devices (section 8.3). The relay's word, not a sealed record.
Merkle forest	One append-only hash tree per actor over sealed envelope bytes (section 8.1).
Asset, revision	A binary file's random identifier and the counter-plus-actor clock that orders its overwrites (sections 9.1 and 9.2). Images are the only binary files the plane admits today.
Secret store	The operating system's credential store: Secret Service on Linux, the Keychain on Apple platforms.

Appendix B. Every domain string and label

HKDF labels: `slate:kdf:v2:auth-hash` (followed by the relay's canonical origin), `slate:kdf:v1:wrapping-key`, `slate:kdf:v1:attest`, `slate:kdf:v1:generation` (followed by the 16-byte generation identifier), `slate:kdf:v1:handover`, `slate:kdf:v1:lineage`, `slate:kdf:v1:write-capability`, `slate:kdf:v1:account-write` (followed by the relay's canonical origin), `slate:kdf:v1:recovery-kit`. The email-salt digest prefix: `slate:kdf:v1:email-salt:`.

The write capability travels in the request header `slate-vault-write`, and the account capability in `slate-account-write`, each as exactly 64 lower-case hexadecimal characters.

Associated-data domains: `slate:op:`, `slate:snapshot:`, `slate:register:`, `slate:compaction:`, `slate:asset-register:`, `slate:asset-bytes:`, `slate:label:`, `slate:dek:`, `slate:root:`, `slate:recovery-root:`, `slate:recovery-seed:`, `slate:handover:`, `slate:lineage-root:`, `slate:lineage-vault:`, `slate:attest:`.

Merkle forest hash prefixes: `slate:leaf:`, `slate:node:`, and `slate:root:empty` for the root of an empty tree.

Sealed-record format versions, each checked before a key is used: wrapped generation key 2, wrapped root 1, wrapped root under the kit 1, wrapped kit seed 1, hand-over record 1, lineage certificate 1, sealed label 1. Formats inside a seal, checked once it opens: heartbeat 2, image register entry 2, the plaintext of a note's operation, of a page registry entry and of a compaction 1, with a page registry entry's own format 1 after that byte. A snapshot's plaintext is the page's seed text, which carries no version. Formats around a seal, on the wire between device and relay: envelope 2; the upload container, the read container and its part frame 1 each; an image's part frame 1. Kit file 1. Sealing protocol version 2. Relay API version 1.

Appendix C. Changelog

- **1, 3 October 2026.** The first published release.
- **2, 10 October 2026.** On iPhone the application keeps the refresh token in the Keychain and sets the Keychain class of it and of the cached wrapping key: sections 5.5 and 10.1, and L-22 and L-37, whose headline now names the Mac. Section 10.3 says what the prompt that says the password was changed elsewhere asks for and what ends it, and L-66 names the account's other devices. L-62 says when a root taken without its kit wrap gets one, and L-66 a case that keeps a change the account moved past in place at sign-in.

Appendix D. References

- A. Biryukov, D. Dinu, D. Khovratovich, S. Josefsson. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*. RFC 9106, 2021.
- H. Krawczyk, P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869, 2010.
- Y. Nir, A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439, 2018.

- S. Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. IRTF CFRG Internet-Draft, draft-irtf-cfrg-xchacha.
- OWASP. *Password Storage Cheat Sheet*, the Argon2id recommendation.
- M. Palatinus, P. Rusnak, A. Voisine, S. Rowe. *BIP-39: Mnemonic code for generating deterministic keys*. 2013.
- D. Mazières, D. Shasha. *Building secure file systems out of Byzantine storage*. PODC 2002. The fork-consistency bound [section 8.2](#) relies on.
- J. Li, M. Krohn, D. Mazières, D. Shasha. *Secure Untrusted Data Repository (SUNDR)*. OSDI 2004.
- M. Kleppmann, D. P. Mulligan, V. B. F. Gomes, A. R. Beresford. *A highly-available move operation for replicated trees*. IEEE TPDS, 2022.
- M. Weidner, M. Kleppmann. *The Art of the Fugue: Minimizing Interleaving in Collaborative Text Editing*. 2023.
- E. Foudil, Y. Shafranovich. *A File Format to Aid in Security Vulnerability Disclosure*. RFC 9116, 2022.
- AgileBits. *1Password Security Design*. Standard Notes. *Encryption specification 004*. Signal. *The X3DH Key Agreement Protocol*. The three documents whose order this one follows.