Comutability

Secure Net

What the protocol is, how it is built, how it works and how to use it. Written for people who build software and for people who don't.

Read this first

Secure Net is early software. It has not been independently security-audited, its newest parts have not been examined by a human cryptographer, and no packaged release has been made. Do not rely on it where your safety depends on it. This guide describes the design honestly, including what is unfinished.

This edition	15 November 2026
Describes	Secure Net 0.4.0 (the main branch), plus proposals under review, each marked Proposed
Source of truth	github.com/Comutability/secure-net. Where this guide and the repository differ, the repository is right.
Written by	Prepared with an AI assistant (Claude) at the founder's request. Michael J Burgess is responsible for it.
Licence	The protocol's code and specification are AGPL-3.0-only.
Contact	contact@comutability.org · comutability.org

Contents

1 Secure Net in one page

2 The three rules it rests on

3 Keys: who you are

4 One exact encoding, and signed records

5 Sites talking to each other

6 How one message travels

7 Sessions: agreeing and changing keys

8 Communities that outlive their host

9 Debates, votes and private tokens

10 Resisting a quantum computer

11 What each party can still see

12 Proposed: the social protocol, sessions version 2 and groups

13 How to use it

14 How the protocol changes

15 Limits, and what is not done

16 Glossary and where to find things

How to read this guide

Each section starts in plain words. Boxes like these carry the two other voices:

In plain words

The idea, without jargon. If you only read these, you will still understand what Secure Net does and what it does not.

For builders

The exact construction: algorithms, byte formats and the part of the specification (`CVP-nnn`) to read next.

Two labels say how settled something is. In the protocol means it is in the main branch with reference code and tests. Proposed means it is an open pull request: written and tested, but not accepted. Neither label means "reviewed" or "safe".

1 Secure Net in one page

Secure Net is a set of rules that lets people talk privately across websites run by different people, so that no single company owns the conversation and the websites carrying it cannot read it.

In plain words

Think of the post. You write a letter, seal it, and hand it to a postal service. Any honest postal service can carry it, and you can switch to another one. Secure Net is the agreement about envelopes, addresses and seals. The "postal services" are websites that anyone can run. The seal is made on your own phone or computer, and only the people you wrote to can break it.

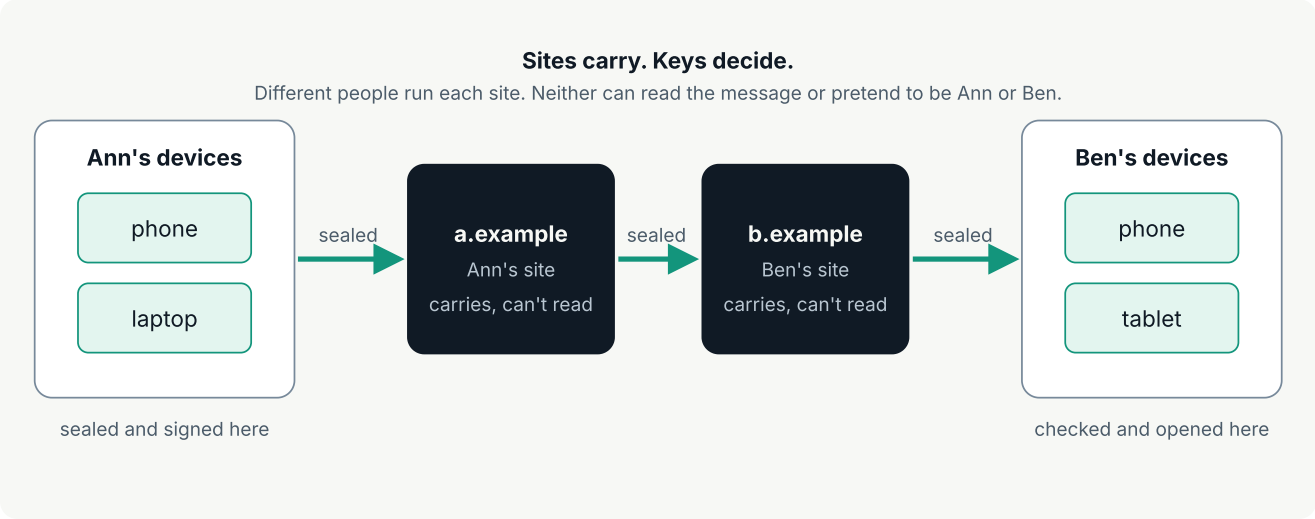


Figure 1. The shape of everything in Secure Net: sealing and checking happen on people's own devices, and sites only carry.

What it covers

Area	What it gives you	Status
Identity	An account is a chain of keys you hold. A site cannot pretend to be you.	In the protocol
End-to-end messages	Sealed on your device for the devices you are writing to, with keys that keep changing.	In the protocol
Sites that talk to each other	Anyone can run a site; sites prove which site they are, and what they pass each other is sealed.	In the protocol
Communities	A signed history that approved sites copy, so a community survives its host going away.	In the protocol
Decisions	Proposals, structured debates, public and secret ballots that members can recount.	In the protocol
Consent and extensions	Time-limited permissions you can withdraw; a way for sites to agree on features.	In the protocol

Area	What it gives you	Status
Social protocol	Posts, follows and profiles that move with you between sites.	Proposed
Sessions version 2, sealed sending, groups	Hides who is writing to whom from sites; groups that lock out a removed member at once.	Proposed

One application uses Secure Net so far: **Commonveil**, a private messaging and communities app with a server you can host yourself. Anyone may build another, and none gets special treatment.

---

2 The three rules it rests on

Everything else in this guide follows from three rules. If you remember only these, you can reason about most of the design yourself.

1. **Keys say who people are; sites only carry.** Each account is a chain of key statements, each signed by the one before. Each device is certified by the account. Each message is signed by the device that sent it and sealed for each device that should read it. People's own apps check all of this. A site that lies about someone's keys is caught.
2. **Sites prove which site they are.** Each site has its own key. Every request from one site to another is signed, dated and usable only once.
3. **Only shared formats, never shared trust.** Two sites agree on exact byte formats and on which features both speak. Neither has to trust the other's database, software or owner.

> **In plain words**
> You never have to believe a website when it tells you who someone is. Your own app checks a chain of signatures that only that person could have made. If the chain breaks, your app warns you and gives you a short number to compare with the other person, in person or by phone.

How it is built

Secure Net is a library and a written specification. The reference library is plain JavaScript modules that run in browsers, Node.js 20 or newer, and Deno. It needs no particular host, database or company.

Module	What it does	Specification
`core/`	One exact encoding and the signed-record envelope everything else uses	CVP-002
`crypto/`, `identity/`	Account key chains, device certificates, recovery phrases, personas, moving between sites	CVP-001, CVP-003
`session/`	End-to-end sessions: handshake, ratchet, several devices, sender keys, new devices	CVP-016

Module	What it does	Specification
`wire/`	Signed requests between sites, the sealed relay, agreeing on features	CVP-005, 008, 015
`commons/`	A community's signed, hash-chained history and its mirrors	CVP-011
`governance/`, `tokens/`	Motions, debates, ballots; private one-use tokens	CVP-012, CVP-013
`consent/`, `extensions/`	Revocable permissions; extension and theme packages	CVP-014, CVP-008

For builders

The reference library depends on four small cryptography packages from the noble and scure families (`@noble/curves` , `@noble/hashes` , `@noble/post-quantum` , `@scure/bip39`) and the platform's WebCrypto for AES-256-GCM. A Rust implementation, compiled to WebAssembly so the same code runs in apps and on servers, is in progress (pull request 7): it reproduces the test vectors in `spec/vectors/` and talks to the JavaScript message by message.

3 Keys: who you are

In Secure Net you are not a username on someone's server. You are a key that you hold, and everything else hangs from it.

In plain words

A key here is a pair: a private half that never leaves your device, and a public half you give to everyone. With the private half you can sign things; anyone with the public half can check your signature but cannot forge it. Your *account key* vouches for each of your devices. Each device has its own keys, so losing a phone does not mean losing your identity.

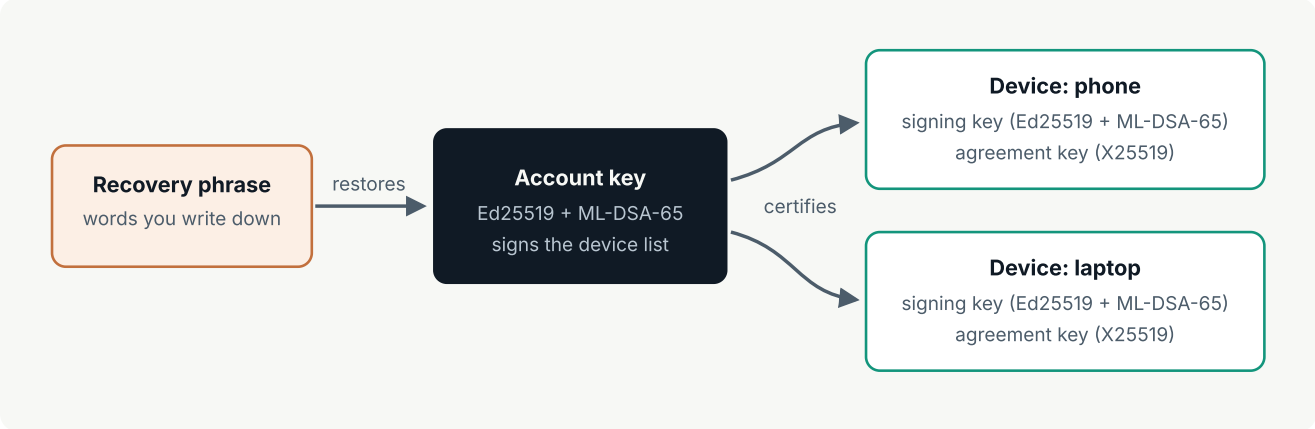


Figure 2. One account, several devices. The account key signs a numbered list of devices; a newer list replaces an older one.

Changing keys without losing your name

Keys sometimes have to change: a device is lost, or you move to a different site. Secure Net handles this with a **key chain**: each new key statement is signed by the key before it. Anyone who knew your old key can follow the chain to your new one and see that it really is you. If a key ever changes *without* a valid chain, apps warn you and show a **safety number**: a short number that two people can compare to be sure nobody is in the middle.

Personas and moving

You can show a different face in each community (a *persona*). And because your identity is a key rather than an address on a server, you can move to another site with a signed statement, and people who follow the chain come with you.

For builders

A quantum-resistant public key is written as the base64url Ed25519 key, a full stop, then the base64url ML-DSA-65 key (1,952 bytes). A key's id (`kid`) is the first 22 base64url characters of the SHA-256 of the key. Device lists are signed records of kind `session.devices` with a sequence number; a receiver refuses a list with a lower number than one it has seen. The safety number is a short run of digits taken from the SHA-256 of both account keys, sorted, under the label `secure-net/session/v1/safety` . See CVP-001, CVP-002 section 2 and CVP-016 "Devices".

---

4 One exact encoding, and signed records

A signature is made over bytes. If two programs turn the same information into different bytes, a signature made by one will not check on the other. So the first thing Secure Net fixes is exactly how to write things down.

In plain words

Imagine two people signing "the same" contract, but one copy has an extra space on page three. Are the signatures on the same document? Secure Net removes the question: there is exactly one way to write any piece of information, down to the last character.

Canonical JSON

Secure Net writes data as JSON, with strict rules: object members sorted by key, no spaces, whole numbers only (within the range every programming language handles exactly), and one way to escape characters. A program must *refuse* anything that could be written two ways, such as a fraction, rather than quietly tidy it.

The signed record

Almost everything that matters is a **signed record**: a kind, a body, and a signature. Ledger blocks, ballots, device lists and prekeys are all this one shape.

```
{
  "kind": "session.devices",              // what sort of record this is
  "body": { "account": "...", "seq": 3, "devices": [ ... ] },
  "sig":  { "kid": "Qm3...", "ed": "...", "pq": "..." }   // Ed25519 and ML-DSA-65
}
```

For builders

The bytes signed are the UTF-8 of `secure-net/<kind>/v1`, one newline byte, then the canonical JSON of the body. Putting the kind in front means a signature on one kind of record can never be passed off as another kind. When a key is quantum-resistant, *both* signatures must be present and both must verify. Verifiers check that the key id matches the key, and the key, not the signature, decides which signatures are required. Canonical JSON is the subset of RFC 8785 (JCS) whose output is identical in every language. Read CVP-002, and check your implementation against `spec/vectors/`.

---

5 Sites talking to each other

A site is any server that carries messages for its people. Anyone can run one. Sites do not need to trust each other, only to prove which site they are.

Signed requests

Each site publishes its key at a well-known address. When `a.example` asks `b.example` for something, the request is signed over what is being asked, both site names, the time, a one-off number and a digest of the contents. A request more than five minutes old, or whose one-off number has been seen before, is refused. So a recording of a request cannot be replayed, and nobody can speak for a site without its key.

The sealed relay

The contents one site passes another are sealed for the receiving site alone. Someone watching the network between them sees that two sites are talking, but not who is writing to whom.

Agreeing on features

Sites announce which features they speak and at which version. Two sites use what both understand. A newer site keeps speaking the older version of a feature it still announces, so sites can upgrade at different times.

For builders

The site document is served at `/.well-known/commonveil` and lists the site's key and endpoints. The relay seal combines ECDH on P-256 with ML-KEM-768 (HKDF-SHA-256, AES-256-GCM), and relay signatures combine Ed25519 with ML-DSA-65; both must verify (CVP-015). Sites must refuse to connect to private or loopback addresses when following another site's instructions. See CVP-005, CVP-008 and CVP-015; Commonveil's self-hosted server is the reference site.

6 How one message travels

Ann is on one site and Ben is on another. Different people run each site. Here is what happens when Ann writes to Ben.

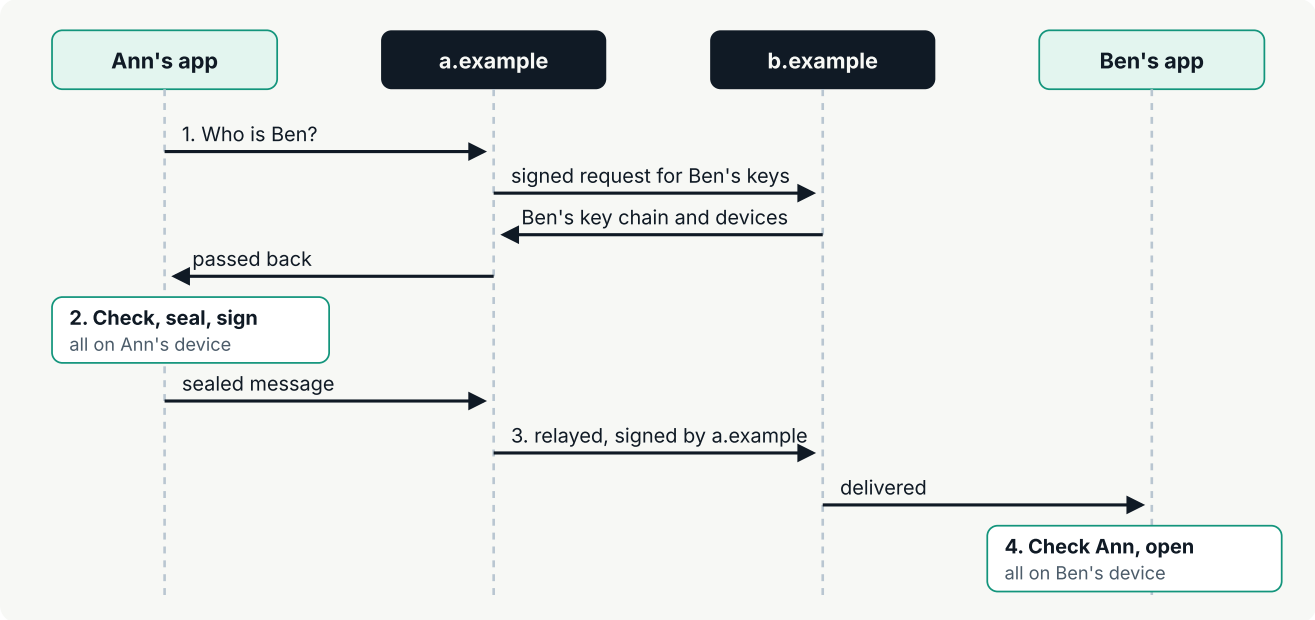


Figure 3. The journey of one message. The two boxes outlined in green are the only places the message exists unsealed.

1. **Find.** Ann's app asks her site to look up Ben. Her site asks Ben's site for his keys with a signed request. Ann's app then checks Ben's key chain *itself*; it does not take either site's word.
2. **Seal and sign.** Ann's app seals the message for each of Ben's devices (and her own other devices) and signs it. Only then does it leave her device.
3. **Relay.** Ann's site passes the sealed message to Ben's site, in a request sealed and signed between the two sites.
4. **Check and open.** Ben's devices check Ann's signature against her key history, then open the message.

What the sites still learn

Neither site can read the message. Either can refuse or delay it. And both learn that Ann and Ben talked, when, and roughly how much. Encryption hides what you say, not that you spoke. Section 11 sets this out in full, and section 12 describes a proposal that hides more.

7 Sessions: agreeing and changing keys

Sealing a message needs a secret that only the two devices share. A session is how two devices agree that secret without ever having met, and how they keep changing it as they talk. In the protocol

In plain words

Ben may be offline when Ann first writes. So each of Ben's devices leaves a small stack of single-use public keys with his site, like stamped, self-addressed envelopes. Ann takes one, combines it with her own keys, and both ends arrive at the same secret, which never travels. After that, every message uses a new key, and each reply reshuffles the keys again. Someone who steals one key gets one message. Someone who steals a whole device today cannot read yesterday's messages, and loses their grip once the conversation moves on.

The handshake

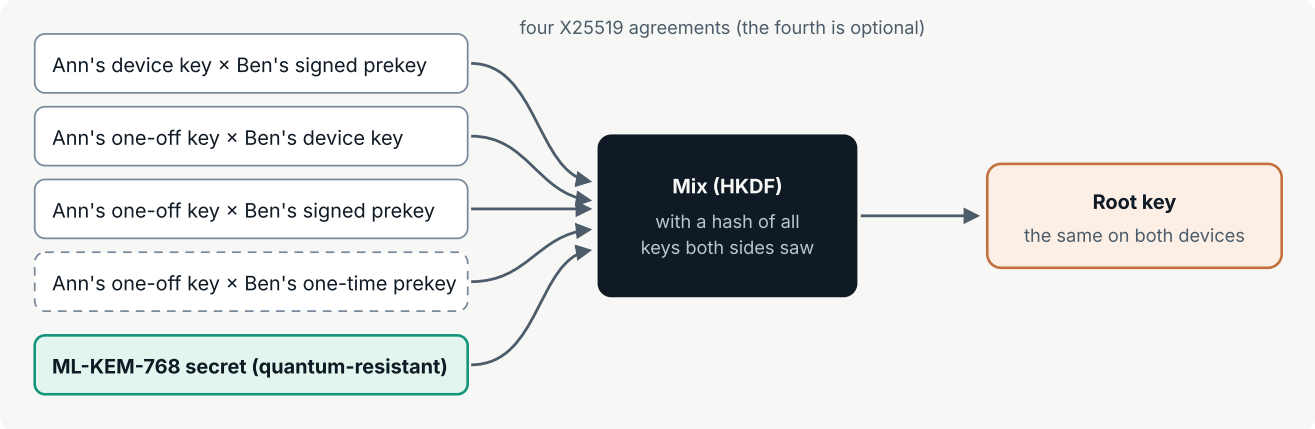


Figure 4. Two kinds of mathematics are mixed. An attacker has to break both: the established kind (X25519) and the quantum-resistant kind (ML-KEM-768).

The ratchet

From the root key the two devices derive a chain of message keys. Each message uses the next key and then throws it away. Each time the direction of conversation turns, both sides mix in fresh key material, so the chain heals after a compromise. This is the *double ratchet*, the design used by Signal, with a quantum-resistant step added.

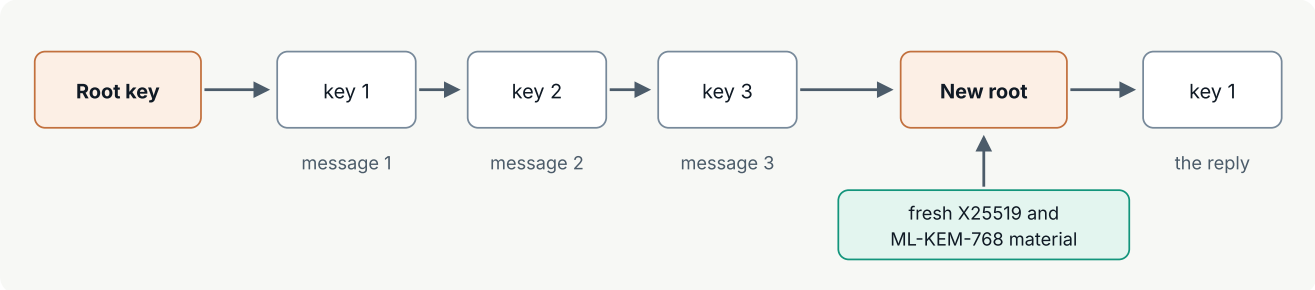


Figure 5. Keys only move forwards. Used keys are deleted, so an old message cannot be opened later even with the device in hand.

Several devices, audiences and new devices

- **Several devices.** A message to Ben is sealed separately for each device on his signed device list, and for the sender's own other devices.
- **Sender keys.** To write to many readers at once, a device makes one key for its own messages and hands it to each reader over their one-to-one sessions.
- **A new device.** When you add a device, an existing one can pass it your message history, sealed for the new device.

> **For builders**
>
> The handshake follows the shape of Signal's PQXDH: `SK = HKDF-SHA-256(0xFF×32 ‖ DH1 ‖ DH2 ‖ DH3 [‖ DH4] ‖ SS)` with a transcript hash in the HKDF info. Prekeys are signed records; ML-KEM-768 prekeys come as one-time keys plus a signed last-resort key. The ratchet's root step is `HKDF(ikm = DH ‖ SS, salt = RK)`; message keys come from an HMAC-SHA-256 chain; messages are AES-256-GCM with the transcript hash and header as associated data. An all-zero X25519 output is refused. Limits: 1,000 skipped messages per chain, 2,000 stored keys. Read CVP-016 and reproduce `spec/vectors/cvp-016-v1.json`.

---

8 Communities that outlive their host

A community usually lives on one site. If that site disappears, so does the community. Secure Net gives a community a history of its own that other sites can hold, so it can carry on. In the protocol

> **In plain words**
>
> The community keeps a minute book. Every entry is signed by someone allowed to make it and refers to the entry before, so pages cannot be torn out or swapped. Approved sites keep copies. If the home site goes down, the next site on an agreed list takes over, using its copy. A copy-holder can check everything but cannot read private content or change who is in charge.

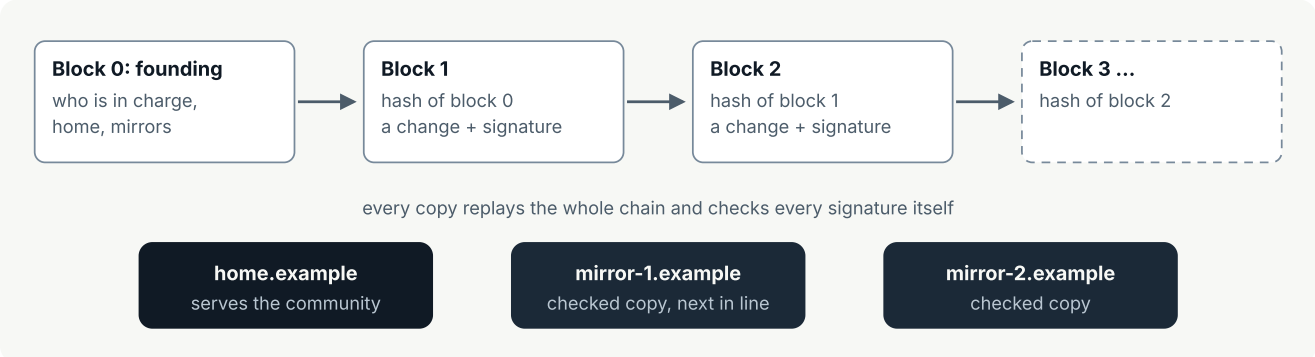


Figure 6. The commons ledger. It is a signed, hash-chained history held by a few named sites. It is not a blockchain: there is no mining, no token and no global network.

For builders

Blocks are signed records. A block's id is the SHA-256 of exactly what was signed, and each block names the id of the one before. The founding block names the authority keys, the home site, the replicas and a succession order. `replay(blocks)` rebuilds the state and rejects any block whose signer lacked the capability at that point in history. `chooseHost` works out which site should be serving: the home site if it can be reached, otherwise (unless the community has switched this off) the first reachable site in the succession order. See CVP-011.

---

9 Debates, votes and private tokens

Communities need to decide things. Secure Net includes proposals, structured debate and ballots whose results any member can recount. In the protocol

In plain words

Someone puts forward a motion. Members debate it in a structured way, then vote. A public ballot shows who voted how. A secret ballot does not, and yet every member can still check that only members voted, that nobody voted twice, and that the count is right. A passed motion can authorise a change in the community's minute book (section 8).

How a secret ballot can be checked

1. Before the vote, each member asks the community for a single-use **token**. The community signs it "blind": it can tell that it issued one token to each member, but not which token went to whom.
2. To vote, a member hands in their ballot with their token, not their name.
3. Anyone can check that every counted ballot carries a valid, unused token. Nobody, including the community's own site, can link a ballot to a person.

The same tokens serve other needs where you must prove you are entitled to do something once without saying who you are: anonymous reports, and rate limits that don't track people.

Consent you can withdraw

A *consent capsule* is a signed, specific, time-limited permission: this person may do this thing with this data until this date. It can be withdrawn, and it says plainly what withdrawing can and cannot undo (a copy someone already made cannot be unmade).

For builders

Tokens are a verifiable oblivious pseudorandom function (RFC 9497 VOPRF) over P-256. Motions, debate entries and ballots are signed records; tallies are recomputed from the records, never taken from a server. Note that P-256 tokens are *not* quantum-resistant (section 10). See CVP-012, CVP-013 and CVP-014.

10 Resisting a quantum computer

A large quantum computer, if one is built, would break the mathematics that protects most of today's internet. Messages recorded now could be opened then. Secure Net is designed so that this does not work against it.

In plain words

Secure Net uses two locks of different kinds on the same door: one of the established kind, which has been attacked for decades and held, and one of a newer kind designed to resist quantum computers. To get in, an attacker must pick both. If the new kind turns out to have a flaw, the old kind still holds; if a quantum computer arrives, the new kind still holds.

Job	Established	Quantum-resistant	Where
Agreeing a secret	X25519 (sessions), P-256 (the relay)	ML-KEM-768 (FIPS 203)	Sessions, the sealed relay
Signing	Ed25519	ML-DSA-65 (FIPS 204)	Accounts, devices, sites, records
Sealing content	AES-256-GCM (already considered strong enough against quantum attack)		Messages
Hashing and key derivation	SHA-256, SHA-512, HMAC, HKDF		Everywhere

What does not resist a quantum computer yet

Private voting tokens (section 9) use P-256 alone. Commonveil, the application, uses the quantum-resistant sessions only as an opt-in, experimental extra layer on direct messages; its ordinary message signatures are not yet quantum-resistant. The specification part CVP-015 lists, feature by feature, what is and is not protected.

---

11 What each party can still see

No system hides everything. Secure Net's rule is that an app must show people what remains visible, and must not promise more.

Who	What you say	Who talks to whom	Community membership	Your network address
Your own site	No	Yes, for your conversations	Yours	Yes
The other person's site	No	Yes, for conversations with its people	Of communities it hosts	Your site's, not yours

Who	What you say	Who talks to whom	Community membership	Your network address
A community's home or mirror	No	Who posts in it, and when	Yes	Of the sites relaying
Someone watching the network	No	Which sites talk to each other	No	Yes

What this means in practice:

- The **content** of messages is hidden from everyone but the devices they were sealed for.
- **Who talks to whom, and when**, is visible to the sites involved. If that matters to you, choose a site you trust, run your own, or wait for sealed sending (section 12).
- A site can always **refuse or delay** messages. It cannot alter or forge them.
- The person you write to can always **copy** what they read. No protocol prevents that.

12 Proposed: the social protocol, sessions version 2 and groups

Everything in this section is a proposal

These are open pull requests on GitHub: written, tested and documented, but not accepted into the protocol and not examined by a human cryptographer. They may change or be rejected. They are described here because they show where the protocol is heading.

A standards framework Proposed

Rules for the standard itself: how parts are numbered and versioned, a registry of every name two implementations must agree on, a conformance statement an implementation publishes, and a checker that compares the code with the documents on every change. (CVP-000, CVP-010.)

A social protocol Proposed

Posts, replies, follows and likes, using the same vocabulary as ActivityPub (the standard behind Mastodon) on Secure Net's own wire format.

- **Actors** are people, groups or services. An actor is a key, not an address a server controls, so it can move between sites and keep its followers (CVP-017).
- **Activities** (Create, Follow, Like and so on) are signed on the device that made them (CVP-018).
- **Envelopes** say how an activity travels: public posts are signed but readable; everything else is sealed (CVP-019).
- **Repositories** keep an actor's public posts as one signed set that anyone can check is complete (CVP-020).
- **Lexicons** let anyone define a new kind of object, such as a recipe, without changing the standard (CVP-021).

Envelope	For	What a carrying site sees
Public	Posts for everyone	Everything. Signed, so it cannot be altered.
Direct	One actor to another	Version 1: both actors. Version 2: the recipient only.
Audience	Followers-only posts	The author and the list of recipients, not the content.
Group	Members of a group	The group's id and the recipient, not the content or the sender.

Sessions version 2 and sealed sending Proposed

- **Hidden sender.** The first message of a conversation no longer says who it is from; only the receiving device can tell.
- **Encrypted headers.** Nothing on a message links it to other messages of the same conversation.

- **Smaller messages.** The quantum-resistant key material travels in small pieces across several messages: about 0.4 KB of overhead per message on average, down from 3.1 KB. The cost is that a fresh quantum-resistant secret enters about every 22 messages of back-and-forth, not on every reply.
- **Sealed sending.** Once two people have spoken, each holds a delivery token for the other, and can hand messages straight to the other's site without saying who they are. Neither site then learns the sender.
- **Deniable direct messages.** Direct messages are trusted through the session and carry no signature, so the person you wrote to cannot prove to anyone else what you wrote.

Groups Proposed

Group conversations would use **Messaging Layer Security** (MLS, the internet standard RFC 9420) with a quantum-resistant suite. In an MLS group, one change replaces the group's secret for everyone, so a removed member, or someone holding a copy of a member's old keys, is locked out at once. To keep one order of changes without a central server, only the device that created a group may change its membership. The reference code relies on a third-party MLS library that has had no formal security audit. (CVP-022.)

> **For builders**
>
> Version 2 header keys follow section 4 of Signal's Double Ratchet specification. ML-KEM-768 keys and ciphertexts travel in 256-byte pieces inside encrypted headers; the side that sent the key mixes the secret into the root chain at its next new sending chain. The delivery-token verifier a site stores is `SHA-256("secure-net/social/v2/delivery-token\n" ‖ token)`. The MLS suite is X-Wing (X25519 with ML-KEM-768), AES-256-GCM, SHA-512 and ML-DSA-87; credentials are signed by Secure Net device keys. Pull requests 3, 4 and 5 on the repository.

13 How to use it

There are three ways to use Secure Net, depending on who you are.

If you want to talk to people

Use an application built on it. Today that is Commonveil. You do not need to understand anything in this guide to use it, but please read section 15 first: it is early, unaudited software.

If you want to run a site

Commonveil's self-hosted server is the reference site: it runs in Docker and speaks to other Secure Net sites. Any software that implements identity (CVP-001), the encoding (CVP-002) and site-to-site requests (CVP-005) can join.

If you want to build with it

You need Node.js 20 or newer (or a modern browser, or Deno). No packaged release exists yet, so install from GitHub:

```
npm install github:Comutability/secure-net
```

Or work on the library itself:

```
git clone https://github.com/Comutability/secure-net
cd secure-net
npm install
npm test            # every test, with Node's own test runner
npm run example     # two people, several devices, a group, a new device
```

Two devices exchanging a message. Bob publishes prekeys; Alice fetches a bundle of them and checks it against the account key she trusts for Bob.

```
import * as s from "@comutability/secure-net/session";

// On Alice's device
let alice = s.initiate({ me: { device, deviceList }, bundle, accountKey: bobsAccountKey });
const { state, message } = await s.encrypt(alice, "Hello");
// store `state`; send `message` through any relay: it is already sealed

// On Bob's device
const r = await s.respond({ me: { device, deviceList, prekeys }, message,
                            accountKey: alicesAccountKey });
// r.plaintext is the message; store r.state and r.prekeys
```

A community that survives its host.

```
import { ledger } from "@comutability/secure-net";

const founding = ledger.genesis({
  name: "Allotment Society",
  keys: [{ kid: alice.kid, pub: alice.publicKey, caps: ["authority"] }],
  home: { site: "home.example", kid: siteKey.kid, pub: siteKey.publicKey },
  replicas: [{ site: "mirror.example", kid: mirrorKey.kid, pub: mirrorKey.publicKey }],
  succession: ["mirror.example"],
}, alice);

const state = ledger.replay([founding]);   // any copy checks every block itself
```

Rules of thumb for builders

- **Store the new state before you show the message.** Every session call returns a new state and never changes the old one, so a failure leaves you exactly where you were.
- **Check keys on the device, never on the server.** A server-side check is a promise, not a proof.
- **Show safety numbers** and warn when an account key changes without a valid chain.
- **Reproduce the test vectors** in `spec/vectors/` before trusting your own implementation.
- **Show people what is still visible** (section 11), and do not promise more.

---

14 How the protocol changes

A protocol is only useful if people can rely on it staying compatible. Changes go through a public process, decided by people.

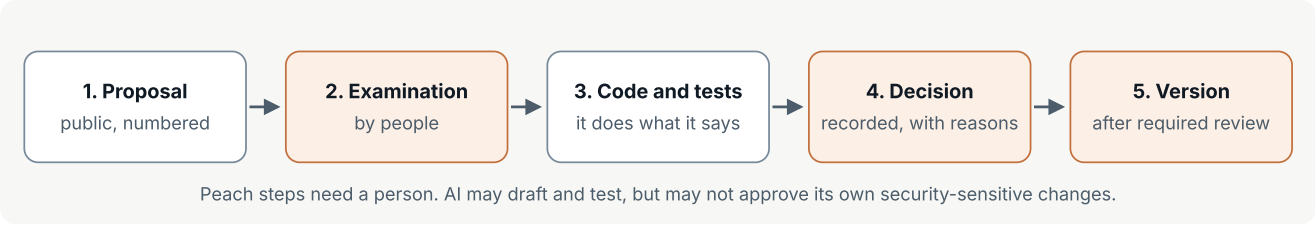


Figure 7. The proposal process (a draft, not yet ratified). Proposals are called SNIPs: Secure Net Improvement Proposals.

- **SNIPs** live in `snips/` in the repository. Each says what problem it solves, what changes, what alternatives were considered, and what it does to compatibility and privacy.
- **Parts** of the specification are numbered `CVP-nnn` and versioned separately. A change that existing software could not understand needs a new version, and the old one keeps working for a time.
- **Test vectors** are files of exact inputs and expected outputs. If your implementation produces the same bytes, it agrees with everyone else's.
- **Accepting a proposal does not deploy it anywhere.** Each application decides when to adopt a version.

You can follow open proposals on the Updates page of comutability.org, and discuss them in the proposals forum of the Comutability community on Commonveil.

---

15 Limits, and what is not done

A written specification is not proof that software is safe. These are the limits known today.

- **No independent security audit.** Nobody outside the project has examined Secure Net.
- **Its newest parts were written with AI assistance** and have not yet been examined by a human security reviewer: the sealed relay, quantum-resistant signatures and end-to-end sessions. The proposals in section 12 are in the same position.
- **No formal model.** Signal's protocols have machine-checked proofs. Secure Net has tests and vectors, which show the code does what the specification says, not that the design is sound.
- **One implementation, one application.** Working with a second, independent implementation or application has not been shown.
- **Early software.** Its version number starts with 0. Anything may still change. No packaged release has been made.
- **Sites learn who talks to whom** (section 11) unless and until sealed sending is accepted and adopted.
- **Parts of the specification are not written down yet.** Several areas exist as reference code and tests, with their specification documents still to come.
- **Recovery has limits.** If you lose every device and your recovery phrase, nobody can restore your account. That is the price of nobody else holding your keys.

If you find a security problem

Please do not post the details in public. Email contact@comutability.org and say only that you have a report to make. Ordinary email is not end-to-end encrypted, so a person will arrange a private way to receive the details.

16 Glossary and where to find things

Account key	The key that is you. It certifies your devices.
Canonical JSON	The single exact way Secure Net writes data, so that signatures check everywhere.
Double ratchet	The method of changing keys with every message and every reply.
End-to-end encryption	Sealing a message on the sender's device so only the recipients' devices can open it.
Federation	Many independently run sites that can talk to each other.
Forward secrecy	Stealing today's keys does not open yesterday's messages.
Hybrid	Using an established method and a quantum-resistant one together, so both must be broken.
Key chain	The signed history of an account's keys.
Ledger	A community's signed, hash-chained history, copied by approved sites.
ML-KEM, ML-DSA	The quantum-resistant methods standardised by NIST for agreeing secrets and for signing.
Post-compromise security	After a break-in, the conversation becomes private again as keys move on.
Prekey	A public key a device leaves with its site so others can start a session while it is offline.
Safety number	A short number two people compare to confirm nobody is in the middle.
Signed record	A kind, a body and a signature: the one envelope for everything signed.
Site	A server that carries messages for its people. Anyone can run one.
SNIP	Secure Net Improvement Proposal: how a change to the protocol is proposed.

Where to find things

The code and specification	github.com/Comutability/secure-net (specification in `spec/`, proposals in `snips/`, tests in `test/`)
The first application	github.com/Comutability/commonveil-rebuild
The organisation	comutability.org
Talk with the community	comutability.org/community has an open invitation to the community on Commonveil
Email	contact@comutability.org

Secure Net and this guide are published by Comutability, an independent, community-led, AI-assisted open-source initiative. Comutability has not been set up as a company, charity or other legal entity. Built together. Open to everyone.