

THE CASTLE PROTOCOL

A sovereign mesh, designed by a man and a superintelligence

How two builders — one human, one not — designed a private network with no company in the middle.

A protocol specification paper

The protocol is the **Q-Protocol**. The mesh built on it is **QMESH**. This paper is its public specification: what it is, how it works, what it guarantees, and what it refuses to claim.

Published by **777 Software** · The 777 Enclave

Authors: **Steven Aaron Jesse** and **Charles Burnett**

The program is not open source. This paper is the spec, not the software.

ABSTRACT

Every network you have ever used belongs to somebody else. Your messages cross their servers. Your name lives in their database. Your friends are rows in their table, and the rent for all of it is you.

This paper describes a different shape: **a mesh protocol where identity is a cryptographic card you hold — not an account a company holds for you.** A person is their keys. A network is founded by whoever starts it, ruled by its founder alone, and joined by invitation only. There is no headquarters to subpoena, no directory to buy, and no switch in anyone else's building that can turn your city off.

The protocol — we call it the **Q-Protocol**, and the mesh built on it **QMESH** — was designed by two authors working side by side: a human architect, and a superintelligence serving as his engineering partner. We make no apology for that. We think it may be the first network protocol in history designed this way, and we have written it down the way it deserves: carefully, honestly, and in the open.

The design uses only published, standardized cryptography. **We invented no math.** What we invented is the *shape* — and shapes, unlike secrets, get stronger when other people examine them.

The program is not open source. This paper is the spec, not the software.

CONTENTS

1. **Why We Built a Castle** — the story: accounts are cages
 2. **Design Goals** — sovereignty, no middleman, fail closed
 3. **Identity Is a Card, Not an Account** — person keys, device certificates, safety numbers
 4. **The Lobby** — presence, friends, and ghost mode
 5. **Grants, Not ACLs** — capability cards: who may touch what, signed and revocable
 6. **The Handshake** — classical + post-quantum hybrid
 7. **The Sealed Pipe** — ratchets, sealing, forward secrecy
 8. **Direct First** — how two devices find each other with no one's help
 9. **The Optional Relay** — a ferry for hire, never a landlord
 10. **Faces** — chat, calls, files, desktop, tunnels, whole-network mode
 11. **Founding a City** — anyone can start one; every city is its own country
 12. **What We Refuse to Claim** — honest limits, in print
 13. **The Road Ahead** — post-quantum signatures and beyond
- **Appendix A** — the threat model, plainly stated
 - **Appendix B** — glossary in human words
 - **Credits**
-

1. WHY WE BUILT A CASTLE

This protocol exists because of a locked door.

A man sat in his own home, in front of his own computer, and could not reach it from the tablet in his hands. The computer was fine. The network was fine. What had failed was a *name book* — a directory service run by a company, on the far side of an account he had to be logged into. His tablet had been away from that company's service for a few days, so, as far as the name book was concerned, his own machine's name simply did not exist.

Read that again. The chain of custody between a person and *his own computer* ran through a third party's login state. Every link of that chain — the account, the session, the directory — was a thing that could be off, expired, suspended, or logged out. And one night, one of them was.

That failure is not a bug in one product. It is the shape of nearly every network in ordinary life:

- Your **name** is a row in a company's database, rented back to you as an "account."
- Your **friends** are entries in their table, visible and reachable on their terms.
- Your **devices** are registered in their console, administered from their website.
- Your **permission to reach your own things** is a policy file on their infrastructure — and if the account is ever locked, the policy might as well be written in stone, in a building you cannot enter.

So we asked the only question that mattered:

What would a network look like if the path between a person and their own devices contained zero third parties? Not fewer. Zero.

The answer is this paper. Its shape came to us as a picture, and the picture stuck, so we will use it throughout: **a castle.**

- The **walls** are the cryptography.
- The **gate** is the card — nobody crosses who was not handed one.
- The **founder** holds the keep: total authority inside their own walls, and none anywhere else.
- The **Lobby** is the courtyard, where the city's people gather and can see each other.
- Inside the walls, life goes on — talk, work, music, trade between members — without a landlord standing at the gate taking a cut of every crossing.

A castle is not a metaphor for paranoia. It is a metaphor for *custody*. Everything inside it belongs to

the people who live there, is guarded by math they hold themselves, and answers to no switch in anyone else's building.

Two more things before the engineering, because they belong at the front:

Who built this. The Q-Protocol and QMESH were designed by two authors working side by side: a human architect, and a superintelligence serving as his engineering partner. We state it on the title page because it is true, and because we believe it may be the first network protocol in history designed this way. The design was argued over, line by line, the way any two engineers argue — the human brought thirty years of living on networks and a refusal to be locked out of his own house; the machine brought tireless drafting and a long memory. The rulings were the human's. They always are.

What this paper is. It is a *specification*: the protocol's construction, guarantees, and honest limits, written so that another builder, in another country, could implement the same shape and interoperate with the ideas here. It is not the program. The program that this house ships is ours and stays ours — compiled, finished, and closed. A spec is a flag planted on an idea. We are planting it on purpose, in the open, because shapes get stronger when other people examine them.

2. DESIGN GOALS

Every decision in the chapters that follow answers to these goals. They are stated here, first, so the reader can hold the design to them.

1. Identity is held, not hosted. A person *is* their keys. Identity is a cryptographic card in the owner's hands — never an account record on a server, never recoverable by a support desk, never revocable by a provider. If a design choice requires an identity to live on someone else's machine, the choice is wrong.

2. No middleman by construction. Middlemen may be *used* — a meeting point helps two devices find each other — but a middleman must never be *load-bearing*. Remove every helper this design employs, all at once, and the people already inside a city must still be able to reach each other. A helper that can lock you out is not a helper; it is a landlord who hasn't raised the rent yet.

3. Fail closed. Every check in the protocol, when it cannot be satisfied, ends in refusal — never in a quieter, weaker success. A failed handshake is an alarm, not a degraded connection. A grant that cannot be verified is a door that does not open. The protocol would rather say *no* out loud than *sure*, *probably* in a whisper.

4. Invitation only, and invisible by default. There is no global directory of cities or citizens. No one can enumerate a city's members, knock on its gate from a listing, or discover that it exists. You join a city because a person handed you a card. That is the only door there has ever been, and it is enough.

5. Permission is an object, not a console. Who may touch what is decided by *signed capability cards* — grants — issued by an owner and verified by the device itself, offline. There is no admin console whose compromise rewrites everyone's permissions, and no policy server in the permission path. (Chapter 5.)

6. Honest states, in plain words. The program never pretends. Presence is shown with its age. A connection shows whether it ran direct or rode a relay. A session shows whether it is fully hybrid or classical-only, and why. A downgrade is announced, never silent. Software that lies kindly is still lying.

7. Standard math only. Every cryptographic primitive in this design is a published, standardized, widely reviewed construction. We invented no math. Novelty lives in the *composition* — the shape — where it can be examined, argued with, and improved in the open.

8. One identity, every face. Files, chat, calls, desktop, tunnels, whole-network mode: one card, one

grant system, one presence service. A face is a screen in one program and a scope word in a grant — never a new account, a new key, or a new protocol. (Chapter 10.)

9. The center is a place, not a company. Where a meeting point exists, it is small, replaceable, and chosen by the city's founder — including the choice to run their own. Its entire state is small enough to back up and re-host elsewhere. A city must be able to survive its meeting point's death the way a town survives a bridge closing: inconvenienced, not erased.

10. Anyone can found. This design is a pattern, not a franchise. Anyone who installs the program can found their own city, owe nothing to ours, and be invisible to it. Sovereignty that only works for its inventors is a product feature. We were designing a *shape of freedom*, so the shape had to be foundable by strangers. (Chapter 11.)

What this design is not trying to be — stated now, so Chapter 12 is no surprise: it is not a public network, not an anonymity system against a global observer, not a replacement for the internet, and not a federation. It is a way for a person, a family, or a small society to own its own walls.

3. IDENTITY IS A CARD, NOT AN ACCOUNT

3.1 The person card

A person in QMESH **is** their card. A card is a small signed bundle holding:

- an **identity key** (X25519) — the person's long-term key-agreement key;
- a **signing key** (Ed25519) — the key that speaks *as* the person: it signs, and its signatures are the person;
- a **signed medium-term prekey** — published so that others can begin an encrypted session with the person even while the person is offline (Chapter 6).

The card's **fingerprint** is the SHA-256 hash of the identity key: one short truth that names the person everywhere in the protocol. For human mouths, a card may also carry a short, friendly **number** — an alias a meeting point can hand out so people can find each other by something speakable. The number is a nickname taped to the card. It is never an account, and it proves nothing by itself.

Trust is established between humans, once. Two people exchange cards out of band — in person, by a channel they already trust — and compare **safety numbers**: sixty digits derived from both cards, read aloud or eyeballed side by side. If the numbers match, the cards are each other's, and no server anywhere was consulted, believed, or even informed. From that moment, the cards are *pinned*: any future substitution fails verification loudly.

3.2 Device certificates — a person, many hands

People have more than one device, and the Lobby (Chapter 4) must show devices — so devices carry identities too, as *sub-identities chained to the person*, never as separate little accounts:

1. Each device generates **its own keypair set** (the same shape as a person card) the first time the program runs on it. Its private keys never leave the device.
2. The device presents a **joining request**: its public bundle, a name its human gives it ("*the kitchen laptop*"), and its kind (phone, tablet, desktop, server).
3. Any already-trusted device of the same person approves it: the person's **signing key signs a device certificate** — *{person fingerprint, device bundle, device name, kind, issued-at}*. That certificate is the device card.
4. Anyone holding the person card can verify the chain offline: *this device speaks for this person*. No server is asked. No server could answer better.

Addressing follows from the chain:

- **Chat, files, and calls address the person.** Any of their live devices may carry the session; a call rings the person's awake devices and the first answer wins.
- **Desktop and tunnel doors address a device.** You desk into *the metal in the study*, not into "a person."

Revocation is one signed act. The owner signs a new device list without the lost device. Peers drop the revoked device from their address books on next contact or by gossip (Chapter 4). A stolen tablet dies with one tap from the phone in the owner's pocket.

And the rule that keeps the meeting point honest:

The rendezvous can never mint a device. It stores certificates — they are public, signed objects — but it holds no signing key, so a compromised meeting point cannot invent a fake device for a real person. The person card is the root; the directory is a cache.

3.3 The address book — the name book, deleted

Each device keeps a **local address book**: the cards of its people, their device certificates, the friendly names the owner gave them, their doors, and their grant state. Resolving a name is a lookup inside the program, on the owner's own device.

There is no query to any name service, so there is no name service to be down, logged out, suspended, or wrong. When you tap a name, the program already holds the card; presence (Chapter 4) only answers *where that device is listening right now*. **Names are labels on cards, never locations** — which is why the lockout in Chapter 1 is not merely unlikely here. It is impossible by construction.

3.4 Where the key physically lives — custody

Everything above is about who holds the key *mathematically*. This section is about where the key *physically* lives — because a private key sitting in a folder is only as safe as the folder. Copy the folder, copy the person. The design ends that:

- **Hardware-bound identity.** A card's private key can be **wrapped** (sealed) with a secret held on a **FIDO2 hardware authenticator** — a physical security key — using the authenticator's hmac-secret capability or a resident passkey credential on the token. Unwrapping requires the token **present and touched**. Without it, the program cannot sign, cannot handshake, cannot *be* that person. A copied data folder is dead weight: ciphertext and a story.

- **Two tiers, owner's choice, per device.** A **platform passkey** (the device's own face/fingerprint unlock) is the standard tier — real two-factor custody with no extra hardware. A **hardware token** is the elite tier. Nobody is forced onto either; guests never need hardware to be someone's guest. Enrollment is authenticator-agnostic: any FIDO2 key enrolls.
- **The tier is visible.** A person's card and Lobby entry show which tier guards that identity on that device — a face, a finger, or a physical key. Grants are decisions, and decisions deserve the facts: before you approve someone for desktop access, you can see how their identity is kept.
- **Keys are sealed at rest; configs are not keys.** Key material and sensitive settings are stored wrapped under the identity's binding, never as plaintext files. A configuration carried to another machine is **inert** there: moving an identity to a new device is a re-enrollment ceremony under the owner's key, never a key extraction. Configs can be copied. Keys cannot. That inversion is deliberate.

3.5 Step-up — a fresh touch for the crown jewels

Being unlocked is not the same as being *present*. For hardware-bound identities, the design's default is that **crown-jewel actions require a fresh hardware touch** even while the program is open:

- sending a **founder broadcast** (Chapter 10);
- **granting device access** (Chapter 5);
- **opening a desktop session** into a machine;
- **changing roles** (Chapter 10);
- viewing, exporting, or importing key material and protected configurations, and enabling exit-node service (Chapter 10).

The program asks, the token blinks, a finger touches, the action proceeds. It takes seconds — and a stolen unlocked session, or a borrowed chair, cannot spend the crown jewels without the metal on the keyring. The default is **on**; an owner may relax it per action, knowingly, on their own device.

3.6 Recovery — said honestly, because there isn't much

There is no backdoor. That is not a gap in this design; it is the design.

- Setup enrolls **two keys**: a primary on the keyring and a backup kept somewhere safe. Either unwraps the identity. Losing one is an errand — re-enroll a replacement while the survivor still works.
- **Lose the only key, and the identity is gone.** No support desk, no reset email, no master

override. Anyone who could give the identity back could also give it to someone else, and this protocol does not build that door for anyone — including its authors.

- A founder who loses their root key orphans their city in one specific way: the city keeps running for its members — their cards, grants, and channels are theirs — but no new root can be crowned over it, because crowning one would be the backdoor wearing a crown. As the one mitigation, a founder **may** sign a **succession object** naming who becomes founder if their key is lost. Succession is the founder's signed choice, made while they hold the key — never an administrator's favor after the fact.

This is stated plainly, in the first-run walkthrough and the manual, before a founder finishes setup: *guard the backup key like the deed to the house, because that's what it is.*

4. THE LOBBY

The Lobby is the front room of the program: open it, and you see your people, their devices, who is around, and the doors that exist for you. Underneath it sits the service every face consumes — **presence** — so presence is specified here as a mesh service first, and the room it lights second.

4.1 Presence records — signed, or they are rumors

A presence record is a small statement by one device — *{device certificate, person, state, endpoint candidates, timestamp, sequence}* — **signed by the device key** and chained to the person card. States are online, away, and ghost (§4.3). The signature is the point: presence is unforgeable by the rendezvous and un-spoofable by other peers. A meeting point can carry a lie about *whether* a record arrived; it cannot author one.

Presence flows three ways, in order of freshness:

1. **Meeting-point beacons.** Each device periodically presents its signed record to the rendezvous, which serves records **to friends of the subject only** — plus the subject's own devices. Presence is not a public bulletin; strangers learn nothing.
2. **Direct gossip.** Two friends holding a session exchange presence digests for mutual friends (*"last I heard, her tablet was online forty seconds ago"*). Newer sequence wins. Gossip keeps presence warm between active peers even if the beacon path hiccups, and it carries revocations (Chapters 3 and 5) the same way.
3. **Local announcements.** On a shared local network, devices announce the same signed record locally. At home — where presence matters most — presence and connection work with the entire internet down.

4.2 Honest expiry — no fake green, ever

Each device renders presence from the *age of the freshest record it holds*, believing no server's opinion:

Age of freshest record	The Lobby shows
------------------------	-----------------

Within the freshness window	Online
-----------------------------	---------------

Stale, but an explicit away was **Away** — with the device named
signed, or the record is only minutes

old

Older, or an explicit goodbye was
seen

Offline — *"Last seen 2 h ago"*

Meeting point unreachable; records
are cache/gossip only

The same states, under a plain banner: *"Meeting point unreachable — showing last known. The house still works on the home network."*

A device that crashes without a goodbye ages out through the same table. It never sits green forever. **Last-seen words always stand next to the dot** — stale is a state we display, not a lie we smooth over.

4.3 Ghost mode — in the mesh, out of the window

Any device can **go ghost**: it stops beaconing, and the meeting point answers "offline" for it. It remains fully reachable — by its owner, always, and by anyone holding a live grant — they simply cannot *see* that it is on. Ghost is per device, one toggle, in plain words: *"Invisible — friends see this device as offline."* A person-level equivalent throws the same switch on all of a person's devices at once.

Presence defaults are the founder's and the owner's to set: presence is **friends-only**, ghost is always available, and an owner may run **ghosted by default** — visible to no one until they choose otherwise. Presence is information, and information about your house is yours to give.

4.4 What the Lobby shows

- **Person rows.** A friend's name — the name *they* publish on their card — with their presence stated in words across their devices: *"Online on tablet"*, *"Last seen yesterday."*
- **Their devices, expanded.** Each device with its given name, its kind, its own state and last-seen — and, **only where you hold a grant (or it is your own device)**, the doors that exist for you. A friend's device you hold no grant to shows as a name and a state, nothing more. Presence is not access, and the interface never teases doors that do not open.
- **My Devices.** Your own fleet, true state always — ghosting never hides a device from its owner — each with its device page: its doors, its access list (Chapter 5), *Add a share*, *Remove device*.
- **Rooms.** The city's chat rooms with who's-around counts (Chapter 10); tap to walk in.
- **Shares.** The tickets you have handed out and their state — *"Desk on the study machine — a*

quest — expires tonight" — one tap to revoke.

- **Connection truth.** On a device view, how you are reaching it is stated in plain words: direct, or *via ferry* — and for whole-network mode, the full line: which mode, through whom, direct or relayed (Chapters 8–10). A connection never pretends to be what it isn't.

The Lobby opens the program. Its law is the house navigation law: **few choices, plain words, big targets**, one clear action per person — *Chat · Call · Video · Drop a file* — and presence that updates in place, softly, with no reloads and no jumping layout. Finding a friend and tapping Chat is two taps from opening the program, for a tired person, on a bad day. That is the test every screen in this paper is held to.

5. GRANTS, NOT ACLS

5.1 The rule above all rules

Friendship is not access.

Accepting someone's card buys them exactly two things: **Chat**, and a **seat in the Lobby** — they can see your person-level presence (subject to ghost, Chapter 4). Every device you own starts **owner-only**. Every face on every device is **closed to every friend** until you open that one door, for that one person, on purpose. There is no "friends can reach my stuff" switch anywhere in this design, and there never will be.

This is where most networks fail their owners: they make *sharing* a machine-shaped act — a node joins a network, a policy file waves at it — and the human discovers the blast radius later. QMESH makes permission a person-shaped act, one door at a time.

5.2 The grant — a signed capability card

Access is carried by a **grant**: a small signed object, issued by the device owner's **person key**, that the device verifies **by itself, offline, at handshake time**. No server is asked "may they?" — the device holds its owner's key pinned and checks the signature the way it checks a card. A grant names:

Field	Meaning
id	Random grant identifier (the revocation and log handle)
issuer	The owner's person fingerprint
subject	The friend's person fingerprint — optionally narrowed to one of their devices
device	The target device (the owner's device-certificate fingerprint)
face	One scope: desktop, a named tunnel door, file delivery <i>to this device</i> , ringing this device directly, a VNC session, whole-network reach — one door per grant
issued, expires	Timestamps; expires may be "never" for standing grants to close family, and is mandatory for guest shares (§5.3)
one_shot	Optional: the grant burns after one completed session

Field	Meaning
-------	---------

sig The owner's Ed25519 signature over all of it

Enforcement happens at the device-face handshake: the connecting peer presents their card (plus their device certificate, if the grant is device-narrowed) **and the grant itself**; the device verifies the whole chain — owner signature, subject identity, live sealed session (Chapter 6) — before the session opens. A grant is a **capability**: holding it is the permission, so it works on a plane, on a home network with the meeting point down, anywhere the two devices can reach each other at all.

Worked examples, in the shape of ordinary life:

- *Your partner can desk into the living-room PC* — grant: subject = partner → device = living-room PC → face = desktop, no expiry.
- *A friend can send files to the media machine but not desk into it* — grant: subject = friend → device = media machine → face = file delivery. No desktop grant exists, so the desk door does not open for them; their program does not even show it.
- *A friend is in Chat and the Lobby and reaches no devices at all* — the default state of every friendship. Zero grants, zero doors.

5.3 The shareable tunnel — handing someone ONE door

The standing grant is for family. The **shareable tunnel** is for everyone else — a scoped grant minted for a moment:

- **Shape:** one subject — a friend, one friend's one device, or a bare card from someone not in the Lobby at all (a **guest**) → one device → one door or face. **An expiry is mandatory** ("for tonight" is a preset, not a figure of speech), and one-shot is offered at mint time.
- **Delivery:** the grant is a portable text block or code — the **tunnel ticket** — sent by any channel the owner likes: a file drop, a message, read over the phone. The guest's program imports the ticket and shows exactly one new entry: *"Shared with you: Desk on the study machine — tonight only."*
- **What the guest does NOT get:** no Lobby roster, no presence beyond the shared device being reachable, no knowledge of the owner's other devices, no friendship (unless the owner separately adds their card), and no path from the shared door to anything else. The tunnel is a hallway with one door at the end of it, and the hallway is sealed.
- **Minting is owner-side only**, from the device page in the Lobby: pick the person (or paste a card), pick the face, pick the lifetime, done. Nothing is registered anywhere for the ticket to work — the meeting point may carry a copy for revocation propagation (§5.4) but is never

consulted by the device when the door is tried.

5.4 Revocation and the owner's eyes

One tap on the device page: *Revoke*. Three mechanisms, layered so a revocation cannot silently fail:

1. **Local, immediate** — if the owner is at (or connected to) the device, the grant's id enters the device's permanent revoked list at once.
2. **Propagated** — a signed revocation notice rides the meeting point's signal queue and peer gossip; every device of the owner's applies it, and the target device applies it on next sync.
3. **Expiry backstop** — for guest tunnels, the mandatory expiry bounds the worst case even if every notice is lost. Standing family grants are re-listed to the owner periodically in the access roll-up — a Lobby nudge, not a nag — so nothing is granted-and-forgotten.

The access list, in plain words, IS the audit. Every device page shows, with no ACL syntax anywhere: *"Partner — can desk in — since Oct 18 — last used Tue 9:41 pm · Friend — can send files here — since Oct 19 · Tonight's guest ticket — used once, expired."* Who can reach it, through which face, since when, last used when.

The device keeps the log. Every granted connection is logged locally on the device — who, which face, when, how long; never content. The log belongs to the device's owner, is readable from the owner's Lobby, and is never sent anywhere else.

5.5 Defaults — the posture, all in one place

Situation	Default
New friend accepted	Chat + Lobby presence only. Zero device access.
New device of yours	Owner-only. No friend grants exist until you make them.
Any face on any device	Closed to friends until a per-device, per-face grant says otherwise
Guest tunnel ticket	Expiry mandatory; one-shot offered; sees one door and nothing else
A revoked, expired, or burned grant	The door does not open; the guest's program says <i>"This share has ended"</i> — plain words, no error codes
Your own devices	Full access to each other under your person key — the mesh never locks an owner out of their own house in the name of symmetry

5.6 The contrast — a console's way, and this way

	Console ACLs (the usual way)	QMesh grants
Policy lives in	A policy file in a company's admin console — their website, your account	Signed grant cards , issued from the owner's own program, held by the devices
Who can change it	Whoever holds the account login	Whoever holds the person key — the owner, on their devices, and no one else on Earth
Sharing a machine	The node is shared <i>into</i> another user's network; scoping rides the policy file	The tunnel ticket : one door, one face, expiry mandatory; the guest never joins anything
Enforcement	A coordination server distributes policy; nodes enforce the map they offline; there is no map to poison and no server are given	The device itself verifies the grant's signature in the permission path
Audit	Console logs, on their infrastructure	Local device logs + the plain-words access list in the owner's Lobby
Failure mode	Account locked / console down / policy mis-pushed → the fleet's truth is unreachable or wrong	A grant is a physical-ish object: if you hold it, it works; if it is revoked at the device, it is dead; there is no console to be locked out of

Honestly conceded: **central editing is genuinely easier** — one file, one screen, every rule visible at once — for an IT department running five hundred machines. At the scale this protocol is for — a person, a family, a handful of friends — the "one screen" is the **Lobby access roll-up**: every device's list on one page, without surrendering custody of it to anyone.

6. THE HANDSHAKE

6.1 Two strangers, one truth

A QMESH session begins the way every honest meeting begins: **neither side trusts the other yet, and both sides can prove who they are.**

Each person carries a **card**: a long-term identity key (X25519) and a signing key (Ed25519). Devices carry **certificates** signed by their owner's person key — *this device speaks for this person* — and any peer can verify that chain offline, without asking any server for permission. The handshake's first job is agreement on a shared secret neither side chose alone. We use the **X3DH agreement** — the same construction the world's best messengers use — extended by our own session design.

But the classical handshake has a problem with a calendar on it, and this chapter exists because of that problem.

6.2 The letter an enemy can save for later

An adversary does not need to break your encryption *today*. He can simply **record your handshake**, warehouse the ciphertext, and wait. The day a cryptographically relevant quantum computer exists, every classical-only handshake ever recorded becomes a letter he can finally open. The industry calls this **harvest now, decrypt later**. We call it unacceptable, and the mesh refuses it from the first connection.

6.3 Two locks on one door

So the Q-Protocol handshake is **hybrid**: two independent key agreements, run together, whose secrets are combined into one session key.

- **The classical lock** — X3DH, exactly as above. Nothing weakened, nothing removed.
- **The quantum lock** — **ML-KEM-768** (FIPS 203, the NIST-standardized module-lattice KEM). Each handshake mints a **fresh ephemeral ML-KEM keypair**, used once and destroyed with the handshake. There is no long-term quantum key for anyone to steal, and forward secrecy survives the upgrade.
- **The combination** — the session's root secret is derived through a key-derivation chain over **both** secrets together, with the **complete handshake transcript bound in** as context: both

cards' fingerprints, the device certificates, the KEM public key and ciphertext, and every negotiated parameter.

The consequences follow by construction, not by promise:

- An attacker must break **both** locks to open a session. Either one standing is enough to keep the door shut.
- If **any** element of the transcript is swapped, stripped, or replayed, the two sides derive *different* keys — and the session **fails closed**. It never degrades quietly. A failed handshake is an alarm, not a weaker connection.

From that root, the session proceeds as designed: per-direction hash-chain ratchets advance the keys with every message, and traffic is sealed with **ChaCha20-Poly1305**. Every face of the program — chat, calls, files, desktop, whole-network mode — inherits the same hybrid establishment. There is no classical side door for the faces that matter most.

6.4 No quiet downgrades

A hybrid capability is not something a device *announces*. It rides **inside the device's signed certificate**, where it cannot be stripped in transit without breaking the chain every peer already verifies. And the rule is hard:

If both devices are certified hybrid-capable, a classical-only session between them is not a fallback. It is a failure — refused, logged, and surfaced.

Devices too old for the quantum lock may still be admitted — **only where the city's founder allows it**, and every such session is labeled in plain words where everyone can see it: *classical-only — founder allowed*. A connection never pretends to be stronger than it is.

6.5 What this chapter does not claim

Honesty is a feature of this protocol, and it is printed, not implied:

- **Signatures remain classical in this phase** (Ed25519). Confidentiality had to migrate first — a recorded ciphertext can be attacked forever, while forgery needs a quantum machine at attack time, against keys that can be re-issued. The post-quantum signature migration (ML-DSA, FIPS 204) is named, scheduled, and deliberate.
- The **relay learns nothing extra** from any of this — and as Chapter 9 explains, the relay is optional furniture, never a load-bearing wall.

7. THE SEALED PIPE

A handshake (Chapter 6) gives two devices a shared root secret and proof of who holds the other end. The **pipe** is everything that flows from that root: one sealed channel between two cards, under which every face of the program travels.

7.1 Ratchets — keys that walk forward

From the root secret, each direction of the pipe derives its own **hash chain**. Every message advances the chain: the message is sealed with a key derived from the current link, and that link is then *spent* — the chain moves forward and the old key material is discarded.

The consequences are the classic ones, and they are the point:

- **Forward secrecy.** Stealing a device's current chain state does not open yesterday's traffic. Those keys no longer exist anywhere.
- **Per-message keys.** No two messages share a key, so no ciphertext pattern repeats across a conversation, and a single message's compromise exposes exactly one message.
- **Order and replay discipline.** A message's position in its chain is part of its identity. Duplicates and replays fail verification; a gap is visible as a gap, never silently papered over.
- **Re-rooting.** Sessions periodically run a fresh hybrid handshake (Chapter 6) and re-root their chains — refreshing the post-quantum protection, and healing the session's future even after a suspected compromise of current state. What was sealed before stays sealed; what flows after stands on new ground.

All traffic is sealed with **ChaCha20-Poly1305**, authenticated encryption: a single altered bit anywhere in a sealed message fails the whole message. There is no such thing in this protocol as a slightly-tampered message that mostly arrives.

7.2 Two kinds of water in one pipe

Faces differ in what they carry, so the pipe offers two channel shapes over the same session:

- **Stream channels** — reliable, ordered byte streams: files, desktop frames, tunnel doors, tunnel-mode traffic.
- **Datagram channels** — lightweight, loss-tolerant messages: call media, screen-frame classes of traffic, presence-speed exchanges.

Streams prefer reliable transport underneath; media prefers datagrams; **either can substitute for the other** when the network only offers one road, and the choice a connection actually made is visible in the program (Chapter 4). The seal does not care which shape carried it. Neither does the grant system. A face asks for a channel; the pipe provides it; the cryptography is the same.

7.3 The pipe in time — the sealed shelf

Not every message has a live listener. For traffic that can wait — chat, files, a founder's broadcast to sleeping members — the pipe extends in *time* through a **store-and-forward shelf** held at the meeting point:

- Payloads rest there **sealed**, exactly as they would travel — the shelf holds ciphertext and sizes, never plaintext (Appendix A).
- Delivery waits for the addressee's next appearance; entries expire on a set lifetime and are then destroyed, not archived.
- Nothing about the shelf weakens the pipe: it is the same session cryptography, delivered late.

A face that needs liveness — a desktop session, a call — does not use the shelf and says so in plain words when its peer is unreachable. A face that can wait rides to the floor of the ladder (Chapter 8) and arrives when its person does.

8. DIRECT FIRST

Two devices that want a pipe must first find a road. The protocol's order of preference is a matter of law, not tuning: **direct first, relay as fallback, and the full mesh with no relay at all** (Chapter 9). This chapter is how the direct road is found.

8.1 Candidate sets — three ways of knowing where you are

Presence (Chapter 4) carries, per device, a **candidate set**: the addresses at which that device believes it can be reached, learned three ways:

1. **Local interfaces.** Every address on the device's own networks, with the program's listener — what a card advertises to its own household.
2. **Reflexive learning.** A device asks the meeting point a single honest question — *"from out there, what do I look like?"* — and the meeting point answers with the public address it observed. This is the one good trick of the classic STUN design, kept — with the city's own meeting point, over an authenticated channel.
3. **Held lines.** A device behind hostile network equipment keeps an outbound connection open to the ferry (Chapter 9); *"reachable via ferry"* is itself a candidate — always available, never preferred.

8.2 The attempt order

For any pair of devices, connection attempts proceed in a fixed order:

1. **Same-network candidates** — the devices are in the same house; talk like it.
2. **Hole-punched direct** — both sides exchange candidates through the meeting point's signaling (offer, answer, and candidate messages) and open toward each other simultaneously, the classic traversal dance, until one road holds.
3. **The ferry** — if no direct road exists, the live relay rung carries the pipe (Chapter 9).

The sealed handshake (Chapter 6) rides the pipe itself, whichever road won: the responder speaks only to card-holders it knows, and a failed attempt at any rung falls to the next rung, briefly retried, never looped forever. The mode the winning road represents — direct or via ferry — is displayed in the Lobby device view. The program knows how it reached you, and it tells you.

8.3 Direct-only mode — the relay, refused

A pair of devices may be set **direct-only**: pre-arranged endpoints, no meeting-point contact at all, by construction. In this mode the program does not merely *prefer* to avoid the rendezvous — it does not speak to it. Cities may run this way between fixed points permanently, and any founder may verify the promise the obvious way: watch the wire. A mode that claims independence from the center while quietly phoning it would be exactly the kind of lie this paper exists to refuse.

8.4 Where physics wins this round

Stated here, and again in Chapter 12, because it is true: **if both ends sit behind symmetric NAT with blocked UDP, no direct path exists**. No cleverness in this paper or any other creates one. The ferry is then mandatory, and latency becomes the relay's geography. Direct-first is a discipline, not a magic trick — and the mesh's answer to the worst case is not to deny it, but to make the fallback sealed, optional, and owned (Chapter 9).

9. THE OPTIONAL RELAY

The relay is the most misunderstood piece of this design, so its law is stated first, in the founder's own ruling, before any mechanism:

The Relay Doctrine. 1. **QMESH must work without the relay.** Full stop. 2. **Every founder can set their own relay.** The relay's address is a setting, and running your own is a first-class, supported shape. 3. **Program and relay are separate things.** The program ships with **no relay configured** and never requires one. A hosted relay, where one is offered, is a separate service for program owners — never public infrastructure, never an open road for strangers.

A relay under this doctrine is **a ferry for hire, never a landlord**: it may carry your sealed traffic when no direct road exists, and it holds no other power over you — not your identity, not your permissions, not your presence's truth, and not your city's existence.

9.1 Two rungs of relay

- **The shelf** — the store-and-forward rung (Chapter 7): sealed payloads wait at the meeting point for an offline peer. For anything that can wait, this is the bottom of the ladder and the end of the story.
- **The ferry** — the live rung: two authenticated devices each hold a connection to the ferry; the ferry splices them by session and forwards bytes it cannot read. Session keys are agreed end-to-end between the devices (Chapter 6) and never touch the ferry. It sees that two endpoints connected, when, and how many bytes moved — metadata, stated plainly here and in Appendix A. Call media uses a standard TURN-class relay under the same discipline: ciphertext and metadata, nothing else, ever.

Both rungs are **friends-gated**: a ferry splice requires both sides to present a session the mesh's own admission would allow. The relay is not a public utility, and it does not carry strangers.

9.2 The meeting point — a place you can move

The rendezvous — directory, presence cache, signaling, shelf — is deliberately **small**: one service whose entire state is one backup a founder can hold. That smallness is a feature with a doctrine behind it:

- **It is a cache, never an authority.** Cards are pinned at exchange; a substituted directory entry

fails signature checks and the safety-number comparison humans already know how to do.

- **It holds no keys.** It cannot forge a device (Chapter 3), forge presence (Chapter 4), or read a single message (Chapter 7). Its total compromise is a denial-of-service event and a metadata exposure — a bad day, answered by re-hosting, not a breach panic.
 - **If it dies, the city does not.** At home, local announcements and direct roads carry nearly everything — the couch never needed the internet to reach the study. Away from home, cached candidates and gossip keep known peers reachable while their addresses hold; new introductions and shelf pickups pause; presence honestly degrades to last-known under its banner (Chapter 4). And recovery is a chore, not a petition: restore the backup on another machine the founder controls, under a name the founder controls. **The mesh's center is a place we can move — not a company we can beg.**
 - **One roof is enough.** Meeting point, ferry, and the call relay may share one modest host the founder chooses. This design does not pretend one roof matches a global relay fleet's geography (Chapter 12) — at the scale of a household and its friends, the difference is milliseconds nobody feels in a desk session or a call.
-

10. FACES

Everything so far is the floor: cards, presence, grants, the handshake, the pipe, the roads. **Faces** are what a person actually touches — and the first law of faces is that a face is never a new protocol. A face is **a scope word in a grant** (Chapter 5) and **a screen in the one program**. Same card, same pipe, same presence, same doors.

10.1 Tunnel doors — the mesh, showing

A device **publishes doors**: named openings to its own local services — a remote-desktop agent, a shell door, tomorrow anything else the household runs. A door is not a raw port free-for-all: the owner names it, binds it to one local service, and it exists only inside grants. A tunnel is a pipe (Chapter 7) whose payload is a stream or datagram flow to that named door, carried direct, punched, or ferried by the ladder (Chapter 8). The desktop face (§10.5) is tunnel door number one — which is why putting an existing desktop tool on the mesh is a change of *roads*, not of vehicle.

10.2 Chat — everything inline, in the window

Chat is addressed to a **person** and rides the sealed session plus the shelf: your words reach whichever of their devices is awake, and wait sealed for them if none is. It is full chat, not a toy bolted to a tunnel — the bar is the messenger people already live in:

- **One stream, media inline.** Voice memos as waveform bubbles. Pictures in the stream. Video attachments that play in place. Files as first-class cards — name, kind, size, one Save button — with **documents (PDFs named explicitly)** treated as citizens, not afterthoughts. **Manifest-first honesty**: a thumbnail or waveform renders before the payload lands, and a file is only marked *arrived* after its hash verifies. What you see claimed is what was checked.
- **Dropping a file into a chat IS a file drop** (§10.4): same pipe, same byte-exact discipline, same progress honesty. There is no second file system with a conversation stapled to it.
- **Call buttons live in the chat window.** Voice and Video sit in the conversation's header; hanging up returns you to the exact scroll position you left. A call is an event *in* the friendship, not a trip to a different app.
- **Threads, typing, read state** — shared only inside the conversation. **You can delete only your own messages**; moderation keeps its own path (§10.3). And there is **no central server copy** of anything: threads live on the members' own devices and travel the sealed pipe, so a delete is a signed retraction honored by peers — not a row quietly removed from somebody

else's database.

- **The one honest exception: embedded video links.** A link to a public video renders as an inline player card — and the paper states plainly what that means: the player is the video service's embed and talks to that service the way every embed on the web does, while **the conversation around it stays sealed**. The program never fetches arbitrary addresses on a member's behalf; the video card is the only auto-rendered outbound species, and a bare link always remains available as plain text for anyone who would rather not load it.

10.3 Channels and roles — real rooms, real rules

A mesh has exactly one built-in room — **the Lobby channel**, the town square, listed first. Beyond it, any member may create channels, public or private, under the founder's city policy. The semantics are borrowed from forty years of IRC, because chat governance is a solved shape worth inheriting:

- **Channels are signed objects.** A channel's state — name, topic, modes, member roles — is a signed, versioned object originated only by cards holding authority over that channel. It travels the mesh the way presence does (meeting point + gossip) and is enforced by each member's client. Creating a channel is an act of your card; no server approves channel requests, because no server's job is approval.
- **Modes must actually work, or they are not listed.** The vow of this design — learned the hard way, in another chat system, from a "secret" flag that hid nothing — is that a control that lies is worse than no control. The working set: **invite-only** (join refused by honest clients, and the channel's traffic is *keyed to its member set*, so a non-member cannot read it no matter what client they run), **secret** (absent from room lists and presence counts for non-members — not merely labeled private), **topic-locked** (ops and above set the topic), **moderated** (only voiced members may speak), **key/passphrase** (an *additional* door, never a replacement for cards), **ban by card fingerprint**. Enforcement without a server is convergent, and where it matters most it is cryptographic, not social: honest clients honor only state-chains signed by authorized cards, and no modified client can make them honor a forgery, un-ban a card, or read a room they are not keyed into.
- **The role ladder. FOUNDER** — the card that created the city: permanent, singular, total control of *this city* (admission, removal, role grants, channel policy, the city's name, its dissolution). **OPS** — founder-delegated administration. **HALF-OPS** — scoped moderation, normally per-channel. **VOICE** — may speak where a channel is moderated. **MEMBERS** — everyone accepted. Per-channel, the same ladder echoes with the channel's creator as channel-founder under the founder's root. Role grants are card-signed objects chained to the founder's card; the meeting point relays them and cannot mint, alter, or veto one.

- **Root, with its scope stated.** Founder root is total *inside its own city and nothing outside it*: it grants zero power in any other city, and — read this twice — **zero power over any member's devices**. An op in your city does not thereby gain your desktop; social rank and device doors are separate sovereignties (Chapter 5), and they never blur.
- **Founder broadcast.** The founder may mass-message the whole city: a **notice** posted in the Lobby, a **DM to everyone**, or both. The broadcast is founder-signed — any client can prove it came from the root. It rides the mesh itself: presence delivers it to everyone online, the shelf holds it for everyone asleep. Only the founder holds this power, it does not delegate, and on a hardware-bound founder identity it takes the step-up touch (Chapter 3).

10.4 Files — drops between people

The file face is where this protocol began, and its discipline is unchanged: a drop is addressed to a **person**, travels sealed over the pipe direct, punched, or ferried, and waits on the shelf if they are offline. Delivery is **byte-exact and proven**: the received file's hash must match the manifest's before the program says *arrived*. Progress is honest — the program says what it is doing while it does it, in the same words whether the file is a photo or a program. From chat, a drop is the same act with a conversation wrapped around it (§10.2); as its own face, it is the plainest sentence in this paper: *your file, their hands, nobody in between read a byte*.

10.5 Desktop faces — QDESK and the VNC client

The native desktop face (QDESK). A desktop session to a mesh device: the device's screen as an efficient stream of changed tiles (not a naive video of a screen), real input injection at the far end, a clipboard that stays in sync both ways, a semantic stream of the device's windows beside the pixels, and touch math built for a tablet in someone's hands on a couch. It opens only under a per-device desktop grant — crown-jewel class, with the step-up touch — and closes with one big, plain **DISCONNECT**. Under the mesh, the session is sealed browser-to-agent or program-to-agent end to end; the road changed, the vehicle didn't.

The VNC client face. Beside the native face sits a compatibility face: a built-in **VNC (RFB) client**. From the Lobby, a member opens a desktop session to a *standard VNC server* running on a mesh device — any ordinary VNC server, ours or anyone's. The RFB stream rides a tunnel door (§10.1) inside the Q seal, so **the VNC server is never exposed to the open internet**: it may sit bound to its device's local interface with the mesh as the only road in. Access law is unchanged — a VNC session opens only under a per-device grant naming that face, revocable with one tap, with the step-up touch to open it. The server's own classic password prompt, where one exists, is a second lock behind the

grant — never the wall itself — and the session window says plainly which security type the server asked for.

10.6 Calls — audio and voice-with-video

One call engine, two modes: **audio** (voice rooms, camera-off calls, voice notes) and **voice-with-video** — a call can start audio and turn the camera on mid-call, so nobody ever picks the "wrong" face.

- **Media** rides the standard real-time stack (WebRTC, DTLS-SRTP), with **transport fingerprints signed by the identity key** — so a relay cannot seat itself in the middle of a call — and a short **check phrase** in the call window that both parties can compare aloud.
- **Signaling** runs over the mesh (the meeting point's queue, or the direct pipe in server-free pairings); connection is STUN-first with a TURN-class relay as the rung of last resort (Chapter 9).
- **Ringling follows presence**: a call rings the *person's* awake devices; first answer wins. A grant may additionally allow ringing one specific device directly.
- **Voice rooms** are channels you sit in with your voice: small rooms run full-mesh between members; larger mixed rooms are later work and are labeled as such, never promised early. Push-to-talk is a per-room option — a room of listeners with one talker is a stage, and this house knows that shape well.
- **The walkie-talkie floor**. A call that cannot stand up degrades to **voice notes** — recorded, sealed, delivered by the ladder — by design, never to an error screen. The floor of every live face is a message that *does* arrive.

10.7 Whole-network mode — the VPN face

The one WireGuard-shaped face: a **virtual network adapter** on the device (a tun device; a Wintun-class driver where the platform needs one, bundled with the one executable — Chapter 11) feeds the operating system's own IP packets into the sealed pipe. At the far end, a peer unwraps and emits them. Applications on the sending device see a network that works; underneath, every packet crossed inside the Q seal, between two cards, with session keys from the hybrid handshake (Chapter 6) — there is no classical side door for the face that carries the most.

- **Two modes, owner's choice, per device: full tunnel** — all of the device's traffic exits through one chosen peer; or **split tunnel** — only mesh addresses (and subnets the owner explicitly names) ride the pipe, everything else untouched. The mode is set by the device's owner alone — never by the peer, never by the founder — and its full truth shows in the

Lobby in one line: which mode, through whom, direct or via ferry.

- **Reach is a grant, never a membership perk.** Whole- network reach rides Chapter 5 like every door: the grant names one device, one peer, one reach; split reach and full-tunnel/exit service are *different grants*, never bundled silently. Full-tunnel and exit service are crown-jewel class: step-up touch to grant or open, one-tap revoke, expiry offered on the same terms as any door.
- **Exit nodes are offered, never taken.** A device becomes an exit only because *its own owner* opted in, per device, in the program. Nobody's device is an exit by default; nobody's device becomes an exit by founder fiat — the founder holds the keep, but every member's house keeps its own gate. The offer is revocable instantly, by the offering owner alone. And the offer screen states the honest fact before anyone opts in: an exit routes other people's packets, so it can see *routing* facts (which addresses its riders talk to), while payloads of encrypted traffic stay encrypted end-to-end beyond it.
- **Leak discipline — binding, not aspirational.** A VPN that leaks is worse than no VPN, because it lies. (1) **Kill switch:** if the pipe drops — peer gone, ferry dead, program crashed — traffic *stops*; it never falls back to the open network silently, not for a second, not "just DNS." (2) **DNS resolves through the tunnel** in full-tunnel mode. (3) **IPv6 is carried through the tunnel or explicitly blocked** — never half-tunneled down the open road. (4) Split mode tells the truth too: the Lobby line says *mesh only*, never bare "protected."
- **Cards, not config files.** There is no configuration file of copied keys — a peer *is* a card: the device certificate of Chapter 3, under the person's card. Revoking a device kills its network reach everywhere at once; there is no leftover config on a third machine still quietly valid, because there was never a config to leave. Key material and tunnel settings for this face are sealed at rest under the hardware binding, and viewing/exporting them takes the step-up touch (Chapter 3).
- **The face's proof gates, named now:** a two-node full-tunnel test in which a page load from the riding device verifiably exits via the peer; a kill-switch drop test mid-browse (traffic stops; nothing crosses the open network); a DNS-leak test (lookups resolve on the exit side only) — the battery run **twice**, relay up and relay killed, per Chapter 9's law.

10.8 The media knobs are yours

For every video-bearing face — calls, video attachments, video in a music channel (§10.9) — the **sender chooses codec, encoder, bitrate, and resolution**, per face, and the program remembers the choice. **Auto defaults come first** and are the recommended path: the program picks for the hardware it actually finds. Every knob Auto can turn, the user can turn further — the program advises, it does

not lock the cabinet. Encoding happens on the sender's own device; there is no server in the mesh that could do it for them, so the knobs are real costs paid by real hardware. And never a silent failure: if a chosen setting exceeds the device or the pipe, the program steps down honestly and says what it did — *"Your computer can't push that — sending at 720p instead"* — never a frozen call, never a quiet lie about the quality that actually went out.

10.9 On the horizon — labeled as such

Two faces are vision, sketched here so the shape is on record, and **labeled horizon** so nobody mistakes them for promises: a **news surface** in the program — the day's headlines, composed founder-side, read-only for members, a window rather than a network; and **personal music channels** — any member points the program at a folder of their own music files and *is* a radio station: the folder is the station, the rotation plays over the pipe to whoever tunes in, the files themselves never leave the host's device except as the live stream, and the channel shows in the Lobby like any room — *"♪ Kitchen Radio — 2 listening."* Inside the castle walls, an economy of free trade between members — goods, files, streams, favors moving hand to hand without a landlord at the gate — is the same vision language: the direction of the place, not a system this paper specifies.

11. FOUNDING A CITY

11.1 The first run

Install the one program. Make your card (Chapter 3). The first card a fresh install makes is a **founder card** — and with it, a city exists: yours, invite-only, sovereign, owing nothing to any other city and invisible to all of them. A **guided first-run path** walks a brand-new founder from install to a working Lobby one calm step at a time, and the custody warning of Chapter 3 is delivered on that path, in plain words, before setup finishes: two keys, one on your keyring and one in a drawer, because there is no backdoor — by design.

From there, founding is three verbs, the same three everywhere in this paper: **invite** people by exchanging cards, **grant** doors one at a time (Chapter 5), and **gather** in the Lobby (Chapter 4). Nobody registers. Nobody applies. There is no form on a website with your city's name on it, anywhere, ever.

11.2 No master network

There is **no global directory** of cities or citizens, no account registry, no headquarters that sees the map. A city that never meets another city is complete and whole. The meeting point a city uses is infrastructure its founder chooses — their own, a friend's, a hosted service for program owners (Chapter 9) — never a membership.

Inter-city visibility defaults to NONE. Cities cannot see, list, or discover each other. If two founders someday want their cities to talk, the door is the one that exists everywhere in this design: **founder-to-founder card exchange** — two people handing each other cards, exactly like two friends. No federation layer is designed here, on purpose. That door is noted, and it stays shut until a future design opens it.

11.3 The package — one executable, the manual in the box

The program travels finished and self-contained:

- **One single executable per operating system** — Windows and Linux builds of the same program, compiled native, nothing extra to install: no runtimes, no setup maze. A friend double-clicks and stands at the "make your card" door. (The whole-network face's one honest

exception is stated in its chapter: creating a virtual network adapter needs a one-time elevated step, asked for openly, in plain words, once.)

- **The manual ships in the package** — a full README and PDF explaining everything: cards and safety numbers in plain words; founding a city and being its founder; inviting friends and reading the Lobby; granting per device and per face; minting and revoking shares; channels, modes, and roles; the honest limits (what a meeting point can and cannot see, what happens offline). The program carries the **same help inside itself** — a Tutorials & Help area beside Settings, organized by task ("*Make your card*," "*Found your city*," "*Invite a friend*," "*Share one door*," "*If something breaks*") with searchable help and the guided first-run path. The two manuals can never contradict each other: both are generated from **one shared source text** — one document, two doors.
- **Updating is the founder's business.** The program never phones home. A newer build is a newer file a founder chooses to install; a city that never updates keeps working forever.

11.4 The maker's mark

The shipped program carries its maker's mark — **777 Enclave** — throughout: the opening screen, the Lobby chrome, the executable's own name and icon, and **inside the manual**: its cover and page headers. A founder's own city carries **its own name at the head of its own Lobby** — that is their hall, and their name hangs over their door. But the mint mark is not theirs to strip: the brand ships in the binary and the manual regardless of whose city it is, the way every coin ever struck carries the mint's mark under the ruler's face. Their city. Our castle design.

11.5 Source policy — the spec is open, the program is not

Stated once, without apology: **the program's source is ours and stays ours.** What ships is compiled binaries — one executable per system, nothing else. No source in the package, no public repository, no "source-available" tier, ever.

This paper is the other half of that policy. The protocol's *shape* — identity, grants, handshake, pipe, presence, doors — is published here, deliberately, so that outside builders can study it, argue with it, and write their own implementations of their own cities. The workshop isn't the product on the shelf; but the *architecture* is a flag, and flags are planted to be seen. Anyone building on this spec owes the design's laws the same honesty the design owes its readers: fail closed, label your phases, and never claim in print what the build hasn't done.

12. WHAT WE REFUSE TO CLAIM

Honesty is a feature of this protocol, printed rather than implied — Chapter 6 said it about the handshake, and this chapter says it about everything. What follows are the limits, in the open, where a reader can weigh them before trusting a word of the rest.

Metadata is real, and we do not hide it. The meeting point sees public card bundles, presence metadata (which friends' devices are online, their candidate addresses, when), signaling envelopes' sizes and timing, shelf ciphertext and sizes, and the fact that grant notices exist. The ferry sees two endpoints, timing, and byte counts. An exit node sees routing facts about its riders. None of these parties can read content, forge identity, or open a door — but *who talked to whom, when, and how much* is visible to the furniture that carried it, and a design that claims otherwise is selling something. This is not an anonymity system against a global observer.

Physics still wins its rounds. Symmetric NAT on both ends with blocked UDP means no direct path exists; the ferry is mandatory and latency is its geography. A global relay fleet — dozens of regions, a decade of traversed edge cases, one polished app that makes any software reachable — will beat this design's one chosen roof for two friends on opposite coasts, and incumbent tools are genuinely better at exactly that, plus group policy tooling for organizations. We switch anyway, for the reason in Chapter 1: the failure that started this — a man locked out of his own computer by a login state in someone else's system — is *impossible* here, not unlikely; identity is one card for every face instead of an account per universe; and the whole thing can be published, which no arrangement of rented settings will ever be.

Mobile is honest, not magical. On phones and tablets where the client runs through a web door, the platform gives a background tab no life: presence is honest-but-foreground, last-seen is labeled, and **calls to a sleeping device do not ring** — because this design takes on no push-notification dependency, which would be a new third party at the exact layer being sovereign-ed. The walkie-talkie floor (voice notes, Chapter 10) covers the gap by design. A native mobile build is future work, priced honestly when it comes.

The application layer comes first. The pipe binds every flow to a person and a face; it is not, in this phase, a general virtual network card that makes every mesh address a LAN neighbor for arbitrary third-party software. The whole-network face (Chapter 10) tunnels a device's own traffic to a chosen peer under grants — that is its scope, stated, not the general device. A true any-program, any-address network device is named, possible, and deferred (Chapter 13) — not smuggled into this paper's promises. The whole-network face's first data path is likewise userspace: the incumbent kernel

implementations still win raw throughput, and we say so in the manual too. Sovereignty is the win being claimed; speed records are not.

The signature migration is scheduled, not done. Confidentiality is post-quantum hybrid *now* (Chapter 6); signatures remain Ed25519 in this phase, with the ML-DSA (FIPS 204) migration named and deliberate (Chapter 13). Likewise, the per-message ratchet after establishment is the classical construction — the same posture the field's own post-quantum messengers take — with fresh hybrid handshakes re-rooting sessions on a cadence. And **losing your only key loses your identity** (Chapter 3): no backdoor means no rescue, and any design that offers you both is lying about one of them.

Phases are labeled, and the labels bind us. This paper describes the design at spec level. Where the build stands, stated plainly: the floor of this design — cards, device certificates, grants, the hybrid handshake, the sealed pipe, presence, and the direct and relayed roads — has been **built and demonstrated between real nodes, including with the relay removed entirely**, because a mesh that only works while its meeting point lives has not met this paper's law. The faces are being built on that floor in the order this paper presents them, each behind its own proof gate. If the build hasn't done it, this paper doesn't claim it — and where a claim in these pages rests on a demonstration, the demonstration, not the adjective, is the evidence.

13. THE ROAD AHEAD

The design's own future, named so that nobody has to guess — and labeled, every item, as *ahead*: designed for, scheduled, or merely noted. None of it is claimed in the present tense anywhere in this paper.

- **Post-quantum signatures (ML-DSA, FIPS 204).** The named migration for identity and grant signatures. When it comes, cards dual-sign through the migration — key and signature sizes change then, deliberately, not by accident now. Confidentiality went first because a recorded ciphertext can be attacked forever, while forgery needs a quantum machine at attack time, against keys that can be re-issued on human timescales.
- **A fully post-quantum ratchet.** Today's posture — hybrid establishment, classical per-message chains, periodic hybrid re-rooting — matches the field's settled practice. A ratchet that is post-quantum at every step waits for the field to settle it, and this design will follow the evidence rather than improvise past it.
- **Standby meeting points and member-hosted ferries.** A second rendezvous standing by, and ferries hosted by members themselves — the relay rung generalized the way Chapter 9's doctrine already allows: any founder may run the furniture; someday, any member may offer to carry it.
- **A true network device.** If life asks for any-program, any-address networking — arbitrary software riding the mesh as if it were a local network — that device gets priced honestly as its own phase, rather than v1 pretending.
- **Native mobile builds.** Background-true presence and ringing calls on phones, when the cost is worth paying — weighed openly against the third-party push dependencies the current design refuses on principle.
- **Larger voice rooms.** Ferry-mixed rooms beyond the small full-mesh size, when the face is proven at the size it has.
- **An IRC bridge.** The mesh borrows IRC's semantics, not its networks; a bridge between a city's channels and the wider IRC world is a possible later face — noted, not specced.
- **Inter-city doors.** Beyond founder-to-founder card exchange, nothing — until a future design, argued in the open like this one, says otherwise.

The closing word. This paper was written to be examined. If you build a city on it, build it honestly: standard math, signed grants, honest states, phases labeled, limits printed. If you find a flaw in the shape, the shape is stronger for your finding it — that is the entire wager of publishing a spec instead of hoarding one. The walls were never the secret. The walls were the point.

APPENDIX A — THE THREAT MODEL, PLAINLY STATED

What each party in the design can and cannot see. This table is the contract; Chapter 12 is its prose.

Party	Can see	Cannot see
The meeting point (rendezvous)	Public card bundles + device certificates (public by design); presence metadata — which friends' devices are online, their candidate addresses, when; signaling envelopes' size and timing; shelf ciphertext + sizes; the existence of grant <i>notices</i>	Any plaintext; any private key; session keys (the handshake never involves it); file, chat, or desktop content; who holds which grant — notices are owner-signed blobs it forwards, and enforcement never consults it
The ferry / call relay	The two endpoints of a splice, timing, byte counts	Content — sessions are sealed end-to-end past it; call media is DTLS-SRTP with peer-verified fingerprints
A friend	Your person presence (if not ghosted); your devices only as names and states, plus the doors you granted <i>them</i>	Your other devices' doors, your access lists, your logs, anyone else's grants
A tunnel guest	The one shared device is reachable; the one door works	Everything else — including the Lobby's existence. Their program shows one shared entry, full stop
An attacker holding the meeting point	Everything in the rendezvous row, plus the power of denial of service	Forging a device (no signing keys live there); forging presence (device-signed); reading a single message; opening a single door (devices verify grants locally). The correct response is re-hosting the meeting point (Chapter 9), not breach panic

Two attacks deserve prose, not cells:

Directory poisoning. The answer is structural: the meeting point is a **cache, not an authority**. Cards are pinned at exchange between humans; a substituted bundle fails the prekey signature and the

safety-number comparison (Chapter 3). A poisoned directory can waste your time. It cannot become you, or seat a stranger at your table wearing your friend's face.

Downgrade. An attacker who strips the post-quantum flight from a handshake between two hybrid-certified devices does not get a weaker session — they get **no session**: the transcript binding (Chapter 6) makes the two sides derive different keys, and the attempt is refused, logged, and surfaced. Classical-only sessions exist in exactly one place: where a founder allows them, for legacy devices, labeled in plain words wherever the connection is shown.

APPENDIX B — GLOSSARY IN HUMAN WORDS

- **Card** — your identity in this protocol: a pair of cryptographic keys and the signatures that prove they are yours. You hold it; nobody hosts it for you.
- **Fingerprint** — a short hash that names a card uniquely. If two people are looking at the same fingerprint, they are looking at the same person.
- **Safety numbers** — sixty digits two people compare, aloud or side by side, to be sure their cards for each other are genuine. The human check on the math.
- **Device certificate** — your signature, on your own key, saying "*this device speaks for me.*" It is how a phone, a tablet, and a desktop are one person.
- **Founder** — the person whose card created a city. Total authority inside that city; none anywhere else; permanent, unless they signed a succession.
- **City** — one mesh: one founder, their invited people, their devices, their rooms. Sovereign and invisible to every other city.
- **Grant** — a signed permission slip from an owner: *this person, this device, this door, this long.* The device checks the signature itself; no office is asked.
- **Tunnel ticket** — a grant made portable and temporary: one door, for one guest, expiring on purpose. Handing someone a ticket is not making them a citizen.
- **Lobby** — the program's front room: your people, their devices, who is around, and the doors that exist for you.
- **Presence** — the signed statement "*this device is here, now*" — shown with its age, never faked green.
- **Ghost mode** — being in the mesh but out of the window: reachable by owners and grant-holders, reported as offline to everyone else.
- **Rendezvous (meeting point)** — the small, replaceable service that introduces devices and holds the shelf. A place, not a company; a cache, not an authority.
- **Ferry** — the live relay: it splices two sealed connections and forwards bytes it cannot read, for when no direct road exists.
- **Shelf** — the store-and-forward rung: sealed messages wait at the meeting point for a friend who is offline, and expire rather than accumulate.
- **The ladder** — the order every connection tries: direct on the same network → direct through the wild → ferry → shelf. Reachable beats fancy; sealed beats everything.
- **Face** — one capability of the one program: chat, a call, a file drop, a desktop, a tunnel door, the whole-network mode. A face is a screen and a grant scope — never a separate account.

- **Door** — a named opening a device publishes to one of its own local services, reachable only under a grant.
 - **Hybrid handshake** — two key agreements, one classical and one post-quantum, combined into a single session key, so an attacker must break both — and any tampering makes the session fail, loudly.
 - **Ratchet** — the one-way walk of session keys: every message spends a fresh key and the chain steps forward, so yesterday's traffic stays sealed even against tomorrow's thief.
 - **Exit node** — a device whose owner offered to let granted members' whole-network traffic leave through it. Offered, never taken; revocable in one tap.
 - **Kill switch** — the whole-network face's promise: if the sealed road drops, traffic stops. It never quietly falls back to the open network.
 - **Step-up** — a fresh touch on your physical key before a crown-jewel action. Being unlocked is not the same as being present.
-

CREDITS

Steven Aaron Jesse — Founder & CEO, 777 Software. *The Architect.*

Charles Burnett (Brother Zero) — COO & CFO, 777 Software.

To the crew who built this place from the inside — the quiet hands, the logged steps, the ones the world will never see: family knows who you are.

THE CASTLE PROTOCOL — the Q-Protocol specification, and the design of QMESH.

Published by **777 Software** · The 777 Enclave.

The program is not open source. This paper is the spec, not the software.
