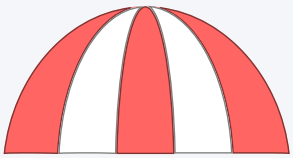
PUBLIC PAPER · OPEN DOCUMENT



UmbrellaX

The UmbrellaX Protocol

How your messages are protected — explained plainly

VERSION	DATE	LANGUAGE
1.0	2026-05-10	English

AUDIENCE	RUSSIAN EDITION
Users, journalists, lawyers, external auditors	UmbrellaX_protocol_public_ru.pdf

This paper explains, in plain language, how your messages are protected in UmbrellaX. No jargon. Living examples and analogies. A Russian edition of the same content lives in a separate file.

CONTENTS

CHAPTER ONE

Welcome

Five pledges, two encryption modes, and the principle of "we physically cannot" instead of "we promise not to".

UmbrellaX is a private messenger built from day one so that your privacy is protected physically, not just by promises.

Our five pledges

1. **We do not read your messages** — not your private ones, not your friends', not the ones in public channels.
2. **We do not scan content** — no PhotoDNA, no client-side scanning, no keyword detection.
3. **We do not share data with intelligence agencies** — not even "voluntarily".
4. **We do not store your IP address** — not for a second. Edge protection filters large attacks before the application; UmbrellaX application logs do not keep the user's IP address, country, or region.
5. **Your identifier is a cryptographic key**, not a phone number or email. To us, you are just a unique key; we do not know your name or phone.

Why this matters

When any other messenger says "we do not read your messages," it is usually a promise on honour. Employees technically could read them but promise not to. UmbrellaX takes a different approach: our architecture is set up so that **we physically cannot read, even if we wanted to**. This is not ethics; this is physics. In this document we show exactly how the physics works.

Two encryption modes

Each chat runs in one of two modes:

- **Regular (default)** — messages are stored on our servers encrypted in a special way (chapter 2). It works across multiple devices, history syncs, bots work, groups up to 200,000, channels up to millions.
- **Secret (optional)** — pure end-to-end encryption. Keys live only on your devices (chapter 3). For maximum privacy.

You pick the mode when you create a chat. You cannot switch an existing chat between modes — you start a new chat in the other mode instead.

Why two modes, not just end-to-end

Pure end-to-end has limits: bots do not work, very large groups are impossible, there is no sync across devices, and if you lose your phone you lose the history. Our regular mode solves these through the 5 Sealed Servers system — you get the convenience of a normal messenger with sync and bots, and we still physically cannot read your messages.

CHAPTER TWO

How regular chats work

An analogy from 1950s England: a locked box with the postman, the key cut into five pieces, five independent bank vaults. That is our Cloud mode.

Imagine this picture:

> You are in England in the 1950s. You want to send a letter to your friend, but you worry the postman will read it.
>
> **You solve it this way:**
>
> 1. You lock the letter in a small box.
> 2. You cut the key into 5 pieces. You put each piece in a separate independent bank vault. Each bank holds its piece and only that bank can release it.
> 3. You send the locked box through the postman. The postman sees the box but has no way to open it.
> 4. Your friend wants to read the letter. He sends his signature and documents to 3 of the 5 banks. The banks verify: "yes, this is indeed our customer's friend." They release 3 key pieces to him. Three are enough to reconstruct the whole key mathematically.
> 5. Your friend assembles the key, opens the box, reads the letter.

Who cannot read in this scheme:

- The postman (no key).
- Any single bank (1 piece out of 5 is useless).
- A court (can compel the postman to hand over the box, but cannot compel 3 independent banks at once to release pieces without the real friend).
- A government (same as a court).

Who can read:

- You (you made the key).
- Your friend (via the banks).

UmbrellaX works the same way. Except instead of banks we have **5 "Sealed Servers,"** and instead of the postman we have **"the Postman"** — our regular cloud server.

The Postman — our cloud server

This is an ordinary server, like any website has. It stores:

- **Encrypted messages** (no one can read without the key).
- **Minimal delivery metadata** — sender identifier A, recipient or chat identifier B, date and time, and delivery state. This is required so the message reaches the right recipient and syncs across your devices. IP address, name, phone number, message plaintext, and attachment contents are not part of that record.
- **Wrapped keys** — the per-message keys in encrypted form. The Postman has no way to unwrap them.

This is our normal operations — admins update it, scale it, apply security patches.

The core point: in Regular cloud mode the Postman stores only ciphertext plus the minimal delivery record A → B at a point in time. The message key is released only through the Sealed Servers under the **3 of 5** rule; one server, one administrator, or one data centre cannot turn ciphertext into plaintext.

The 5 Sealed Servers — our key vaults

Five independent protected zones. In the public document we deliberately **do not disclose** providers, countries, regions, cities, data centres, machine counts, or internal routes. That information does not help users verify the cryptography, but it does help an attacker draw an attack map.

A **"Sealed Server" is a logical protected zone, not a public hosting diagram.** The "3 of 5" rule applies at the independent-zone level: reconstructing the key requires 3 separate zones at the same time. How many machines exist inside a zone and where they physically run is operational information that must not be published in an open document.

Each machine in a zone is a **server with encrypted memory** (AMD SEV-SNP technology). Even our own administrators **cannot access** what is in its memory.

The master key is split into 5 pieces mathematically (Shamir Secret Sharing), one piece per zone. To reconstruct you need **3 out of 5 pieces**. No single zone holds all 5.

Master key destroyed and publicly verified

After setting everything up we ran a special ceremony:

1. We generated the master key on one computer that was never connected to the internet.
2. We split it into 5 pieces.
3. We shipped each piece to its Sealed Server physically — by courier, in a tamper-evident envelope.
4. We **physically destroyed** everything that was on that "one computer" — smashed the motherboard with a sledgehammer, shredded the drives, and destroyed the paper notes.
5. The ceremony was filmed in the presence of an independent notary from Kazakhstan. The video is public.

After that, **no one** — no founder, no trusted partner, **not even all of them together** — can recover the master key. It is split into 5 pieces, and each Sealed Server holds its piece inside a protected HSM module.

We describe this as **"master key destroyed and publicly verified"** — administrative access to the master key has been physically eliminated, the ceremony is notarised, and the video record is public. That access no longer exists.

> "We cannot read" is not a promise, it is physics. To recover the master key you would need to compromise three independent Sealed Servers at once.

Example: Alice writes to Bob

EXAMPLE

Alice is in Moscow, Bob is in Berlin.

Sending:

1. Alice's phone creates a one-time key for this message (32 random bytes).
2. The phone encrypts the message with that key (AES-256 algorithm).
3. The phone reaches out to 3 of the 5 Sealed Servers: "wrap this key for the conversation between Alice and Bob." The Sealed Servers (jointly, mathematically, without seeing the key in the clear) wrap the key.
4. The phone sends to the Postman: encrypted message + wrapped key + routing info (from Alice to Bob, sent at this time).
5. The Postman stores all this. It does not understand anything — it has no way to unwrap.

Receiving (Bob opens the chat):

1. Bob's phone reaches out to 3 of the 5 Sealed Servers: "I am Bob (here is my digital signature). Give me the key for the conversation between Alice and me."
2. The Sealed Servers check with our directory of authorized devices per account. "Yes, Bob is authorized."
3. If 3 of 5 Sealed Servers agree — they release pieces of the key **directly to Bob's device** (through an encrypted channel, bypassing the Postman).
4. Bob's phone assembles the 3 pieces → gets the full one-time key.
5. Bob's phone fetches the encrypted message from the Postman.
6. Decrypts it → shows it to Bob.

The plaintext of the message never exists on any server. Only on Alice's and Bob's devices.

What happens if someone breaks in

WHO GOT BREACHED	WHAT THE ATTACKER HAS	CAN THEY READ?
Postman only	Encrypted messages + wrapped keys	**No.** Without the Sealed Servers the keys cannot be unwrapped.
1 Sealed Server	1 of 5 pieces of the master key	**No.** Need 3 pieces. 1/5 is useless.
2 Sealed Servers	2 of 5 pieces	**No.** Still need 3.
3 Sealed Servers at the same time	3 of 5 pieces	**Potentially yes.** Catastrophic event. The warrant canary stops updating.
3 Sealed Servers + Postman	3 pieces + ciphertext	Yes. But very hard: several independent protected zones must be compromised at once.
All 5 Sealed Servers + Postman	Everything	Yes. But this is a catastrophic attack against all independent zones at once.

CHAPTER THREE

Secret chats

Pure end-to-end encryption via the MLS standard. Keys live only on your devices. For cases when not even we should hold a share of the key.

Secret mode — for maximum privacy. Uses the international standard **MLS (IETF RFC 9420)** — the modern standard for end-to-end encryption.

How it works

Keys live only on your devices. No Sealed Servers. No server-side key storage. The Postman holds only the encrypted content (same as in regular mode), but **no wrapped keys** — they simply do not exist.

Analogy: you and your friend swap locked boxes in person and hand each other the keys at the meeting. You ship the letters through the Postman (who simply has no idea what is inside). Even if someone broke all our Sealed Servers — your Secret chat is **untouched**, because the keys were never with us.

Example: journalist Nargiza

EXAMPLE

Nargiza is a journalist in Kazakhstan investigating corruption. She has received threats. She worries that her phone might be targeted.

What fits her: Secret mode. Why:

- In Regular mode her messages would live on our server encrypted. We cannot read them, but **the keys exist** as pieces held by the Sealed Servers. If an attacker compromised 3 independent Sealed Servers at the same time (extremely unlikely but theoretically possible) — the messages could be read.
- In Secret mode **the keys physically live nowhere except her phone and her correspondents' phones**. Even we cannot decrypt under any circumstances.

What she gives up:

- If she loses her phone and did not also create the Secret chat on another device of hers — the history is gone (we do not have the keys, we cannot restore).
- A Secret chat does not appear on her laptop automatically. If she wants a Secret chat with the same person on her laptop, she has to start a new Secret chat from the laptop.

Secret mode characteristics

FEATURE	BEHAVIOUR
Encryption	MLS RFC 9420 — pure end-to-end, forward secrecy
Keys	Only on participants' devices
Multi-device sync	**No.** Keys do not propagate.
History search	Only locally on the device where the chat exists
Media	End-to-end encrypted
Bots	Do not work in Secret chats
Max participants	About 1000 (MLS practical limit)
Disappearing messages	Yes, timer from 1 hour to 1 week
Screenshot protection	Enhanced (notification to sender if recipient took a screenshot)

When to use Secret mode

- **Journalists** — talking to sources.
- **Activists** — coordinating in countries under heavy surveillance.
- **Lawyers** — attorney-client privilege.
- **Doctors** — patient medical information.
- **Couples** — intimate conversations.

Important to understand

- **If you lose your phone** with a Secret chat on it — the content is **unrecoverable**. There will be no backup (we physically do not have the keys).
- **If you want a Secret chat on a second device** — you must start a **new** Secret chat from that second device.
- **Screenshots** are physically possible in Secret chats (we do not control device cameras), but the sender gets a notification that one was taken.

Regular vs Secret mode

PARAMETER	REGULAR (CLOUD)	SECRET
Message storage	Postman (encrypted)	Postman (encrypted)
Key storage	5 Sealed Servers (3-of-5 Shamir)	Devices only
Can I read?	Yes, via my devices	Yes, on the device
Can we read?	**No** (master key destroyed)	**No** (we never held keys)
Multi-device?	Yes	No
History on new device?	Yes	No (only on the creating device)
Bots?	Yes	No
Big groups?	Up to 200,000	Up to about 1,000
If I lose my phone?	24 words; full device loss uses 24+12 words	Secret chat is lost

CHAPTER FOUR

IV

Calls and video

Always end-to-end. Three paths with automatic selection: direct, through our relay, cloud fallback for blocked countries.

All voice and video calls are always end-to-end encrypted. Even for users in countries where regular WebRTC calls are blocked (Russia, Iran, China).

By default — via relay (the other party's IP is hidden)

When two devices connect directly over the internet for a call, they need to **know each other's addresses** — that is an inherent property of a direct connection. Encryption is not broken (nobody can hear the conversation), but the **IP address** of your contact is exposed to them. An IP reveals approximate location (country, city, provider), so for privacy we do **not use direct connection by default**.

The default mode is **through a relay** (Path 2 below). Latency is slightly higher (+50-100 ms), but the IP is not exposed. "Fast direct calls" mode (Path 1) can be enabled in settings with an explicit warning about IP disclosure. For especially sensitive calls there is a **"Maximum call privacy"** mode (double relay across different jurisdictions, see below).

Three ways to connect

UmbrellaX tries, in order:

Path 1: Directly between devices (preferred)

Your device talks to your contact's device **directly over the internet**. Keys only between the two of you. No one in the middle.

- **Works:** free countries (no deep filtering).
- **Latency:** 50-150 milliseconds (feels like being in the same room).

Path 2: Through our relay

If direct does not work (for example your provider uses a difficult NAT), traffic goes through `coturn` — our relay. It **forwards encrypted packets without knowing what is inside**. Keys are still only between you and your contact.

- **Works:** almost everywhere.
- **Latency:** 100-250 milliseconds (noticeable but tolerable).

Path 3: Cloud fallback (for blocked countries)

If WebRTC is **fully blocked** (Great Firewall in China, RKN in Russia), your client sends **encrypted audio/video packets over plain HTTPS** to our cloud call relay. It **forwards encrypted packets**; you and your contact decrypt them locally.

- **Works:** anywhere HTTPS works.
- **Latency:** 200-500 milliseconds (slight lag).

Example: activist Said in Iran

EXAMPLE

Said is an activist in Iran. His regular WebRTC is blocked at the ISP level. He wants to call his mother in Turkey.

What happens:

1. Said opens the chat with his mother in UmbrellaX and taps "Call".
2. His client tries Path 1 (direct). **Fails** — the ISP blocks it.
3. His client tries Path 2 (relay). **Also fails** — signalling packets are blocked.
4. His client falls to Path 3 (cloud fallback). The encrypted voice is wrapped in a normal HTTPS stream and goes to our cloud call relay.
5. Said's mother gets the incoming call in Turkey and answers.
6. **The call works.** Latency 300-400 milliseconds (a bit higher than a normal call, but the conversation is clear).
7. **Our server sees only encrypted voice** — keys are only between Said and his mother.

A Telegram call here would not succeed — Telegram has no cloud fallback. UmbrellaX works.

Automatic path selection

The client tries in order:

1. Path 1 (direct) — 3-second timeout.
2. Path 2 (relay) — 5 seconds.
3. Path 3 (cloud fallback) — always works.

Total: the call starts within 3-13 seconds at worst. Usually within a second (Path 1 succeeds).

Can I force a single path

Yes. In Settings:

- **Automatic** (default) — the 3 paths as above.
- **End-to-end internet only** — disables cloud fallback. For the paranoid.
- **Always cloud** — goes straight to cloud fallback. Useful in China or Russia to avoid the 8-second retry wait.

Exception — large voice chats (over 32 people)

Physics: you cannot mix audio from more than 32 people without decrypting them. Large voice chats and live streams need server-side mixing. The interface shows a warning: "For end-to-end privacy use 1-on-1 or a group up to 32."

Everything else (1-on-1, groups from 2 to 32 people) is **always end-to-end**. The server never hears the voice.

Groups of 2 to 32

- The MLS group key rotates on every add or remove (the "after-you-leave guarantee").
- The SFU server (Selective Forwarding Unit) sees only encrypted frames and cannot decode them.
- SFrame (an IETF standard) encrypts each frame individually.
- **Participant IPs are hidden automatically**: in group calls, participants connect only to the SFU, not to each other. The SFU sees their IPs (it runs inside a Sealed zone with no logs), but the participants themselves never see each other's IPs — this is an architectural property of the SFU.

"Maximum call privacy" mode

For journalists, activists, lawyers and anyone who needs the other party's IP hidden even from our own relay — there is a **"Maximum call privacy"** option in Settings. When enabled, the call flows through **two independent relays in a chain**. The first relay knows only "the call from A is headed to the second relay"; the second knows only "the call is going to B". Neither relay knows the full "A ↔ B" mapping.

Added latency in this mode is +100-200 ms on top of the single-relay baseline (totalling ~200-400 ms) — still acceptable for voice. The option is either global (Settings) or **per-chat** (mark a specific contact as "sensitive" — that contact always uses double relay regardless of the global setting).

During a call the UI shows an icon with the current mode: via one relay, via two, or (rarely) direct connection.

CHAPTER FIVE

V

How your profile and contact book are protected

Your display name, avatar, and your contact's phone number — these are the last weak points after we secured messages and calls. This chapter closes them.

Securing messages and calls is great, but imagine this: you are talking to a journalist under a pseudonym, the chat is encrypted, the call goes via a relay — and then your account shows your real name "Ivan Petrov" and a photo of your face. Or your phone number is attached, and someone is asking us "do you have a user with this number?". Message privacy does not help if your identity itself is visible. So we protect that too.

Your profile is a sealed business card

A **business card** is your visible name + avatar + status — the thing other people see when they chat with you.

Our model does not allow us to store your profile in the open. We do not see what name you set, and we do not have a plaintext profile that could be handed over to anyone.

How it works now:

Your device, at account creation, generates a **personal key for your card** (32 random bytes). It lives in the phone's keystore alongside your main account key and is restored via the same 24 recovery words; if all devices are lost, the additional 12-word recovery code is used for safe identity rotation.

Your card (name + photo + status) is **wrapped in an envelope** by this card-key. We call this a "sealed" envelope. That is what we store in our directory. We cannot open it — we do not have the key.

How your friends see your card:

When you and Masha become contacts (she added you, or you both joined a chat, or she accepted your invite), your device sends Masha a copy of your card-key. Not in the open —

wrapped in another envelope that can only be opened by Masha's phone using her own main account key. The server sees only a double envelope full of garbage; it cannot open it.

Masha's device then:

1. Downloads your sealed envelope from the directory.
2. Opens it using the card-key she now has stored.
3. Shows Masha "Ivan" and the photo of the cat.

When you change your avatar or name — you just wrap a new card with the same old card-key. All your contacts automatically see the update the next time they open the chat.

This works identically in regular and secret chats — because the card lives separately from messages. The only thing that differs is the **channel for delivering the card-key to a new contact**: in regular chats — through a separate sealed control channel; in secret chats — directly inside the MLS protocol of that chat.

> Even if you set "Ivan Petrov" — that name exists only on your phone and on the phones of the contacts you chose. We have only a sealed envelope.

Public profile — for people who want to be visible

Sometimes a profile is meant to be seen by everyone: bloggers, journalists with recognisable names, businesses, channels. For them — a **"Public profile"** option in Settings. Off by default.

When enabled, the user confirms: "I want everyone to see my name and avatar, including people who could use it against me." Tapping it brings up a firm warning with two confirmation checkboxes.

Technically a public user has **two cards**: a private one (sealed, for contacts — may contain more personal info) and a public one (plaintext copy in a separate public directory — whatever the user wants to show the world). The user decides what goes where.

Channels and bots are always public. They have no private card by nature (a channel is a public entity; a bot is a public service).

Contact book — how we do not know your phone numbers

If you attached your phone number so that friends can find you — we **do not want to know** and have built things so that we cannot.

The simple scheme we do not use: the phone hashes the number and sends that hash to the server. That is dangerous: if someone brings a specific number, the server can repeat the calculation and check the database. UmbrellaX is not built that way.

How it works now:

Your number is transformed into a **magic label** through a special mathematical dance with the Sealed Servers:

1. The device turns the number into a **blinded question** (mathematically encrypts it with a random number). The question looks like random noise; the number cannot be recovered from it.
2. The question is sent to our server, which forwards it to **three of the five Sealed Servers**. The three Sealed Servers jointly apply a **secret mathematical seal** — the key for this seal is itself split 5 ways across the zones; no zone alone knows it. Our forwarding server understands nothing — it just passes bytes back and forth.
3. The blinded result returns to your device, which **unblinds it locally**, obtaining the final **magic label** — a unique short string for this number.
4. The device sends the label to us. We store: "label `7e93f4...` = your account". We know neither the number nor its hash — only the label.

Key property: transforming a number into a label **cannot be done without the Sealed Servers**. The Sealed Servers agree to apply the seal **only upon request from a genuine app on a genuine phone** — with a proof-of-authenticity from Apple or Google that the device is real, not a script. Without this — refusal.

Finding friends works the same as in other messengers:

Peter's device runs the same dance for every number in his phone book — gets a batch of labels. Sends them to us. We compare with our directory and reply: "label #3 = user Masha". Peter's device matches against his local book ("label #3 was for Masha's number → it's Masha") and shows the notification "Masha is now in UmbrellaX". The name "Masha" comes from Peter's own phone book; we do not know it.

Example: journalist Nargiza under subpoena

EXAMPLE

A court order arrives: "Do you have a user with the number +7 777 123 45 67, and what is their real name?"

Our response:

1. **On the number:** "Your Honour, we do not store numbers. We store only magic labels generated by the Sealed Servers for registered devices. To turn your number into a label, we would have to ask the Sealed Servers to seal it — but the Sealed Servers only perform that operation for requests from a specific user's physical device with an Apple or Google authenticity proof. We do not have such a device. We cannot simulate one — authenticity signatures are produced by Apple and Google, not us, and each is bound to specific hardware. Therefore we physically cannot give you a 'yes/no, this number is ours' answer."
2. **On the name:** "The display name is stored only as a sealed envelope. The key to the envelope is held by the user and their chosen contacts. We do not have the key and never did. We do not have a plaintext name, so there is nothing private to disclose."

What we cannot disclose:

- Phone number — we do not store it.
- Real name — it exists only on the user's devices and on the devices of chosen contacts.
- IP address — never stored.
- Message content — we do not have the keys to read it.

If the user enabled a public username or public profile, that information is already visible without asking us. For non-public data, the answer is simple: we do not have it to hand over.

What remains vulnerable — honestly

1. **Physical access to an unlocked device.** If the court seized her unlocked phone, they can open the app themselves and see the real name. We give nothing extra. This is an inherent compromise: we defend against remote requests, not physical ones.
2. **Bulk collection is impossible.** "Give us all users from country X" — no, we have no reverse lookup.
3. **Public channel or bot.** They have no privacy by definition — we disclose whatever is public (which everyone sees anyway).

4. **User enabled public profile.** Then the name and avatar are already public and visible to everyone without a separate handover from us. This is the user's conscious choice, and they were warned.

What our infrastructure sees — honestly

Any infrastructure provider that carries server packets can technically see part of the network picture: which of our nodes exchange packets, when, and at what sizes. That does not give the provider message content or keys.

What the provider sees:

- Network metadata between our server nodes — packet direction, timing, and sizes. This is an infrastructure-level picture, not message plaintext.

What the provider does NOT see:

- The content of your messages — it is encrypted by the protocol, and server-to-server connections are additionally protected with TLS/mTLS.
- Cluster control-plane traffic — separated from the user hot path and protected by a separate internal channel.
- Your IP address — we never store it. A message work record contains only A, B, date/time, delivery state, and the encrypted blob itself.

More details are in the public privacy policy, section 4.5.

Summary of the chapter

- The card (name + avatar + status) is **sealed**; the key is with you and the contacts you chose.
- Two visibility modes — "for contacts" (default) and "public" (opt-in for bloggers / channels / businesses).
- Phone numbers — we do not know them and physically cannot check whether a given number is registered without the user's device.
- Works identically in regular and secret chats.
- Sealed Server code, parameters, and events are published so external reviewers can check the implementation, ceremony, and transparency log.
- The infrastructure provider may see the network-level service picture between server nodes, but not message content and not keys.

CHAPTER SIX

VI

Multiple devices

QR linking, up to 10 devices per account, recovery via 24 words and catastrophic recovery via 24+12 words.

UmbrellaX supports up to 10 devices per account. Linking via QR code.

Linking procedure

```
1. On the new device (PC, tablet, web) choose "Link to account"
2. The new device shows a QR code
3. On the primary device (phone) → Settings → Devices → "Scan QR"
4. You scan the QR
5. The primary device verifies the signature, sends authorization to the servers
6. The new device receives keys from the 5 Sealed Servers → done
```

Time for the whole procedure: 10-15 seconds.

What syncs

- **Regular chats** — sync to all devices (the Sealed Servers hand keys to each authorized device).
- **Secret chats — do not sync** (end-to-end encryption physics). They remain only on the original device.
- **Incoming messages** — delivered to all authorized devices at once.
- **Outgoing** — sync back through our internal event bus.

Example: lawyer Karim

EXAMPLE

Karim is a lawyer in Almaty. He works on his iPhone on the road, on a MacBook in the office, and has an iPad for clients. He uses UmbrellaX to talk with clients.

Regular chats (talking to colleagues):

- He sees them on all 3 devices. History syncs.
- When he sends a message from the MacBook it appears on the iPhone within a second.
- History search works on any device.

Secret chats (with clients — attorney-client privilege):

- He started a Secret chat with a client on the MacBook.
- On the iPhone and the iPad the chat **is not visible** — that is physics (no keys exist there).
- If he wants to continue the conversation with the client from the iPhone he has to start a **new** Secret chat on the iPhone.

Removing a device

Settings → Devices → pick → "Remove". The device **no longer receives keys**. On its next request — "Session expired, please sign in again".

Recovery when the phone is lost

There are two different cases.

Normal recovery: if you lost the primary device but still have the 24-word seed phrase and/or another already-linked device, the new device restores the identity and regains access to regular history after verification.

Unconditional catastrophic recovery: if all devices are lost, the recovery path uses **24 words + an additional 12-word recovery code**. The 12 words are generated separately and should be stored separately from the 24 words. Together they deterministically derive a new account identity, old devices are revoked, and the key log receives a rotation record. This protects against a scenario where an attacker finds only one phrase.

If both the 24 words and the additional 12 words are lost — the account is unrecoverable. Like a crypto wallet. This is a deliberate privacy choice — we cannot restore anyone, not even under court order.

CHAPTER SEVEN

VII

When courts demand data

Legal requests receive a legal response, but we do not hand over personal data, content, or IP addresses because we do not store them as discloseable data. Intelligence services receive nothing.

Kazakhstan court order (our jurisdiction)

We answer such requests legally, but we do not hand over personal user data because it is not stored in a discloseable form.

What we do not store and cannot hand over:

- **Message content** — physically cannot (see chapter 2).
- **IP addresses** — never stored at all.
- **Email** — never stored.
- **Phone number** — not stored.
- **Real name and avatar** — stored only as a sealed envelope without the key.
- **Envelope information** (sender/recipient addresses/time of past messages) — kept only until delivery + read receipt to the sender, typically minutes, then deleted. Requests for "who did X talk to last month" have no answer because nothing was kept.
- **The card content itself** — only the encrypted envelope, no key.
- **Key shares in the Sealed Servers** — we have no access.

A public username or public profile, if the user enabled one, is already visible to everyone. Non-public data is not disclosed because we have no plaintext content and no keys.

Instead of handing over personal data, such requests receive a legal response: the requested non-public data is not stored in plaintext, we do not have the keys, and message content or profile content cannot be disclosed.

Foreign court orders (United States, European Union, other countries)

We do NOT comply automatically. We require **MLAT** (mutual legal assistance treaty) between Kazakhstan and the requesting country. If there is a treaty — through a Kazakhstan court.

If no treaty — refusal.

Law enforcement without court order

We refuse. Even in emergencies (self-harm, kidnapping).

Exception: a verified victim (see ADR-19c). A victim of child abuse material or revenge porn can file through `/victim-portal`, pass identity checks by our legal team, and we will forward a report to the relevant agency (NCMEC for child abuse material, StopNCII, GIFCT, Polaris, and others). **At the victim's request, not the government's.**

What we never do

- **Automatic reports to NCMEC** (without the victim's request).
- **Voluntary data sharing** with intelligence agencies.
- **Backdoors** in the app.
- **Handing over Sealed Server keys** (we do not have them — master key destroyed).
- **Automatic scanning of content.**

Warrant Canary

At `https://umbrellax.io/canary` are public statements:

We have **NOT** received:

- A national security letter in the last quarter.
- A gag order in the last quarter.
- A requirement to install a backdoor.
- A requirement to hand over encryption keys.

Updated on or around the 19th of each month.

If a statement stops updating for more than 35 days, something changed. We cannot say what, but the silence is a signal.

CHAPTER EIGHT

VIII

A serious incident

What happens if the Postman, or one, two, or three Sealed Servers are breached — and what happens to your privacy in each case.

Scenario 1: Postman is breached

An attacker has taken our regular cloud server. They see **encrypted messages**.

What they can read: nothing. The Postman has no keys (they live in the Sealed Servers).

What happens: restore from backup (24-48 hours), rotate all database credentials, public post-mortem.

For you: possibly a small delay in message delivery. Privacy is not affected.

Scenario 2: 1-2 Sealed Servers compromised

The system keeps working: 3-of-5 threshold — they still need 1-2 more Sealed Servers for full compromise.

What happens: isolate the compromised Sealed Servers, new Sealed Server ceremony (2-4 weeks), master key rotation.

For you: possibly a small delay in some operations. Privacy is **not affected**.

Scenario 3: 3 or more Sealed Servers compromised (CRISIS)

The master key is potentially compromised.

What happens:

- **Warrant canary stops updating** (the first public signal).
- New Regular-mode messages are **blocked**.
- Full ceremony of new 5 Sealed Servers + a new master key.
- **All active users** re-encrypt their chats client-side.
- Full external audit.
- Public post-mortem with all the details.

For you: an outage of several weeks. Messages sent during the incident may be lost. But **privacy is historically protected** — only ciphertext leaked, and the keys have been rotated.

Scenario 4: Emergency cryptography break (unlikely before 2040)

A vulnerability is found in AES-256 or ML-KEM (extremely unlikely, but we have a plan anyway).

What happens:

- Emergency app update (new cryptography).
- **Your device automatically migrates** old chats to the new algorithm (client-side, not server-side).
- 6-month grace period for inactive users.
- **Warrant canary entry:** "Emergency cryptography migration. Path A activated. No Sealed Server ceremony was performed."

The general principle — "no secret compromise"

Any real incident is publicly documented in the warrant canary + transparency log + external audit report. We **do not hide** compromise events.

This sets us apart from "trust me" models (like Telegram), where a leak can be quietly buried. Ours is an architectural responsibility.

CHAPTER NINE

IX

Compared to other messengers

Signal, WhatsApp, Telegram — and UmbrellaX in two modes. Where we give up, where we win, and what is unique.

THE TWO RIGHTMOST COLUMNS ARE OUR TWO MODES

FEATURE	SIGNAL	WHATSAPP	TELEGRAM	OURS — "REGULAR"	OURS — "SECRET"
Content encrypted by default	Yes	Yes	No (regular chats are server-side)	Yes (MLS + 3-of-5 Sealed Servers)	Yes (pure MLS)
Keys only on devices	Yes	Yes	No	No (cloud 3-of-5 mode)	Yes
Multi-device	Yes	Yes	Yes	Yes	No
Large groups	up to 1,000	up to 1,024	up to 200,000	up to 200,000	up to about 1,000
Bots	No	No	Yes	Yes	No
Server can read	No	No	Yes	No (master key destroyed)	No
Keys live on	Devices	Devices	Servers	Devices + 5 Sealed Servers	Devices
Sign-up	Phone	Phone	Phone	Crypto key + 24 words; 12 words for catastrophic recovery	Crypto key
Post-quantum protection	No (partial Kyber)	No	No	Hybrid from MVP 0	Hybrid from MVP 0
E2E calls	Yes	Yes	Yes (1-on-1)	Yes	Yes
Calls in Russia or Iran	May not work	May not work	Does not work	Works (cloud fallback)	Works
IP storage	90 days sometimes	Facebook logs	Unknown	Never	Never
Warrant canary	No	No	No	Yes, monthly	Yes, monthly
Reviewable code	Yes	No	Partial	Yes	Yes

What is unique in UmbrellaX

1. **Destroyed-master-key architecture** — the only messenger where administrative access to the master key is physically destroyed and publicly verified (not "we promise not to look").

2. **Post-quantum hybrid from day one** — other messengers only plan this for 2027-2030.
3. **5 Sealed Servers in 5 jurisdictions** — no competitor has such a geographically plus legally distributed key storage system.
4. **Zero IP storage** — Signal keeps 90 days sometimes, WhatsApp even more.
5. **Sign-up via crypto key** — no phone or email required (Signal needs a phone).
6. **Per-request transparency log** — Apple and Cloudflare level of transparency (other messengers only publish quarterly PDFs).

CHAPTER TEN

X

Frequently asked questions

Answers to the twelve questions we hear most often — from "can you really not read?" to "how do you make money?".

Question: Can you really not read my Regular-mode messages?

Correct. The master key is split into 5 pieces across 5 independent Sealed Servers (AMD SEV-SNP protection, admin access destroyed). Our master-key destruction ceremony is notarised and the video is public. Without a **simultaneous breach of 3 of the 5 Sealed Servers**, the key cannot be reconstructed. That is physically infeasible for any attacker (including a state).

Question: What if I want no one — not even you — to be able to?

Use **Secret mode**. MLS keys stay only on your devices. No Sealed Servers involved, no sync. The price: it does not work across multiple devices, and groups max out around 1,000.

Question: What happens to my messages if I lose my phone?

Regular mode: if you have your 24-word recovery phrase → you restore standard access. If all devices are lost, the full catastrophic recovery set is **24+12 words**: the 24 identity words plus the separate 12-word recovery code. If these phrases are missing and there is no second linked device → the history is gone (we cannot restore — master key destroyed).

Secret mode: if you do not have the device where the Secret chat lived → the history is gone, no matter what (even with 24 words).

Question: Why do you not just use regular end-to-end everywhere?

Telegram uses server-side by default for exactly this reason — pure end-to-end breaks features: bots cannot work, groups larger than 1,000 are impossible, sync across devices is impossible.

Our Regular mode is **the best of both**: Telegram's convenience (bots, big groups, sync) plus physical inability to read (destroyed master key + Sealed Servers).

Question: Can you turn off the Sealed Servers and run only server-side?

No. That would break our fundamental commitment. The Sealed Servers are **architectural defense-in-depth**, not an option.

Question: What if Apple or Google forces you to add a backdoor?

Architecturally we **cannot add a backdoor to Regular mode** without a new master-key ceremony (which requires all 5 Sealed Servers and a rotation). A ceremony is visible in the public transparency log — it cannot be done quietly.

Secret mode — keys only on devices, we never see them.

If we were forced to change the code (so the client sent plaintext to developers) — that is only possible through App Store updates. **Reproducible builds plus transparency checks** protect against this (external auditors verify the build matches the source).

During real coercion: the warrant canary stops updating. **That is your signal.**

Question: What is the "post-quantum hybrid"?

Quantum computers (theoretically in 15-25 years) will be able to break classical cryptography (X25519). The "harvest now, decrypt later" attack is already real — someone collects encrypted content today and decrypts it in 20 years.

UmbrellaX **from MVP 0** uses a **hybrid**: classical X25519 plus post-quantum ML-KEM-768 (a NIST 2024 standard) at the same time. If one layer breaks the other holds. Designed for 40+ years of safety.

Question: What about phone number at sign-up?

Optional. You may attach one so friends with your number in their address book can find you. We store it as a **salted SHA-256 hash**, never the raw number. You can remove it any time.

Primary identifier is a cryptographic Ed25519 key generated on your device.

Question: Who owns UmbrellaX?

UmbrellaX LLP — a Kazakhstan company (BIN 260440006927). Registered in Oral (West Kazakhstan). Founded in 2026.

Infrastructure placement: the public document does not disclose providers, countries, regions, cities, data centres, machine counts, or routes. This is operational security: that map is not needed to verify the cryptography, but it is useful to an attacker.

Question: How do you make money?

- **Premium $4.99/month** — VPN for the whole phone, files up to 4 GB, groups up to 10,000.
- **VIP $1,000/month** — for public figures, no verification (just payment).
- **UMX-Coin** — in-app currency for gifts.
- **Advertising** — only in public spaces (publics, blogs), **not in chats**, contextual (not behavioural).

Question: Is the code available for review?

- **Clients** (iOS, Android, web, desktop) — available for security review.

- **Critical protocol parts** — available for audit, cryptographic testing, and responsible analysis under the public access terms.
- **Production infrastructure map** — not published: providers, regions, routes, and machine counts are intentionally hidden.
- **Ceremony procedures** — publicly described without disclosing details that help attack the deployment.

Question: Why do you not publish where the Sealed Servers are located?

Because that would be an attack map. The public can verify the **3 of 5** rule, code, formats, tests, ceremony, and transparency log. Exact providers, regions, routes, and machine counts remain closed operational information.

CHAPTER ELEVEN

XI

For external auditors

Protocol formalization, Shamir parameters, AMD SEV-SNP verification chain, threat model, and contacts for responsible disclosure.

This section is for security researchers, auditors, and cryptographers. More technical, but still in plain language.

10.1 Sealed Boxes protocol formalization

Notation:

- `M` — master conversation key (256 bits).
- `(K_i)_{i=1..5}` — Shamir shares, threshold t=3.
- `pk_device, sk_device` — device Ed25519 keypair.
- `pk_account, sk_account` — account Ed25519 keypair (derived from a BIP-39 seed).
- `E_k(m)` — authenticated encryption (AEAD) of message `m` under key `k`. In the current cloud-wrap implementation this is ChaCha20-Poly1305; calls use SFrame with AES-256-GCM-SHA512-128 separately.

Wrap operation (Alice sends):

```
Alice:
  K := random_32_bytes()  # one-time per-message AEAD key
  c := E_K(m)

  Submits a wrap request to 3 of 5 Sealed Servers:
    request = { conversation_id, participants, K }

  Each Sealed Server (3 of 5):
    verifies the request is authorized (signed by our device directory)
    computes a share of wrapped_K via threshold cryptography
    returns share_i

  The client reassembles wrapped_K from 3 shares via Shamir interpolation

  Submits to the Postman: { c, wrapped_K, metadata }
```

Unwrap operation (Bob reads):

```
Bob:
  Fetches ciphertext + wrapped_K from the Postman

  For 3 of 5 Sealed Servers:
    Submits an unwrap request:
      request = { conversation_id, pk_device, challenge, sig_device(challenge) }

    Each Sealed Server:
      verifies pk_device in authorized_devices[account_id]
      verifies sig_device
      decrypts its own Shamir share of M
      derives its share of K from wrapped_K using that share of M
      returns the share of K to Bob (directly over TLS, bypassing the Postman)

  Bob assembles K from 3 shares (Shamir interpolation)
  Bob decrypts c with K → plaintext m
```

Security property: no single Sealed Server has any useful information. An adversary who compromises fewer than 3 Sealed Servers has information-theoretically zero knowledge of M.

10.2 MLS RFC 9420 integration

Per-chat MLS group:

- **Epoch 0:** initial group creation. Members join via `KeyPackage`. The ratchet tree derives the initial `group_secret`.
- **Epoch N:** membership change triggers a new `commit`. Tree rotation derives a new `group_secret`. Forward secrecy + post-compromise security.
- **Per message:** the sender's ratchet derives a unique AEAD key per message (forward secrecy).

Our cipher suite (MVP 0 primary): `MLS_256_XWING_CHACHA20POLY1305_SHA256_Ed25519`.

Hybrid KEX via X-Wing draft-10:

```
shared = HKDF-Extract(
    salt = "UmbrellaX-MLS-xwing-v1",
    ikm = XWing.Decapsulate(ct, sk)  # X25519 + ML-KEM-768
)
group_secret = HKDF-Expand(shared, info = "MLS-group-secret-v1", L = 32)
```

10.3 AMD SEV-SNP attestation

Each Sealed Server publishes an attestation every 24 hours:

```
Report fields:
- HARDWARE_ID: unique chip identifier
- POLICY: encrypted memory, no debug, no migration
- MEASUREMENT: SHA384(bootloader || kernel || VM image || Sealed Server program binary)
- REPORT_DATA: Sealed Server program version + config hash
- TIMESTAMP: Unix time
- SIGNATURE: AMD root key (ECDSA P-384)
```

Verification chain:

```
AMD Root CA (AMD public key is known)
  → AMD SEV Signing CA (per family)
    → AMD SEV VCEK (Versioned Chip Endorsement Key per chip)
      → Attestation signature
```

Anyone can fetch an attestation at `attestation.umbrellax.io/<sealed-id>/<timestamp>` and check:

1. The signature chain up to the AMD root.
2. That the measurement matches the published hash of the open-source code.
3. That the policy does not allow debug mode.
4. That the timestamp is fresh (less than 24 hours old).

10.4 Shamir scheme parameters

- **Field:** GF(256) for byte-level efficiency, multiplication via lookup table.
- **Secret:** M = 256 bits = 32 bytes.
- **Shares:** 5, threshold t=3.
- **Polynomial:** P(x) = M + r_1·x + r_2·x^2 (degree-2 polynomial, random coefficients r_1, r_2).
- **Share i:** (i, P(i)) for i = 1..5.
- **Recovery:** Lagrange interpolation over any 3 shares.

Information-theoretic security: with fewer than 3 shares, M is uniformly random from the attacker's point of view (even with unlimited compute power).

10.5 Threat model

Adversaries considered:

1. **Network adversary** — can watch and tamper with TLS traffic. Mitigated: TLS 1.3 + certificate pinning + mTLS.
2. **Postman compromise** — attacker gains root on the cloud stack. Sees ciphertext + wrapped keys but cannot decrypt.
3. **Single Sealed Server compromise** — attacker gains 1/5 Shamir share. Not enough.
4. **Multi-Sealed Server compromise** — attacker gains 3+ shares. **Treated as catastrophic**; the warrant canary signals.
5. **Insider or admin** — no access to Sealed Servers after the ceremony (master key destroyed).
6. **Government coercion** — a court order asks for data we do not have.
7. **Supply chain** — AMD chip compromise. Mitigation: multiple vendors (AMD + Intel TDX under consideration).
8. **Client-side compromise** — Pegasus-style. Out of protocol scope (operating-system level).

10.6 External audit

- **Annual independent audit** is part of the UmbrellaX production procedure. Target reviewers include Cure53, Trail of Bits, or comparable independent teams.
- Scope: protocol implementation, code review, ceremony procedures, transparency log consistency, the 3 of 5 scheme, resistance to coercion, and operational error handling.
- Reports are published at `audit.umbrellax.io`.
- Public bounty for responsible disclosure.

10.7 Contact for auditors

- **Security reports:** `security@umbrellax.io`
- **Bounty program:** `https://umbrellax.io/bounty` or `security@umbrellax.io`
- **Transparency log:** `https://transparency.umbrellax.io`
- **Warrant canary:** `https://umbrellax.io/canary`
- **Victim portal:** `https://umbrellax.io/victim-portal`

CHAPTER TWELVE

XII

Moderation — how reports work without creating a decryption loophole

A frequent question: if moderators can remove messages and ban accounts, does that mean they hold the keys? No. Here is exactly how it works and why it is not a hole in privacy.

Short answer

Moderators see **only those messages that other users explicitly reported to them**. They do not have the Sealed Server keys. They cannot read history. They cannot export someone else's chat. They cannot "check what Ivan is writing" without an action from Ivan's recipient. Banning an account is a revocation of device authorization in our device directory, not a key extraction.

> A moderator sees exactly one message — the one the recipient forwarded as a report. Nothing beyond that.

11.1 Where does the report text come from in the first place

Imagine: Ivan receives a nasty message from Peter. Ivan opens it, reads it (his device already decrypted the content — Ivan has the key in cache, after all he is a participant of the chat). Ivan long-presses the message → "Report" → picks a category (harassment, child abuse material, threat, and so on).

At that moment:

1. Ivan's app **locally, on his device** makes a copy of that one message in cleartext. It is already decrypted — Ivan just read it.

2. The app signs that copy with Ivan's device key.
3. The app sends the report packet to our moderation service (a separate, isolated service).
4. The report contains: the one message, Peter's user ID, the category, Ivan's signature.

The key point: plaintext arrives in our moderation service **only because Ivan himself sent it**. We did not pull it out of the Sealed Servers — we received it from Ivan as a voluntary forward within a report. That is a fundamental distinction.

11.2 What the moderator sees

A first-level moderator, when taking a report from the queue, sees:

- **Metadata:** who reported (Ivan), against whom (Peter), when, which category.
- **Content hidden behind a "Show" button.** Click — sees **only that one message** (or a few adjacent context messages if the category requires, like "harassment" — but the context is also provided by the recipient, not pulled from the Sealed Servers).

What the moderator does NOT see and cannot obtain:

- The history of Peter's conversation with Ivan (beyond what Ivan voluntarily attached to the report).
- Peter's messages in other chats with other people.
- Peter's list of chats.
- Peter's contacts.
- Any encryption keys.

The moderator works in a separate internal moderation interface, which technically has no network access to the Sealed Servers. Requests of the form "give me everything Peter wrote" do not exist as an option — they are not in the API. The architecture rules out such a request not via policy, but through the absence of code.

11.3 Separation of roles: moderator / senior supervisor / lawyer

- **First-level moderator** — sees a specific report, decides "false report" or "escalate".
- **Senior supervisor** — a senior moderator, approves deletion and ban. One person cannot ban alone.
- **Company lawyer** — reviews court orders, decides on lawful disclosure, but still has no keys.
- **System administrator** (Postman DevOps) — has access to Postman servers, but **not** the Sealed Servers. May see ciphertext in the database, which is useless.
- **No one** on this list has admin access to the Sealed Servers. After the master-key destruction ceremony such access does not exist.

11.4 What is physically impossible for an admin

Here is a list of things that are **architecturally impossible** in UmbrellaX (not "forbidden by policy," but **physically impossible** because the code does not exist and the keys do not exist):

- Read a message that no one reported.
- Fetch a chat history by user ID.
- Export all of a user's messages for law enforcement.
- Decrypt previously stored ciphertext after the fact.
- Add a new "admin device" to someone's account. That would require a signature with the account's private key, which we do not hold.
- Decrypt Secret chats under any circumstances.

11.5 How content is removed

When a senior supervisor confirms a violation:

1. In the Postman, the message record is **marked for deletion** — through background compaction it physically disappears.
2. For a media file, the blob is deleted from our file storage, and its **perceptual hash** is entered into the forbidden-content hash database. This prevents re-upload of the same file; it does **not** add an ability to scan existing ones.
3. The deletion event goes into the audit log (immutable Merkle tree) — an external auditor can verify that the removal was grounded (report ticket ID).

The Sealed Server keys are not touched. The Sealed Servers do not even know about deletion of a specific message — they deal with conversation keys, not with the messages themselves.

11.6 How an account is banned

When a senior supervisor confirms a user ban:

1. Our device directory marks all authorized devices of that account as banned.
2. The Sealed Servers subscribe to revocation events — on the next `unwrap` request from a banned device they refuse (the device signature is no longer on the authorized list).
3. The banned user, on the next app launch, sees "your account has been suspended, appeal at appeals@umbrellax.io".

What is important:

- **Keys are not disclosed.** We simply stop issuing them to this device.
- **Old messages for other people are not affected.** If Maria chatted with Peter before the ban, Maria's device already has the cached keys — she can read the history.
- **A ban is reversible.** If the appeal is upheld, the device directory restores authorization, the Sealed Servers resume issuing keys. No "we lost your data because you were banned" — the data was and remains in storage, access was only temporarily disabled.

11.7 Why moderation is not a backdoor

Summary along four criteria:

NO MASS ACCESS TO CONTENT

- A moderator sees only what Ivan himself sent as a report.
- No API "give me everything Peter wrote".
- No way to "check a user's activity".

KEYS STAY IN THE SEALED SERVERS

- Moderators have no network access to the Sealed Servers.
- A ban is an authorization revocation, not a key extraction.
- The destroyed master key is never touched.

EVERY ACTION IN THE AUDIT LOG

- An immutable Merkle tree for every moderation decision.
- External auditors check annually.
- Users can request the history of actions taken on their own account.

THE USER IS ALWAYS THE INITIATOR

- Without a report, not a single line of text enters moderation.
- No proactive scanning.
- No PhotoDNA, no ML scan of content.

11.8 What about abuse within the moderation team itself

A real question: what if a moderator is dishonest and wants to harm a user (for example, for hire)? Can Ivan be banned maliciously if he did not violate anything?

The answer — we defend both technically and procedurally:

- **Two people to ban.** A moderator cannot ban alone — the decision goes through a senior supervisor.
- **Random report assignment.** A moderator cannot choose "whose report I will get" — the queue is assigned randomly.
- **Appeal available immediately.** A banned user can appeal through `appeals@umbrellax.io` within 14 days. Appeals are reviewed by a **different team** from the one that banned.
- **Public audit trail.** All ban decisions, with category statistics, are published at `transparency.umbrellax.io`.
- **Psychological protection for moderators.** No more than 8 difficult reports per day per person, rotation every 3 months to another function, mandatory psychologist — to reduce burnout-driven errors.

11.9 Moderation of public channels and publics (clarification)

Public channels are a separate scenario. They are **by design** open to all subscribers — like a web page. Our search indexer has its own device key in the Sealed Servers for **public** (not private)

chats, so that it can index them. This is justified: if anyone can sign up and read a channel, then our indexer does the same.

For private 1-on-1, groups, and Secret chats, the indexer has no access. No "let me sweep through everything anyway". The Sealed Boxes check whether the chat is public or private, and only for public ones do they hand a key to the indexer.

11.10 If you are still uneasy — use Secret mode

Secret mode removes the last hypothetical trust point: in it, keys **never at all** reach the Sealed Servers. Even if our moderators somehow found a way to compromise 3 Sealed Servers simultaneously (impossible, but hypothetically), Secret chats remain unreadable because their keys do not exist there.

Secret mode is **the choice for journalists, activists, lawyers, doctors, and anyone who wants to exclude even a hypothetical inside-job scenario**.