

ENIGMA 3

Your messages. Your server. Nobody else's.

The self-hosted messenger with end-to-end encryption on every layer: messages, attachments and calls. No corporation, no cloud, no telemetry — everything runs on your own hardware, under your control.

🔒 End-to-end encrypted

🏠 100 % self-hosted

⏱️ Self-destructing messages

📞 Encrypted calls

🖥️ Windows · Linux · Android · Web

WHY ENIGMA 3

— Trust nobody. Least of all a provider.

Most messengers promise: “We protect your data.” That promise depends on trust — in the operator, the company, its employees. The most famous “unhackable” services in the world failed exactly there. The facts below come from official Europol, Eurojust and prosecution announcements.

💥 EncroChat (2020) · Sky ECC (2021) · ANOM (2021)

EncroChat: Police broke into the **vendor's servers** — not the cryptography — and pushed an update to all 60,000 devices that captured the keys: **60 million messages** read, €700 million seized, **6,500+ arrests** (Operation Sirus). **Sky ECC:** servers seized, founder arrested — the provider held the keys itself. **ANOM:** a covert FBI operation (Trojan Shield), **the product itself was the backdoor**, 300+ convictions. Official Europol, Eurojust and prosecution records. **The common failure of all three:** the operator was the weak spot, not the encryption. Enigma 3 removes that spot — the server runs at your place. There is nothing left to seize.

The principle: Enigma 3 has no vendor instance to seize and no telemetry to leak hints. What matters happens on your hardware. You decide who may read what. And if you want: nobody.

The everyday comparison

	WhatsApp	Telegram	**Enigma 3**
Server belongs to	Meta	the provider	**you**
Read access	provider (optional backup)	provider (in “cloud chats”)	**nobody**
Metadata	at Meta	at the provider	**on your hardware**
Quitting	provider deletes your data	provider deletes your data	**you are the operator**

Who is this for?

Right for you if …

… you don't want to park family or team matters at a corporation.

… your club or practice wants to just leave GDPR alone instead of explaining it.

… you want to escape the WhatsApp clones without giving up familiar features.

… you want an admin between “too simple” and “enterprise maze”.

SECURITY

— Encrypted to the roof — as the default

You don't configure anything. All of the following applies from the very first message:

🔐 End-to-end encryption

Direct messages: X3DH + Double Ratchet (like Signal) with forward secrecy, self-healing and replay protection. **Channels and groups:** sender keys, rotated on every departure. **Attachments:** AES-256-GCM, filename and type inside the ciphertext. The server stores and relays **ciphertext only** — it cannot read anything.

📞 Encrypted calls — on your server

Voice and video with screen sharing, a media key per call, packed to the participants' device keys. New key at every participant change. **The LiveKit server runs at your place** — what Signal and WhatsApp do in their data centers happens here on your hardware. Can run without the LiveKit container if you prefer.

⏱️ Self-destructing messages

Lifetime per message: off · 1 h · 7 h · 24 h · 3 · 7 · 30 days (default 7). The timer lives **inside the encrypted content** — the server cannot extend it. The client deletes at the earlier of its own timer or the server's time and shows deviations.

🔥 Irreversible deletion

Crypto-shredding: every message has its own key. Deleting or expiry destroys the key — copies on the server and in backups become unreadable. Deliberately **no built-in backups** or snapshots that would defeat deletion.

📱 Devices & safety numbers

A separate key per device, never leaving it. Review and revoke devices, alerts on new devices, optional auto-removal of inactive devices. Safety numbers for comparison rule out unnoticed eavesdropping.

🔑 Local vault

Biometrics or a PIN seal the keys on the device (Argon2id, AES-GCM). Android Keystore/StrongBox, iOS Secure Enclave, Windows DPAPI. Without the PIN, a seized device yields only ciphertext.

CRYPTOGRAPHY

How the encryption works — in detail

Vetted standard primitives, no proprietary crypto: NIST P-256, HKDF/HMAC-SHA-256, AES-256-GCM. Private keys never leave the device.

🤝 Key agreement: X3DH

The sender fetches the recipient device's **prekey bundle** and verifies the signed prekey against the **pinned identity signing key** (trust on first use). The result is a shared secret known only to these two devices. Prekey retrieval is rate-limited against draining one-time prekeys.

🔁 Double Ratchet (DMs)

Every message gets its **own key**. The symmetric ratchet gives forward secrecy; the Diffie-Hellman ratchet gives **self-healing** after a compromise. Headers are encrypted. Seen base keys are rejected (replay protection). Each message is encrypted for **every recipient device** (up to 64) and your own other devices.

👥 Sender Keys (groups & channels)

Each device maintains a chain per room: `mk = HMAC(ck, 0x01)`, `ck' = HMAC(ck, 0x02)`. Every message is signed with the chain's signing key; the signature covers room and sender. Chains are distributed **through the pairwise Double-Ratchet sessions** to confirmed members only. Rotation: on any departure, device removal, device change, after 7 days, or before counter overflow. New members get the chain from the current state — **older messages stay unreadable**.

📞 Calls & attachments

Calls: LiveKit Insertable Streams. A leader generates random media keys per call and distributes them packed to each participant's device keys; new key at every change. **Attachments:** AES-256-GCM via the ratchet (DMs) or sender key (channels), bound to room and sender. EXIF, GPS and XMP metadata are stripped on upload.

Keys per device

Identity key (ECDH P-256)	Device identity, basis of the safety number — permanent per device
Signing key (ECDSA P-256)	Signs the signed prekey; pinned on first contact (TOFU)
Signed prekey & one-time prekeys	For the X3DH exchange — renewed regularly, replenished after use, retrieval rate-limited

ENFORCEMENT & TRUST

— What the server refuses — and what the client watches

🚫 The server rejects plaintext

`CreateMessage`, `UpdateMessage`, `CreateUpload` and image descriptions return `e2ee_required` without ciphertext — in DMs, groups and channels. This duty **cannot be switched off in production**, neither by setting nor environment variable. Old plaintext channel messages are destroyed at startup: key destroyed, storage securely overwritten.

🚪 No silent eavesdroppers

The client remembers the first seen member list. Members added later stay **pending** until you confirm them in the room or verify their safety number. The server can claim new members — but cannot hand out keys unnoticed. In DMs, sending pauses until you confirm a changed key.

⚠️ Device warnings (TOFU)

Device keys are pinned on first contact. Deviations mark messages "**New device**", turn the header lock red and offer confirm or verify. Never-verified contacts carry a subtle symbol. New sign-ins alert **your other devices** with name and time — "Was that me?" — with one-click revocation.

🧹 Controlled deletion

System messages (joins, calls, room changes) are deletable immediately or automatically; the server securely overwrites stored events. After deletion only a content-free deletion proof remains. Only entries that define current state survive: room, name, settings and the last join per person.

🔒 Hard defaults

- ✓ **Invitation-only** registration
- ✓ **12-character** minimum password
- ✓ Login throttling with growing lockout
- ✓ **Absolute session lifetime** (default 12 h) — not just idle
- ✓ HTTPS with HSTS, strict CSP, no embedding
- ✓ Auto-removal of devices inactive 30–365 days (optional)

🧨 Removed, not disabled

- ✓ **No bots** — API keys, MCP, webhooks gone; existing bot accounts revoked at startup
- ✓ **No SSO/OAuth** — password login always active
- ✓ **No link previews** — the server would see the URL in plaintext
- ✓ **No telemetry, no crash reporting**

DEVICE SECURITY

Keys under lock — the local vault

🔑 How the vault works

The PIN or passphrase derives a key-encryption key with **Argon2id** (64 MiB, t = 3; otherwise PBKDF2-SHA256 ≥ 600k iterations). It wraps a random 256-bit **data key (DEK)**. Every vault entry is sealed with **AES-GCM** — unique nonces, the record ID as additional authenticated data. Platform binding adds a second layer: Android Keystore/StrongBox, iOS Secure Enclave, Windows DPAPI via safeStorage. On lock, inactivity timeout, background and app exit the DEK is discarded from memory. On mobile, every return from the background requires Face ID/Touch ID/fingerprint, PIN as fallback.

⛔ Attacks and answers

Wrong PIN: exponential backoff; optional vault wipe after N attempts (default selectable: 10). **Forgotten PIN:** local wipe and re-registration as a new device — server messages stay. **Stolen database:** without the PIN only ciphertext; hardware binding makes offline brute force expensive. **Weak short PINs** stay offline-bruteforceable — a long passphrase is recommended.

📱 Device management

Settings show all devices with message keys; each is individually revocable. A revoked device is logged out and gets no new keys. Inactive devices can be auto-removed after 30, 60, 90, 180 days or a year — revoked completely along with keys, sessions and tokens.

✅ Safety numbers

Derived from both accounts' identity keys. Compare in person or via QR code — afterwards the person counts as **verified**. In groups and channels this works per member through the lock in the header, which also shows every member, their devices and the sender-key status. A key change on a verified device pauses sending until you confirm — in DMs, groups and channels alike.

THE SECURITY MODEL

Who sees what — and who cannot do what

Server operator / attacker with server access	Sees **metadata**: who with whom, when, which room, message sizes, IPs, device lists. **Cannot:** decrypt content, attachments or calls; extend timers; slip in members or devices unnoticed (if you check safety numbers).
Network observer	Sees a TLS connection. **Cannot** read content.
Stolen / seized device	With active PIN: keys and caches sealed (Argon2id/PBKDF2 KEK, AES-GCM DEK). Without PIN: ciphertext only; hardware binding makes brute force expensive. **Cannot** reconstruct older ratcheted messages of other devices.
Push provider	Sees time and target device. By default **no** plaintext titles or sender names — never content or sender names from the payload.
Investigator with server seizure	Memberships, timestamps, sizes, device lists, push tokens. **Without users' device keys: not a single readable message.**

What stays unencrypted — and why

- ✓ **Metadata** — communication partners, times, room membership, sizes. Inherent to any server architecture; state it openly.
- ✓ **Room names, profiles, reactions** — not covered by E2EE; protect server, `.env` and backups accordingly.
- ✓ **System messages** — generated by the server core (e.g. "… joined"); deletable, but not E2EE payloads.

Backup discipline: back up the VM **encrypted** and keep backups **no longer than the longest message lifetime you use** (default 7 days, max 30). Older backups would defeat deletion. Store `.env` separately — it contains the master key.

METADATA PRIVACY · STAGE 1

— Encryption hides content — stage 1 takes the metadata too

The adversary here is the server operator or anyone who obtains a copy of the server data: event log, buckets, snapshots, backups, logs. Stage 1 removes the metadata the server does not need to operate and makes the rest less precise. Experimental, default-on, no configuration.

🏷️ Encrypted channel names

The client encrypts **channel name and description** with the room's sender keys; the server keeps only ciphertext and a neutral placeholder (🔒 `#4f2a9c01d7`). The plaintext name never reaches the server — renaming a migrated channel in plaintext is rejected (fail closed). Members who can read messages read the name; non-members and devices without the key see the placeholder. Old plaintext name records are deleted at migration.

📏 Size padding

Every encrypted body and edit is padded to a size bucket before encryption (Padmé rule, ≥256 bytes, at most ~12 % overhead, invisible marker characters). **The server cannot tell a short reply from a long sentence.** Attachments already used buckets.

⌨️ Typing opt-out

Settings → Security & privacy: “Send typing indicators” off means this device sends none — while you still see everyone else's. The choice is per device and server, never stored centrally.

🪪 Encrypted profile names

A display name and avatar are stored end-to-end encrypted; the server sees only ciphertext and a stable placeholder. Anyone who shares a readable room receives the grant key automatically — including devices that join later. Login names stay the visible constant.

😊 Encrypted reactions

New reactions travel the same path as the message itself: the server stores only an id and ciphertext and cannot tell how many reactions a message carries or which emoji they are. Reactions from the plaintext era stay readable and removable.

🗒️ Minimal server logs

By default, process logs write **no user IDs, login or display names, room names, e-mails or IPs** — identity fields show as `[redacted]`. Sign-in audit records keep no hash of IP or user agent unless the operator enables it — and then only with a key that rotates on a window (default 24 h), so hashes cannot be chained across time. The session list still shows the coarse device class (“Firefox on Windows”).

CONTROL

— Your account, your server — with the hard buttons

🔑 Passkey sign-in (WebAuthn)

Sign in without a password: the server stores only the **public credential**, the private key stays on your device. Passkeys are optional — password sign-in always stays available and cannot be disabled. Each ceremony uses a five-minute nonce that verifies **at most once**. Adding or removing a passkey alerts your other sessions; removing revokes established credentials like a password change. The relying party is derived from the public HTTPS URL — credentials can never be scoped broader than your server.

📡 Active sessions: who is online

Settings → Security lists **every device signed into your account**: description, sign-in method, sign-in time, last use, an **online marker** and a “this device” marker. Online means a live realtime connection, tracked per session with 60-second heartbeats — not a guess. Every session except the current one can be **signed out with one click**. A stolen laptop or forgotten browser can be logged out precisely — without touching the account.

📱 Devices & keys under your hand

Settings → “Devices with message keys” shows every device of your account; each is **individually revocable**. A revoked device is logged out, loses sessions and tokens and gets **no new sender keys** — future messages skip it. New logins alert your other devices with name and time and the option “**That wasn’t me**” for one-click revocation. Operators can auto-remove devices inactive 30–365 days; abandoned devices are revoked completely, keys included.

Buttons for the hard cases

🗑️ Delete everything — the admin button

Administrators find a danger button in **Admin → Security → self-deleting messages**: “Delete all messages”. One action wipes **every message in every room, thread and DM** — with reactions, edits, attachments and link previews. The server reuses the expiry machinery: each message key shredded, body events erased, attachment bytes removed, projections updated, up to eight rooms in parallel as a background job with progress. **Accounts, rooms, memberships, devices, keys and settings stay.** Final, and in the audit log without message content.

🧹 Delete all *your* messages

Users have the same power over their own data: Settings → Self-Destructing Messages → “Delete all my messages”. The server retracts every post you wrote — including rooms you left, thread replies and their echoes — newest-first. Membership stays; content, reactions, pins, search entries, notifications, attachments and previews go. Crypto-shredded, not hidden.

⚙️ Admin → Security: inside the page

A dedicated **Security** page with three sections. **Access:** blocked usernames refused at registration; every OAuth client settable to default, trusted or blocked. **Retention:** “Delete all messages” danger button (background job with progress); system events — joins, calls, room changes — cleared immediately or following the message lifetime, securely overwritten. **Sessions:** absolute session timeout (e.g. 12 h, not just idle); **inactive devices auto-removed after 30, 60, 90, 180 days or 1 year** — revoked completely, keys, sessions and tokens, the last device of an account included; the panel shows how many devices are pending removal. Plus invitation links with expiry, password policy, login throttling, branding, audit log without content, IP-watch diagnostics.

CAPABILITIES

A messenger that does everything — encrypted

💬 Full-featured chat

Rooms, threads, groups and channels, mentions, reactions, Markdown, pinned messages, files — all end-to-end encrypted. Read receipts for direct messages, optional in both directions.

🖼️ Files & media

Images, documents, videos — encrypted, metadata stripped. Filename and type inside the ciphertext.

📞 Conferences

Voice and video with screen sharing, encrypted per call, keys only for confirmed members.

🔔 Notifications

Push carries only sender and location — never your text. UnifiedPush/FCM/APNs/Web Push supported.

🛡️ Verification

Safety numbers per person, device overview, alerts on new devices, optional auto-removal.

🧹 Controlled deletion

System messages immediately or automatically deletable — the server overwrites securely.

At home everywhere

🪟 **Windows 10/11 (x64)**	Installer; keeps all data and keys across updates.
🐧 **Linux x86_64 / arm64**	AppImage — make executable, double-click.
🤖 **Android 7+**	Signed release APK (`CN=Enigma 3`), biometric lock, Keystore vault.
📱 **iPhone / iPad**	Use the web app in Safari.
🌐 **Browser**	Web app served directly from your own server.

Updates: only from the official repository, checked against SHA-256 sums — without a matching checksum the app offers no update. **Integrity:** `sha256sum -c SHA256SUMS.txt`.

GETTING STARTED

— Three steps to your own chat

1

Run the server

A Linux machine or VM. A single **setup.sh** walks you through the setup: domains, Cloudflare, SMTP and Docker — no Docker knowledge needed, it asks for everything and configures it. You need your own domain and an SMTP mailbox. That's all the routine it takes.

2

Install the app

Desktop app, APK or web app — open “Connect server” and enter your address. Registration works by invitation link only, so the operator decides who joins.

3

Start securely

First direct message, compare the safety number, test the first call. If the call drops, the router ports are missing. Everything else is default.

Worth knowing: Enigma 3 is self-hosted software for people who want to take back control — not a free hosted service. Running it needs your own Linux machine and a domain. Technical details, the security model and installation are fully documented in the open.

Switch. Keep talking.

Messages that stay with you — on hardware you own.