HushMessenger

TECHNICAL & COMPANY WHITEPAPER

Private conversations, with a defined lifetime.

Client-side encryption, platform capabilities and access expiry, with the implementation, metadata and security limits explained.

Hush · Company to be incorporated in Qatar

8 October 2026

Contents

- Abstract

1 Who we are

2 Why this product exists

3 The concept in one picture

4 What Hush does (features)

5 The security model, end to end

6 What the server can and cannot see

7 Threat model: “what if…?”

8 Verifiability: don’t trust, verify

9 How we test the system

10 Platform capabilities and release status

11 Honest limitations and roadmap

12 Compliance and governance

13 Summary for partners and stakeholders

14 Glossary

15 Frequently asked questions

Abstract

Hush encrypts message content and attachments on participants' devices. The relay does not hold the Signal or attachment decryption keys. These properties assume authentic client code, verified peer identities and uncompromised devices; they do not cover all account or invitation metadata.

Implementation status, 8 October 2026. This document describes the reviewed source tree, with particular attention to the Web application and its relay/account services. Native capabilities are included where they share these services or have recorded validation. It is not a certification of the deployed service or distributed binaries. Encrypted username invitations reveal sender and recipient account metadata to the server; their room details remain inside ciphertext. Account sessions may record IP addresses and user agents according to deployment configuration. Technical diagnostics retain selected operation metadata. Automated checks do not establish complete security or replace device testing and an independent audit.

This whitepaper describes Hush's founder and funding objectives, the conversation workflow, implemented platform capabilities and the security model. It separates source features and recorded validation from planned investment, distribution status and remaining device checks.

---

1. Who we are

1.1 Founder and company

Gabriel Genoni is the founder of Hush. His project combines a familiar conversation list, encryption on participants' devices and conversations with a defined access deadline. Its design aims to simplify private exchanges while explaining what encryption and deletion can actually protect.

Hush is a private-messaging project aimed at professional exchanges and also available to individuals. **The company will be incorporated in Qatar.** Public legal information and the funding page are available on the website.

1.2 Funding objective: US$1.1 million

The project is seeking **US$1.1 million**, with five planned uses, consistent with the website's funding page:

- **Recruit two cybersecurity experts.** Strengthen human review of code, protocols and architecture, and develop advanced security controls with dedicated specialists.
- **Recruit a marketing director.** Develop professional interviews, positioning, pilots and partnerships, while keeping a clear path for individuals.
- **Run supervised security hackathons.** Invite researchers into authorized test environments with a written scope, test data and responsible disclosure; prioritize findings, correct them and verify the fixes.
- **Build dedicated test infrastructure.** Create isolated environments for protocol, load and failure testing, and increase infrastructure capacity according to measured needs.
- **Introduce continuous AI-assisted code analysis.** Flag anomalies and help the specialists investigate vulnerabilities in code and tests. This analysis would not access private conversation content and would support human review.

The proposed sequence is to build the team, challenge the product through scoped tests and reviews, then learn with professional pilots. These are intended uses of future funding. They do not establish awarded funding, completed recruitment, scheduled events or an operating enterprise service.

1.3 Planned incorporation in Qatar

The company will be incorporated in Qatar, where the project plans to base its specialist team and test environments. The Startup Qatar Investment Program is a potential funding route to explore; eligibility, support, funding amounts and milestones would need confirmation through its evaluation and conditions. Admission, investment and program endorsement are not established by this document.

---

2. Why this product exists

2.1 The problem

Some professional and personal conversations only need to remain accessible for a defined period. Hush combines a familiar conversation list with client-side encryption and an access deadline. Creating an account requires a username and password, without a phone number or mandatory email. This reduces required identity information; it does not provide anonymity from service or network metadata. The website describes potential uses for diplomats, journalists, lawyers, doctors, engineers, investors and entrepreneurs. These examples are not sector certifications, approved classified-information handling or certified health-data hosting. Current

plans are per account; an enterprise contract, centralized team administration and a service-level agreement are not implemented product capabilities.

2.2 The workflow

Create a conversation from an account, invite a contact by link or username, exchange encrypted messages, and let access expire. Link guests can join without creating an account, after owner admission. The conversation list remains available across authorized account devices; each device keeps its own local message history. Exports and backups have separate retention.

2.3 The current plans

The current source implements these plans:

- **Free**: account creation with a username and password; the current configuration creates **30-day text conversations**, with encrypted attachments and owner-signed admission. Creation and relay usage remain subject to rate and storage limits.
- **Pro**: a choice of **1 hour, 24 hours, 7 days or 30 days**, encrypted calls on compatible clients, and deletion before expiry. All plans use the same attachment policy: one download per recipient device recorded at upload, with server deletion after the final expected acknowledgement or at expiry.

The conviction behind that decision is one of our core principles: **strong cryptography should not be reserved for large corporations with dedicated IT teams.**

---

3. The concept in one picture

The entire design can be summarised with a single image that we will return to throughout this document:

> ***Our server is like a post office that only ever handles sealed safes.*** *It receives encrypted messages, queues them for delivery, and deletes acknowledged or expired records. It does not hold the participants' Signal message-decryption keys. The metaphor describes message content; accounts, invitations and network metadata need separate treatment.*

The **message relay** validates routing information, enforces limits and forwards encrypted packets. It cannot decrypt messages already encrypted to participants' keys using its ordinary

server-held keys. The wider service also processes account, invitation, billing and connection information (Section 6). A compromised operator could alter client delivery or public-key distribution; authentic clients and verification of the intended recipients are essential.

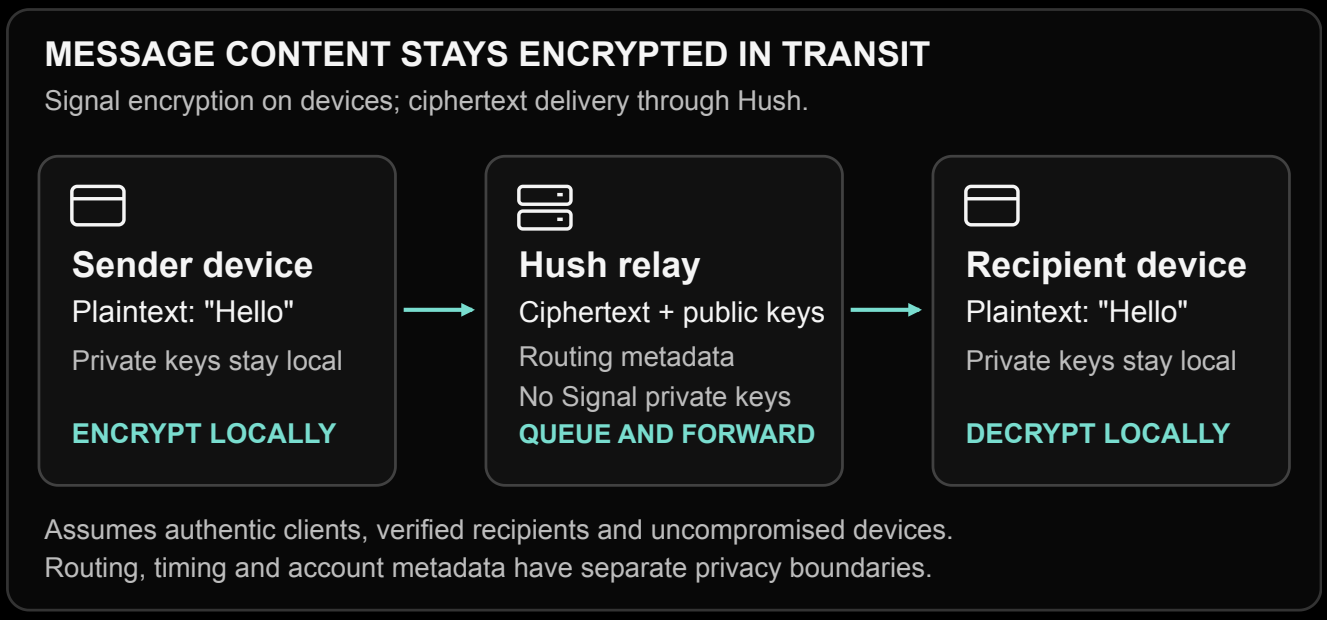


Figure 1. End-to-end encrypted message delivery: encryption and decryption happen on devices; the relay handles ciphertext and routing metadata.

Message encryption and decryption happen in participants' browsers or apps. Clients do not upload their Signal private keys in plaintext. Account vault synchronization and authorized-device recovery can transmit encrypted key material; they do not give the ordinary relay the keys needed to decrypt message content.

---

4. What Hush does (features)

4.1 Core experience

- **An account with a client-encrypted vault.** Signing up takes a username and password. A recovery email is optional. Account creation does not require a phone number or identity verification. The server receives a derived authentication key, not the password, and stores account metadata (Section 5.4).
- **A conversation thread, like a familiar messenger.** Signed-in users see a conversation list with remaining-lifetime indicators. The account list is encrypted on-device before synchronization, and ordinary server-held keys do not open it. Message history remains local to each device, rather than being a synchronized account archive.

- **Invite-link access, no account needed to join.** A random room identifier opens the conversation in a supported browser; the owner must admit the joining device before it can exchange messages. Native guests use a local password to protect this device's encrypted storage; it does not create an account or synchronize the guest history elsewhere.
- **Participant names.** Link guests may choose a temporary nickname or continue without one. A signed-in creator uses their account username as their participant name in conversations they created. Display names are exchanged through encrypted profiles after admission; the native clients show known sender names on text and file messages.
- **Multi-device conversations with separate limits.** The signed roster admits up to **100 devices per room**. A separate, configurable anti-abuse ceiling defaults to **1,000 unexpired device publications**, including pending and offline devices. Encryption fans out per recipient device, and calls use a mesh; these limits do not establish usable capacity or smooth performance.
- **Encrypted text messaging.** Every message is end-to-end encrypted on the sender's device before it is sent.
- **Encrypted file sharing.** Attachments are encrypted on the device with a unique per-file key before upload; the server only ever stores an opaque encrypted blob.
- **Expiring conversations.** Free rooms currently last **30 days**; Pro supports **1 hour, 24 hours, 7 days or 30 days**. The API refuses expired rooms. Scheduled server cleanup and device cleanup remove application records when they run; they cannot erase other participants' copies, exports or backups.
- **Read receipts and typing indicators** are carried inside encrypted control messages. Their payloads are not plaintext relay fields, but packet routing, size and timing remain visible.

4.2 Encrypted audio and video calls

- **Frame-encrypted audio/video calls** in supported Web and native runtimes. The current clients refuse calls without frame-encryption capability or a configured key; browser/device compatibility still needs real media testing.
- **Screen sharing has platform limits.** Available in supported desktop browsers with capture permission and the native macOS app; mobile screen sharing is not available. Source implementation does not establish physical-device compatibility or deployed performance.
- **Configurable STUN/TURN.** The source supplies no third-party STUN/TURN service by default. Operators can configure their own TURN server or a managed Cloudflare TURN service. A configured provider and the hosting/network operators can observe connection metadata even when media content is encrypted.

- **Forced TURN relay mode** can hide direct peer ICE addresses from the other participants when a working TURN service is configured. It does not hide connection addresses or traffic timing from that relay/provider.
- **Continuity within the app.** The Web client keeps an active call while its interface is minimized and the user switches conversations. Native clients can retain active media during a background transition while blocking private interface access until a fresh unlock on return. Actual OS suspension, microphone/camera permissions and network conditions still require device testing.
- **Call workspace.** One-to-one video includes a local camera preview; groups offer a gallery and participant selection. Users can mute the microphone, toggle the camera, show the discussion, minimize the call and hang up. Supported Web browsers offer full screen; native clients also expose camera switching. Screen presentations are displayed without cropping and take priority when a new presentation arrives, while users can still select another view. Incoming calls for a conversation that is not open have a separate in-app notification. Call events are carried through encrypted chat controls and can appear in local history.

4.3 Identity verification (anti-impersonation)

- Each contact is assigned a **Signal-style "safety number"** that both parties can compare out-of-band (read aloud, etc.).
- Link entrants wait for an owner decision in an automatic Accept/Reject dialog. Acceptance signs the exact device identity into the room membership; it does not prove who controls that device. Username invitations carry an encrypted, targeted admission capability, which lets the owner client admit the invited device automatically without another dialog. Safety numbers and their guidance are hidden by default; clicking a participant opens them in a separate dialog for optional comparison. There is no comparison attestation or code validation button. The settings page does not display the user's own account fingerprint; cryptographic account keys and identity-pinning checks remain in use. A directory key change requires a new owner-signed admission before exchanges resume; a changed previously pinned account key blocks account invitations. A reinstall or reset can also change a key; the warning is not proof of an attack.

4.4 Access control

- **Owner admission for links; automatic admission for targeted username invitations.** Protocol 3 requires an owner-signed roster in both cases. Other admitted devices automatically follow the exact signed identities, without a second local approval. Automatic username admission requires an online owner client to verify the invitation capability and sign

the roster. Initial account keys use trust on first use. Safety-number comparison through a separate trusted channel remains available for identifying the intended contact; the client does not require or record a comparison action.

4.5 Plans

Plan	What you get
Free	**30-day** text conversations, encrypted attachments, owner-admitted link access and safety numbers available on demand. Subject to rate, storage and device-presence limits. Each recipient device recorded at upload can download once; the server copy is deleted after the final expected receipt or at expiry.
Pro	Free features plus a choice of **1 hour, 24 hours, 7 days or 30 days**, frame-encrypted audio/video calls in supported runtimes, and deletion before expiry. The creator holding the close secret closes the conversation for everyone; a participant removes it only from their own synced list. The file policy is identical to Free.

Pro subscriptions and billing management use the website and Stripe. Native apps explain that subscription takes place on the website and refresh the account's plan. The creator's plan sets the room's capabilities at creation; invited participants do not need to purchase Pro or create an account to join.

The Web settings show the current plan, trial or paid-period deadline and renewal state. Monthly and annual subscriptions are offered; source defaults are **US$99 per month**, **US$999 per year** and a **seven-day first-subscription trial requiring a payment method**, subject to configured Stripe prices and eligibility. Stopping renewal preserves access until the confirmed period end; a pending cancellation can be reversed while the subscription remains eligible. These are separate operations from immediate account deletion. A later plan change does not rewrite an existing room's duration or enable calls in a room created without them. Early room closure requires a valid Pro creation pass and the creator's close secret; it can also close a room originally created on Free.

Payment fields are rendered through Stripe on the dedicated Web upgrade page. The client deliberately reaches that page with a full load, whose CSP permits the payment integration. Stripe processes payment and associated metadata; the ordinary conversation interface has no Stripe payment form. Subscription changes are applied through signature-verified webhooks, and the return path checks that the Checkout session belongs to the signed-in account. No live payment was made for this review.

Deleting an account on the Web or through the native account API first closes its tracked payable Checkout sessions and confirms immediate cancellation of its linked Stripe subscriptions. A payment completing during Checkout expiration is reconciled to its resulting subscription before

deletion. This operation requests neither a new invoice nor prorations, and does not automatically refund earlier payments. If Stripe cannot confirm that billing has stopped, the account, encrypted vault and authentication remain available for a retry. Accounts predating Checkout tracking require a verified reconciliation of their billing history; deletion can be temporarily unavailable while that history is unknown. These paths are covered by offline Stripe SDK tests, not a live billing exercise.

4.6 Web workflow and account controls

- **Organize conversations.** Search the decrypted conversation list by its display name, rename entries and view unread and remaining-lifetime indicators. Rename and favorite changes are stored in the encrypted account vault and synchronized with conflict handling. Search is performed on-device; it is not a server-side message-history search. The desktop conversation menu can collapse without remounting the discussion or interrupting the call, preserving its current search and draft.
- **Favorite contacts and start by username.** Save favorite usernames and start a conversation with a selected contact. The authenticated directory lookup exposes the queried username/account to the server and can report that a username exists. A recipient must have published a compatible invitation identity; a changed previously pinned identity blocks the invitation.
- **Share or paste a complete invitation.** The sharing sheet generates a QR code, copies the link and uses the OS/browser share action when available. New protocol-3 links include the owner's public authority key in `#hush=…`; new devices need that trust anchor, rather than a bare room ID. The fragment is omitted from the page's HTTP request, but the recipient, clipboard and external share destination can see the link. A trusted delivery channel is still needed. Owner signing and close secrets are not included in the invitation.
- **Account security settings.** Add, replace or remove the optional recovery email, change the account password and delete the account after checking the current derived authentication key. Registration presents a separate recovery key for the strict vault. On a new browser, password sign-in and vault recovery are separate steps; an already authorized browser can restore its locally protected master. Losing every authorized copy and the recovery key leaves the strict vault inaccessible even after an email password reset.
- **Distinct deletion scopes.** Before expiry, the Web deletion action is Pro-gated: a creator with the close secret closes the room for everyone; another participant removes the entry from their own account list. Expired entries can be cleaned up without Pro. Account deletion removes account records and local application data; it does not automatically close every room or erase other participants' histories and exports.

- **Messages and attachments.** The Web discussion shows participant names, typing state and sent/delivered/read states, preserves the current draft when its workspace remains mounted, and asks for confirmation before fetching an attachment. Recognized Web, email and phone links are rendered locally; HTTP links open with `noopener noreferrer`. Following a link or saving a file involves a destination outside the encrypted relay's control. Attachments are shared with the recipient devices selected at upload; a device joining later is not automatically given access to earlier attachments or messages.
- **Language and appearance.** English, French and Spanish are implemented, with light, dark and system appearance choices, responsive layouts and selected reduced-motion, contrast and accessible-control support. These features do not constitute a full accessibility certification.

4.7 Transparency, diagnostics and administration

- **Public information.** The website provides feature and pricing pages, application availability, founder and funding information, the downloadable whitepaper, legal documents and an account-free security self-check page. Public presentation pages have language-specific SEO metadata and a sitemap; accounts, conversations, administration and the audit page are excluded from the application's search-indexing policy. That policy is not access control.
- **Contact form.** Visitors can send a general, Pro, security or press enquiry after accepting data processing. Validation, per-IP throttling and a honeypot reduce abuse; the service sends the enquiry and a best-effort confirmation by email. This support channel is readable by the service and mail providers, with separate retention; it does not use conversation E2EE.
- **Technical journal.** Signed-in Web users can inspect, copy, export and clear a local technical journal in settings; guests access it through the participants menu. Native diagnostics also provide local exports. It records operation names, timings, failures and selected media statistics, with an allowlist that excludes message/file contents, private keys, account/room/device identities, SDP, ICE candidates, IP addresses, URLs and raw exception messages. The journal is not automatically uploaded. Server diagnostics are separate, enabled by default in source and privately stored; Section 6 describes their limits.
- **Operator console.** A separate administrator role can inspect aggregate account/room/device metrics and active room identifiers, run selected diagnostics, manage TOTP and recovery codes, and purge one or all live rooms for incident response. A behavior laboratory can run synthetic scenarios against the configured service, available by default only in the local environment unless explicitly enabled elsewhere. These tools do not decrypt private conversations. They do give the administrator service-disruption and deletion powers; a compromised admin session is an availability risk.

---

5. The security model, end to end

This is the heart of the document. We will walk through exactly what happens when a message travels from one person to another, then break down each protective layer.

5.1 A complete scenario: Alice sends "Hello" to Bob

Step 1 - Alice and Bob join a room. Alice creates a conversation from her account. It receives a random UUID, e.g. `/r/8f3a…`. The room row has no creator-account reference, but creation-pass issuance/use and username invitations can correlate an account with that room (Section 5.4). Bob can open the invite link without an account.

Step 2 - Each device generates its own keys. Alice's browser and Bob's browser each create a set of cryptographic keys. They publish **only their public "locks"** to the server - never their private keys.

Step 3 - Alice admits Bob's device. Bob is pending after opening the link. Alice accepts the request, and her client signs Bob's exact device identity into the conversation roster. Until that admission, authentic Hush clients refuse message, file-key and call exchanges with Bob. Relay transport checks have a separate scope (Section 5.7). Comparing their safety number is optional and is not required to approve the request.

Step 4 - Alice writes "Hello". Before anything is sent, Alice's device:

1. fetches Bob's public key material,
2. establishes a secret, secure session with him (using the Signal Protocol's X3DH handshake),
3. **pads** the message to a standard size to hide its true length,
4. **encrypts** "Hello" into opaque ciphertext such as `x9F2k…8sQ==`.

Step 5 - The server carries the sealed safe. The server receives `x9F2k…8sQ==` and cannot decrypt this message with its ordinary server keys. It sees room and device identifiers, public keys and delivery metadata. Account invitations and other service records may associate participants with usernames or recovery emails. It queues the envelope for Bob.

Step 6 - Bob collects and unlocks. Bob's device retrieves the safe, **decrypts it with the message key derived from its Signal session**, removes the padding, and displays "Hello". His private keys remain on his device.

Step 7 - The safe is destroyed. Bob's acknowledgement deletes his queued envelope from the live relay database. Undelivered envelopes become inaccessible when the room expires and are removed by the configured purge job. This does not guarantee forensic erasure from backups or from participant devices.

> ***The key point:*** *authentic clients encrypt "Hello" before sending it. The relay receives ciphertext rather than the message plaintext; plaintext also exists in the participants' application memory and local history, not just on their screens.*

5.2 The Signal Protocol: our primary encryption

For text messages and control signals, Hush uses **community implementations of the Signal Protocol**: `@privacyresearch/libsignal-protocol-typescript` on the Web and `libsignal_protocol_dart` in the native apps. These are not Signal's official `libsignal` library. They implement:

- **X3DH** (Extended Triple Diffie-Hellman) to establish a shared secret when two devices first make contact.
- **The Double Ratchet** to derive a new message key as the conversation proceeds.

The Double Ratchet advances keys as messages are exchanged. Forward secrecy depends on deletion of old keys and uncompromised implementations. Recovery after compromise requires fresh secret input and the attacker losing access; it is not immediate or unconditional. The current clients request a one-time pre-key reservation through the relay. A server transaction allocates a distinct available pre-key to each authenticated initiation, binds retries to the same device/identity context, and retains spent-key records until room deletion. An exhausted pool rejects a new initiation until replenishment; it does not silently reuse a reserved key. These controls do not prove complete protocol security.

5.3 The encryption systems, in full

The main cryptographic systems found in the reviewed source are listed below, with their intended purpose and relevant limits.

#	System	Purpose (plain language)
1	**Signal Protocol (X3DH + Double Ratchet)**	End-to-end encryption of all text and control messages, with per-message key rotation.
2	**AES-256-GCM (files)**	Each attachment is encrypted on-device with a unique key; the server stores only an opaque blob.
3	**AES-256-GCM (calls)**	Hush media v2 encrypts the media payload and authenticates the readable VP8/Opus framing header. Calls negotiate these codecs and require a valid key and installed cryptors. This custom format is not a claim of RFC SFrame compliance.

#	System	Purpose (plain language)
4	**AES-256-GCM (local-at-rest)**	Browser secrets and Signal records use a non-exportable local key. Same-origin malicious code can still invoke decryption; a complete browser-profile compromise is outside this guarantee. Native storage uses a password-wrapped master key, with OS biometric storage when enabled.
5	**Safety numbers (SHA-512, iterated)**	A 60-digit fingerprint of both parties' identity keys, comparable out-of-band to detect a man-in-the-middle.
6	**Message padding**	Plaintext is padded to standard size buckets before encryption to reduce size-based metadata leakage.
7	**Transport encryption**	HTTPS/WSS protect API and signalling traffic. WebRTC media uses DTLS-SRTP in addition to Hush's frame encryption.
8	**Selected database fields encrypted at rest**	Recovery emails, admin two-factor secrets and native push-provider tokens are encrypted with the application key, which the operator holds. Stripe customer, subscription and tracked Checkout-session identifiers remain readable in the application database.
9	**Hashing of passwords & secrets**	Derived account authentication keys and admin recovery codes are hashed; device tokens are hashed with SHA-256.
10	**TOTP two-factor (admin)**	The super-admin console supports authenticator-app two-factor, with an encrypted secret and hashed recovery codes.
11	**Unguessable identifiers**	Rooms (UUID), device IDs and tokens are long, cryptographically random values.
12	**Randomised call (TURN) credentials**	Temporary credentials do not embed a room ID. The operator still sees connections, IP addresses, traffic volume and timing, and the application server issues the credentials.
13	**CSP + Subresource Integrity (SHA-384)**	Restricts script sources and detects an asset differing from its declared hash. A compromised host can replace both the HTML and the hash; these controls do not solve malicious client delivery.
14	**PBKDF2-SHA256 + HKDF (accounts)**	The account password is stretched on the device (600,000 iterations, random salt) to derive an **auth key**, sent to the server and stored only as a bcrypt hash. New accounts use an independently random vault master; older password-derived vaults require explicit migration on an authorized device.
15	**AES-256-GCM (conversation list)**	The list of a user's conversations (names, links and admission capabilities) is encrypted with the vault key before being saved. The server stores and versions an opaque blob.

#	System	Purpose (plain language)
16	**Signed creation pass (HMAC-SHA256)**	A short-lived token proving "an account on plan X", with no account identifier, used to create conversations through the stateless relay.
17	**Ed25519 signatures (vault and room authority)**	Strict vault writes require a master-derived owner signature. Room admission uses a separate owner-signed roster binding exact device IDs and Signal identity keys. Signatures prove possession of the relevant authority, not a person's real-world identity.
18	**X25519 + HKDF-SHA256 + AES-256-GCM (username invitations)**	Encrypts targeted room details and admission capabilities to the recipient account's invitation key; Ed25519 binds public invitation keys and signs envelopes. Sender/recipient account routing remains visible. This invitation format is separate from the Signal message ratchet.
19	**HMAC-SHA256 (targeted admission capability)**	An invited account proves possession of an encrypted grant for a particular room, authority, device identity and deadline. The owner's online client verifies it before signing admission; the grant does not itself replace the signed roster.
20	**X25519 + HKDF-SHA256 + AES-256-GCM (native wake hints)**	The server encrypts a minimal hint to the native wake key to hide routing details from the push provider. The server creates this hint and knows its contents; only the authenticated Signal invitation authorizes an actual call.

5.4 Client-encrypted account vaults and account metadata

Accounts keep a client-encrypted conversation list across devices and manage Pro. The server cannot open that vault using the stored authentication hash alone. However, username invitations expose sender and recipient account metadata, and signed creation passes can be correlated between issuance and use. The service does not offer cryptographic account-conversation unlinkability.

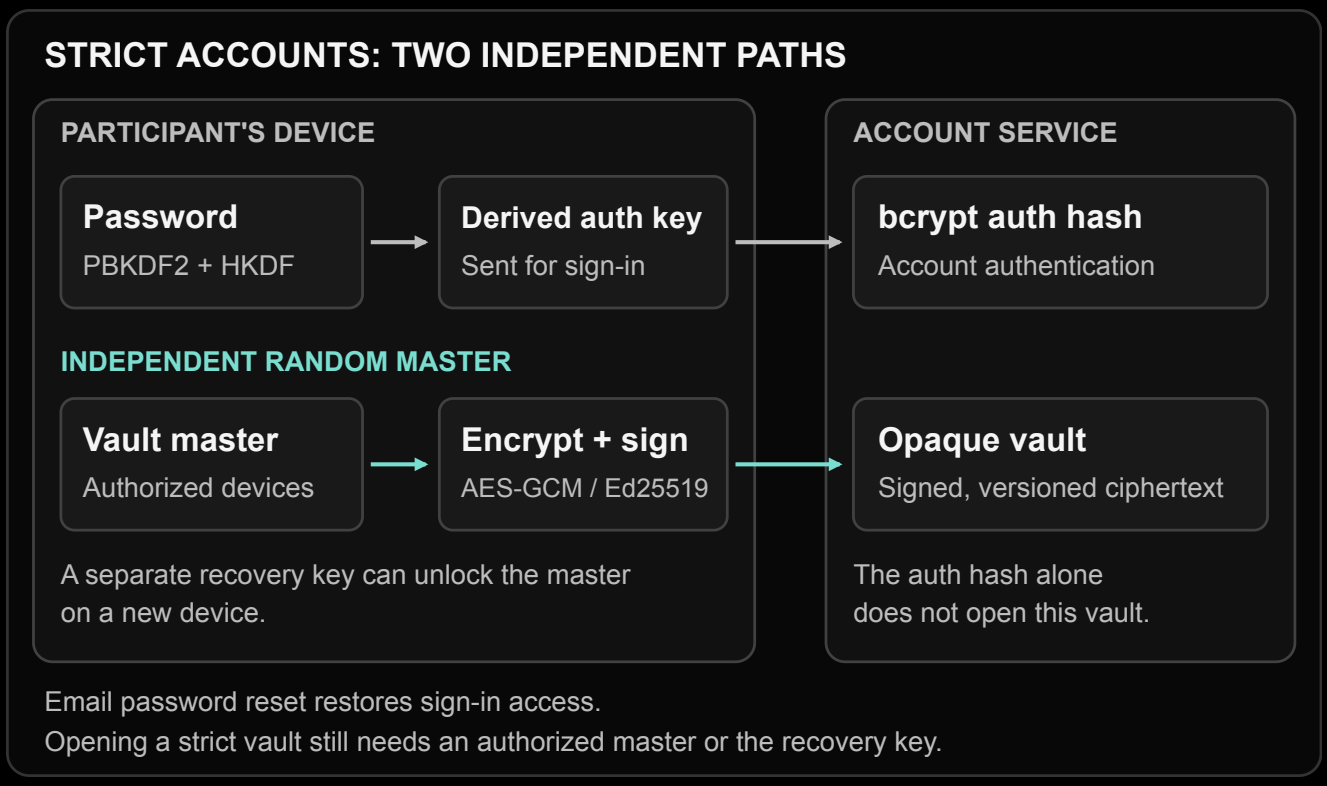


Figure 2. Strict accounts separate sign-in credentials from the independent vault master and its recovery key.

- **The legitimate client sends a derived authentication key, not the password.** It derives that key with PBKDF2-SHA256 (600,000 iterations, random salt) and HKDF; the server stores a bcrypt hash. Strict accounts encrypt their vault with an independently random master held by authorized devices; a separate recovery key permits opening it on a new device. Malicious client delivery or a compromised endpoint can defeat this separation.
- **The conversation list is a sealed safe too.** Names, invite links, favorites, pinned account identities, admission capabilities and the creator's room-authority/close secrets are encrypted with the vault key (AES-256-GCM) before being saved. Strict writes also require an Ed25519 owner signature bound to the account, key identifier, expected version and ciphertext digest. Possession of a login session alone does not permit a valid vault write. The server stores an opaque, versioned blob and rejects conflicting writes; the client decrypts and merges them locally. Removal tombstones prevent an older device copy from silently restoring a removed list entry. These checks do not give the server plaintext access or provide a global proof against a malicious server presenting different histories to fresh devices.
- **No room row references an account.** Conversations are created through the same stateless relay as everything else, with no session cookie. To prove that the creator has an account (and which plan), the browser presents a **creation pass**: a signed token carrying the plan, an expiry and a random nonce, without an account id. It is reusable for up to 12 hours, capped by the Pro period end. It is a bearer credential: exposure permits its holder to exercise

its allowed relay actions until expiry. Changing account credentials does not immediately revoke an already issued stateless pass.

- **The account Web conversation route uses a URL fragment** (`/app#/c/…`), which is not included in that page's HTTP request. Relay API calls still name the room, and invite links use `/r/{id}` . Fragments do not make room activity unlinkable to sessions, invitations or connection timing.
- **Recovery email, only if wanted.** It is stored encrypted, with a keyed hash for lookup. Resetting an account password by email changes sign-in access; it neither opens nor replaces a strict vault. Opening that vault on a new device still requires an authorized master or the saved recovery key.
- **Authentication credentials remain sensitive.** A derived auth key can authenticate to the account even though it is not the password and does not derive a strict vault master. Web account routes use session cookies and CSRF protection; native account tokens are separately hashed and expire after a configurable 30 days by default, and are not accepted as relay device tokens. Password changes and resets revoke native account tokens and rotate the credential binding used to reject old Web sessions on their next authenticated request. This does not remotely erase offline local copies or revoke already admitted room identities.
- **Recovery and logging boundaries.** Password-reset tokens are single-use, hashed in storage and valid for 60 minutes; expired reset rows are separately purged after two hours. Responses avoid confirming whether a recovery email exists. In production, reset mail refuses logging, array and unreviewed mail transports, including unsafe fallback chains. The mail provider and mailbox still receive the recovery link. Sensitive credential/ciphertext fields are excluded from flashed form input, and sensitive account/relay/billing exceptions use sanitized application logging.
- **Signing out removes application records.** Keys, Signal sessions, history and vault keys are removed from application storage. This cannot guarantee forensic erasure of browser/OS backups, swap, exports or other participants' copies. Native expiry cleanup runs when the OS permits the app or background job to execute.

5.5 Anonymity: protecting the *context*

Encryption protects the *content*. We also work hard to minimise the *context* - who, when, and from where.

- **Guests do not need an account.** A room link opens the conversation, then the owner admits the exact joining device identity. Relay room/device records do not directly contain account IDs or emails, but separate invitation, account and session records can expose associations.

- **The public relay is stateless.** Public read-only pages and relay routes do not start the account session. Account and admin routes do use session cookies; the legitimate Web relay client omits cookies from its API calls. This does not remove connection metadata or prevent operator-side correlation.
- **Device display names travel in encrypted messages.** This is separate from account usernames and invitation sender/recipient fields, which the server does process.
- **No advertising/analytics pixel is part of the reviewed application.** This source property does not establish what a particular deployment, proxy or hosting provider logs.

5.6 Ephemerality: scheduled self-destruction

- Room lifetime is enforced **server-side**: currently 30 days for Free, and 1 hour, 24 hours, 7 days or 30 days for Pro. The API rejects expired rooms and constrains creation to the plan's configured durations.
- The source schedules `messenger:purge-expired` **every minute**, removing expired rooms, public-key bundles, queued envelopes, file blobs and expired invitations. Expired API reads also attempt cleanup. The deployment must actually run the scheduler; outages or storage failures can delay deletion.
- Recipient acknowledgements remove the corresponding queued envelopes from the live relay database. Delivery does not imply removal from device history or server backups.
- **One download per recipient device on every plan.** Upload fixes the authorized recipient-device set. The client requests consent before fetching and decrypts locally before export. A fetch alone does not consume access; an acknowledgement blocks repeat fetches by that device. The server deletes the blob after the last expected acknowledgement or at expiry; a failed deletion retains metadata for retry. Native clients acknowledge after the save panel reports success; the Web client can confirm download initiation but cannot prove the browser's final disk write. Failed acknowledgements can be retried without exporting again. Saved copies remain under their recipients' control.
- Web and native cleanup remove expired conversation records, keys and local history when their code executes. An offline or suspended device may not run cleanup at the exact expiration time. External exports, OS/browser backups and other participants' copies are outside this cleanup.

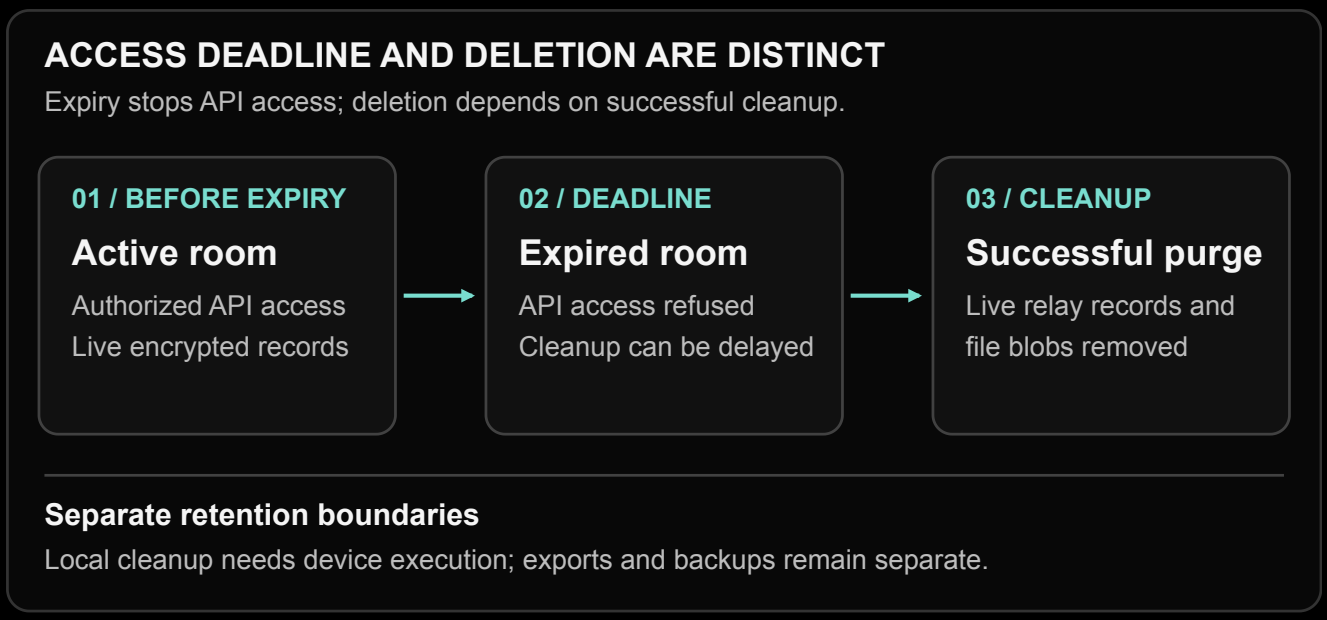


Figure 3. Room expiry ends API access; successful cleanup removes live relay records, while device execution and external retention have separate boundaries.

> ***Example:*** *a one-hour Pro room stops accepting API access at expiry. Live relay records and file blobs are deleted by the next successful purge. Account, billing, session or operational records, and external copies, may have different lifetimes.*

5.7 Defence in depth

No single control is trusted to do everything. Layers reinforce each other:

- **Device-ownership proofs** bind device operations to issued credentials and signed identities. They reject unauthorized access in the implemented flows; compromised credentials, clients or owner keys remain separate risks.
- **Per-IP rate limiting and storage quotas** resist flooding and disk-exhaustion attacks; sign-in attempts are limited per IP and per username.
- **Separate device limits** bound signed membership and candidate storage: up to 100 admitted devices and, by default, 1,000 unexpired device publications per room. Offline and pending publications count towards the latter limit. Neither limit is a performance benchmark.
- **Signed owner admission** binds permission to communicate to each exact device identity. Authentic clients require both identities to be admitted before sending or decrypting an exchange. Possession of a link alone does not admit a device.
- **Pinned room authority and ordered roster updates.** The public authority is taken from a trusted complete invitation or an existing local pin, rather than adopted from the relay directory. Ed25519-signed membership statements use a sequential epoch and previous digest. Clients reject a changed root, rollback, conflicting epoch or missing chain step relative

to their saved state. The server serializes membership updates and checks the expected head on envelope writes. These controls do not authenticate the initial invitation channel or establish a global key-transparency service.

- **Authenticated media and replay checks.** Hush media v2 authenticates the readable codec framing as additional data and encrypts the payload. Encoder instances use a random nonce prefix and a non-wrapping counter; receiver windows reject repeated or stale authenticated frame nonces. Malformed packets, wrong keys and unsupported formats are refused. The nonce namespaces are probabilistically separated, and these are custom implementation controls requiring review, not a formal protocol proof.
- **Nonce-based Content-Security-Policy** and **Subresource Integrity** restrict script execution and check assets against the hashes declared by the page. A compromised origin can replace that page and its hashes; these controls do not authenticate a malicious release.
- **Restricted integration and session boundaries.** The production CSP restricts scripts and stylesheet elements, blocks object embedding, framing and base-URL overrides, and adds Stripe permissions only on the payment page. Account/admin POST actions regain CSRF protection on their dedicated session routes. Source session defaults use encrypted Redis payloads, HttpOnly/SameSite=Lax cookies and secure cookies in production; deployment overrides and database session columns require separate verification.
- **Protocol compatibility gates** reject clients declaring an older cryptographic version, require protocol 3 for new rooms, disable legacy plaintext invitation endpoints and require explicit migration before editing a legacy account vault. The version header is a compatibility declaration, not client attestation; it does not prove the caller actually implements encryption.
- **A minimal server surface**: no decrypt endpoint, no conversation export, no history dump exists anywhere in the API.
- **Admin access controls** separate administrative routes from the public relay and account routes and require the administrator role. Login, configurable TOTP, one-use hashed recovery codes and failed-login controls protect the console. Production first-admin registration requires a setup secret and closes once an administrator exists. TOTP remains configurable rather than mandatory for every account; ordinary messenger accounts do not currently expose a TOTP setting. Absence of a built-in account-to-room view does not prevent an operator with database, logs or application-key access from correlating service records.

Admission is enforced by the clients before encryption and decryption. The relay verifies device credentials and the current membership head for message submissions, but does not check every sender and recipient against the signed roster. It can therefore store packets or encrypted file blobs submitted by an authenticated pending device. Accepting these packets does not itself

grant decryption keys. Revocation does not erase secrets, history or exports already held by a device.

---

6. What the server *can* and *cannot* see (full transparency)

The main categories found in the reviewed source are listed below. Deployment-specific logging, backups and external providers need a separate operational inventory.

Ordinary server-held keys do not decrypt:

- Signal message/control ciphertext encrypted to the intended participants' device keys.
- Attachments encrypted on-device with per-file keys shared through those encrypted messages.
- Supported frame-encrypted call media, provided the client/key setup is authentic and correct.
- The client-encrypted account vault using its stored authentication hash alone.

These properties assume authentic clients, verified recipient identities, sound implementations and uncompromised endpoints. The account password is not sent by the legitimate client, but offline guessing and malicious client delivery remain possible attack paths. Server-managed encryption protects some fields at rest while leaving them decryptable by the operator holding the application key.

The server necessarily handles, for delivery to work:

- **Opaque ciphertext** and public key material.
- **Pseudonymous routing metadata**: room and random device identifiers, sender/recipient routing and delivery timing. Live relay records are removed on acknowledgement or by the scheduled expiry purge; backups and operational logs have separate retention.
- **Admission and key-distribution metadata**: public room authority, signed member identities, membership epochs/digests, signed rejection records and pre-key reservations/spent-key records. These describe room-device relationships even though participant display names travel encrypted. Presence timestamps and queue state also reveal activity.
- **Network-level connection data** includes IP addresses and timing. Database-backed account sessions can persist IP and user-agent data; proxy, application, TURN and hosting logs depend on deployment. Guides cover minimisation, optional Tor access and forced relay, but these must be verified operationally.
- **Approximate ciphertext size**, mitigated (but not perfectly eliminated) by client-side padding into size buckets.

- **For account holders:** the username, KDF salt/iterations, a bcrypt hash of the auth key, public owner/invitation keys and key identifiers, a recovery envelope encrypted by the client, size/version of the encrypted conversation list, plan and renewal/trial state, Stripe customer, subscription and tracked Checkout-session identifiers, and the recovery email (server-encrypted) if one was given. Password-reset records and hashed native account tokens have their own access lifetimes. Authenticated username lookup intentionally reveals whether a contact exists.
- **Creation passes and invitations.** The server sees the exact signed token at issuance and room creation, so it can correlate these even through Tor or a VPN. Encrypted username invitations identify sender and recipient accounts, timing and size; their room details and admission capability remain inside ciphertext. Blind creation credentials remain future work.
- **Native push subscriptions, when configured.** The service associates a room/device with a provider, encrypted provider token, public wake key, wake identifier, expiry and delivery status. The application key can decrypt the provider token. APNs/FCM receive routing tokens, timing and encrypted wake hints; wake encryption does not hide these relationships from the Hush server.
- **Technical diagnostics.** Web journals keep up to 2,048 local events and filter entries older than 24 hours when loaded/exported. They retain timestamps, ephemeral technical session/scope numbers, source locations, allowed errors and media counters. Server diagnostics use fresh per-request trace IDs, controller/operation names, status, duration and selected counters, with rotating private files totaling approximately 15 MiB and age cleanup on subsequent writes. They exclude request bodies, raw URLs, private identities, SQL text/bindings and raw exception messages by construction. Server tracing can be disabled with `HUSH_DIAGNOSTICS=false`. This filtering reduces exposure; it does not make operation timing anonymous, erase exported copies or sanitize independent proxy/hosting logs.
- **Support and external services.** Contact-form names, emails, subjects and messages are readable by the service and mail providers. Stripe receives the configured account/payment metadata. Password-reset mail, payment pages, optional TURN providers and native push integrations are separate processing surfaces from conversation E2EE.

This honest accounting is exactly why we publish a **self-audit** (Section 8) rather than asking anyone to take our word.

---

7. Threat model: "what if…?"

A security design is only meaningful against concrete threats. This section walks through the attacks that matter most and explains, honestly, what happens in each case - including where our protection is strong and where it depends on operators or users.

7.1 Adversary table

Threat	What the attacker tries	What actually happens
Server compromise	Read records or intercept future content	Stored Signal ciphertext lacks its participants' decryption keys, but account/invitation/session metadata is exposed. Access to the application key can decrypt server-encrypted fields. Malicious client delivery or substituted/unverified recipient keys can threaten future content.
Access to retained service records	Obtain available data	Available records can include queued ciphertext, public keys, invitations, usernames, recovery emails, billing information, sessions and logs. Ordinary server-held keys do not decrypt existing Signal messages or the client-encrypted vault. Actual availability depends on retention and backups.
Account database stolen	Guess passwords or open vaults	The database contains salt/iterations, a hashed derived auth key and an encrypted vault. Offline authentication-password guessing remains possible; derivation increases its cost but cannot make a weak password safe. Strict vaults use an independent random master and require an authorized master or recovery key. Older password-derived vaults have different exposure until migration. Account and invitation metadata is separately exposed.
Account credential or reset mailbox compromised	Take over sign-in access	An attacker can authenticate or reset the password and perform authorized account actions, potentially disrupting access or deleting the account. A strict vault still requires its master/recovery key, and valid writes require its owner signature. Resetting the password does not remotely erase or revoke room keys already held on devices.
Malicious operator	Spy on users	There is no normal message-decryption endpoint and no stored Signal private key on the relay. An operator controlling delivered client code, key distribution, invitations or deployment logs still has substantial attack and correlation capabilities.

Threat	What the attacker tries	What actually happens
Impersonation (man-in-the-middle)	Substitute a contact's key	An unsigned room-key change blocks exchanges until the owner signs the exact replacement; changed pinned account keys block account invitations. Owner admission or first-use account pinning alone does not authenticate a person; compare readable safety numbers out of band when needed.
Network observer	Inspect encrypted traffic	TLS protects HTTP payloads, and Signal protects message content. IP addresses, endpoints, packet sizes, timing and traffic volume remain observable; padding reduces rather than eliminates size leakage.
Unauthorized link entrant	Exchange messages using a known room link	Authentic clients refuse exchanges until the owner signs the exact device identity into the roster. Relay transport credentials do not themselves establish admission; the relay can still accept packets from authenticated pending devices.
Flooding / denial of service	Saturate the relay or fill the disk	Per-IP rate limits and per-sender/per-room storage quotas bound the damage.
Administrator session compromised	Disrupt or delete the service	The console can purge one or all rooms. Role checks, CSRF, rate limits and configured TOTP restrict access; they cannot preserve availability after an authorized administrative session is taken over. Purging does not erase independent device copies.
Compromised hosting / CDN serving bad code	Inject code that steals keys in the browser	CSP and SRI restrict script loading but cannot authenticate HTML and hashes supplied by a compromised origin. Independent release verification remains necessary.
Lost or stolen device	Read local conversations	Browser storage encryption does not resist malicious same-origin code or a full profile compromise. Native storage uses an encrypted master/keyring and biometric storage when enabled, but its password-wrapped fallback is vulnerable to offline guessing of a weak password after file extraction. Revocation prevents later authorized sync; it cannot remotely erase an offline copy. An unlocked or compromised endpoint can expose plaintext.
All vault keys lost	Recover a strict account's conversation list	Email can restore sign-in access, but neither the password nor ordinary server keys reconstruct the independent vault master. Without an authorized copy or the recovery key, vault recovery is unavailable.

Threat	What the attacker tries	What actually happens
Compromised ratchet state	Decrypt captured ciphertext	Deleted earlier keys may protect past messages. Current/skipped keys and ongoing endpoint access can expose messages; recovery requires subsequent uncompromised ratchet input.

7.2 Where the user and operator share responsibility

We are deliberately clear that some risks are **not** fully solvable in application code:

- **Network-layer metadata** exists at connection level and may be stored in sessions or operational logs. Minimisation must be checked in the actual deployment.
- **Endpoint security** (a malware-infected laptop, an unlocked phone, a shoulder-surfer) is outside our reach. If the device itself is compromised, no messenger can save the conversation.
- **"Harvest now, decrypt later"** against archived ciphertext is addressed only partially today; see the post-quantum roadmap in Section 11.

We describe these openly because a threat model that pretends to cover everything is not a threat model - it is a sales pitch.

---

8. Verifiability: don't trust, verify

The Web client includes **built-in security self-checks** that can be run without an account.

- The displayed check total depends on the source version and runtime. Checks cover selected server controls and browser capabilities.
- It inspects selected cryptographic/runtime capabilities, headers, source/configuration patterns, route isolation, expiry configuration and relay schema properties. Some checks are static or self-reported. A pass does not establish absence of identifying data in other tables/logs, actual scheduler execution, authentic release delivery or complete security.
- It is reachable **without any login**, in the spirit of full transparency, and categorises each result as pass, warning, or fail across **infrastructure, cryptography, and anonymity**.

The repository includes a Web build script comparing declared SRI hashes across two builds and an npm dependency/SBOM-generation script. These are useful checks when run and reviewed. They do not prove that the live host serves that build, cover all native dependencies, or replace independent release-signature verification.

---

9. How we test the system

The repository contains PHP server tests, Web tests, Flutter tests and separate interop/native checks. Test counts change with the source revision. A passing suite establishes the exercised behaviours; it does not establish complete protocol security, deployment safety or hardware compatibility.

9.1 Server-side tests

These verify the behaviour of the "post office". Representative areas:

Area	What is verified
Data minimisation	Relay rows avoid account identifiers; accounts, invitations, sessions and billing have separate metadata.
Security self-checks	Check configured controls and report warnings; they are not independent security certification.
Relay envelopes & messaging	Wire-format validation, routing, access controls and limits. The relay cannot prove that arbitrary submitted bytes were encrypted correctly; authentic clients perform encryption.
Key bundles	Only public keys are accepted; device-ownership proofs are enforced.
Files	Opaque encrypted uploads, fixed recipient set, repeat access refused after a receipt, and purge after the final receipt or expiry.
Room admission	Signed-roster validation, device credentials and membership-head conflicts. Client checks enforce admission before usable exchanges; relay transport checks have the scope described in Section 5.7.
Validation	Malformed API fields and access violations are rejected. Device credentials use request headers; account password-reset links intentionally carry a one-use token.
Content-Security-Policy & asset integrity	Security headers and SRI hashes are present.
Admin & two-factor	Protected login, TOTP, account lockout.
Accounts & vault	Only derived keys reach the server; versioned, conflict-safe encrypted list; reset, rate limiting.
Strict ownership and credential rotation	Owner-signed vault writes, prohibited plaintext private-key fields, migration/version gates, session invalidation after password changes and separation of email reset from vault recovery.

Area	What is verified
Plans & payment	Lifetimes, calls and early deletion gated by plan; a shared file-download policy, creation pass without account id, and Stripe subscription lifecycle.
Logging, recovery mail and diagnostics	Sensitive exceptions and flashed form input are filtered; unsafe production reset-mail transports are refused; runtime traces omit canary secrets and rotate within their configured limits.
Calls (TURN)	Claimed-device access, room plan policy and temporary random credentials; network metadata still exists.

9.2 Web and Flutter tests

These verify what happens **on the user's device**, where encryption actually occurs:

Area	What is verified
Encrypted messaging	Real encrypt/decrypt round trips and rejection of changed pinned identities. Initial identity authenticity needs out-of-band verification.
Room admission	Clients check the signed identities before sending or decrypting messages and file-key envelopes.
File transfer	AES file encryption and secure key sharing work.
Safety numbers	Both devices compute the same number (impostor detection).
Call frame encryption	Frame-codec round trips, tamper rejection and configured cryptor guards in automated fixtures. These checks do not establish live-media encryption on every browser or device.
Media replay and call ordering	Repeated/stale authenticated frame rejection, nonce exhaustion guards, serialized signal batches, offer collisions, screen-track changes and cleanup of call observers.
Padding	Message sizes are correctly normalised.
Key bundles & encoding	Key selection is consistent; data conversions do not corrupt.
Account keys & vault	KDF vectors, key separation, authenticated vault encryption, conflict handling, password rotation and expiry. Non-exportability of a local browser key is a separate storage property, not a guarantee for every derived key buffer.
Call lifecycle & native app	Selected lifecycle, frame-encryption guards, identity checks and storage behaviours with automated fixtures. These do not validate real microphones, cameras, ICE/TURN, background calls or every OS integration.

Area	What is verified
Cold startup and biometric protection	Startup remains locked, cancelled or stale proofs cannot open a later session, and old Apple Keychain items cannot silently bypass the protected item. Password recovery preserves the encrypted vault.
Connection lifecycle and private interface	Stale background failures and transient network events do not produce a lasting offline state. A real loss or current server failure remains visible below the status bar. File-picker return and privacy masks preserve their distinct lifecycle rules.

Validation rerun for this Web/source review, 8 October 2026: all **752 Web tests** and **712 PHP server tests** passed, including 4,464 server assertions. The server suite used an isolated in-memory SQLite database and offline provider fixtures. No live billing, production purge or deployment was performed. These totals measure exercised assertions, not a percentage of all possible scenarios or a security certification.

Latest recorded native campaign, 7 October 2026: the runtime-logging report records **492 Flutter tests passed**, with one optional live-relay interoperability test skipped in that run, successful targeted analysis, and Web/macOS/Android/iOS-simulator debug builds. Earlier separate interop runs are recorded in the call-robustness campaign. Chromium probes exercise synthetic encrypted media and call lifecycle; isolated macOS-to-Chromium probes exercise real screen capture, including static first frames and camera/screen negotiation. The macOS development signature and Keychain profile checks remain evidence from 5 October. These existing records were reviewed, not rerun as native hardware tests for this revision. They do not establish production latency, distribution or physical biometric/background-call behaviour.

9.3 Continuous quality tooling

The current `.github/workflows/tests.yml` is configured for pushes and pull requests targeting its listed branches. It builds Web assets, runs Web types/tests and npm auditing, and runs Composer auditing, PHP analysis and PHP tests on PHP 8.3, 8.4 and 8.5. Its native job pins a Flutter revision, checks locked dependencies, runs Flutter analysis/tests and executes Web-to-Dart interoperability through an isolated local Laravel relay. The interop harness exercises messages, receipts, attachments, call-key envelopes and Hush media v2 packets. A compiled Chromium gate checks bidirectional synthetic audio/video and wrong-key rejection for both encoded-transform APIs; another exercises call lifecycle, simultaneous screen sharing and camera restarts. A macOS job checks shared Apple PBKDF2 and media authentication/replay sources. These gates do not validate physical microphones/cameras, public-network ICE/TURN, background wake-up or distributed native binaries. Native release builds/signatures, coverage reports, the build-hash comparison and SBOM generation are not executed by this workflow. The workflow configuration alone does not prove that a particular remote CI run succeeded.

10. Platform capabilities and release status

The table describes implemented source features, not availability in an app store or verified behaviour on every physical device. Native system integrations still require signed applications, OS permissions and device testing.

Capability	Web	iOS	Android	macOS
Text, groups and encrypted attachments	Implemented	Implemented	Implemented	Implemented
Audio/video in eligible Pro conversations	Requires supported frame transforms	Native WebRTC and cryptor	Native WebRTC and cryptor	Native WebRTC and cryptor
Send screen sharing	Supported desktop browsers	Not implemented	Not implemented	Native capture implemented
System incoming-call interface	In-page UI	CallKit integration	Telecom/ConnectionService and call notification	Actionable local notification
Call notification with process closed	No Web Push transport	APNs/PushKit integration; provider setup required	FCM integration; provider setup required	No remote wake adapter
Configurable biometric vault unlock	No native biometric gate	Keychain biometrics	Strong Keystore biometrics	Touch ID; signed profile required

Distribution controls in the reviewed source, 8 October 2026: the applications page exposes valid configured HTTPS store/download links, restricts App Store/TestFlight/Play links to their expected hosts, and suppresses direct APK/DMG links unless the operator confirms a signed Android release or notarized macOS release. These flags are declarations, not verification of a downloaded binary's signature. Source defaults leave native remote push disabled until the operator configures APNs/FCM and the queue/provider credentials.

Recorded release evidence: the 5 October local review had no configured store links or confirmed production-signed APK/notarized installer, and its APNs/FCM providers were disabled. Android debug and unsigned iOS simulator builds were produced. A macOS development build

was signed with Apple Development: signature verification succeeded and its provisioning profile authorized Hush's Keychain group. Additional debug builds are recorded on 7 October. No production distribution, current live-provider configuration, app-closed delivery or physical-device media compatibility was verified in this whitepaper review. The website's applications page reports its configured availability separately.

10.1 Storage and lifecycle boundaries

- The account vault synchronizes the conversation list, not an account-wide message archive. Device history, received files and exported copies have separate lifetimes.
- On the Web, secret records and message history use AES-GCM protected storage with a local non-exportable key in IndexedDB. Signed-in devices use persistent local storage for selected room state; guests use tab session storage for those values, while encrypted Signal records and local protection keys also use IndexedDB. Closing a tab is not a forensic erasure guarantee. Room/device identifiers, cached participant nicknames, appearance/language preferences and sanitized diagnostic events can exist outside the encrypted account vault. Browser storage has no native biometric gate and authorized same-origin code can invoke decryption.
- A native app starts with a locked local vault, including after complete process termination. When biometrics are configured and available, opening it requires a fresh OS proof; an enabled preference is not itself an unlock. Password unlocking remains available. Real background transitions also require a fresh unlock on return.
- Native apps mask private content during inactivity. Trusted OS file, share and permission panels are temporarily exempt so returning does not interrupt their own action. On Android, the file-picker exception spans both the result and resume events, then ends before file processing; a later real background transition locks again. On macOS, a focus-loss hint invites the user to reactivate the window; it does not bypass a locked vault.
- iOS and macOS use a biometric-protected Keychain item. Older items are never used as proof of a new unlock. Existing installations with biometrics enabled require the local vault password once to migrate and rearm protection; the encrypted master and conversation history are preserved. Changed biometric enrollment invalidates that protected item.
- macOS biometric unlocking uses the data-protection Keychain and requires a valid signature and provisioning profile. Ad-hoc builds offer the local vault password when biometric protection is unavailable. The signed development build's entitlement and profile have been verified; physical Touch ID behaviour remains to be tested.
- Native connection status distinguishes server reachability from general Internet availability. Resume rechecks connectivity; transient suspension events and results from an earlier

foreground session are ignored. A current server failure or durable loss of route still shows a connection banner inside the app's safe screen area. The local development relay persists after the Flutter session exits so cold simulator/emulator launches can reach it; this is a development fix, not a production uptime guarantee.

- Accounts, invitations and network records can reveal associations. Encrypting content does not hide all metadata or protect a compromised device.

---

11. Honest limitations and roadmap

Transparency means naming the things we have *not* solved, or have solved only partially. We would rather you hear this from us.

- **Browser support for encrypted calls.** The source supports legacy encoded streams and modern `RTCRtpScriptTransform`, with receivers armed before SDP. A local Chromium test decodes synthetic audio/video in both directions and rejects a wrong key for both APIs. This does not validate Safari or physical iOS, Android and macOS devices. Hush media v2 must be deployed across clients together; calls with older media formats are refused.
- **TURN availability and direct ICE.** The source defaults to forced relay and the API refuses calls without usable authenticated TURN credentials under that policy. This does not silently switch to a direct connection. If the operator explicitly enables direct ICE, participants may learn each other's network addresses. TURN providers still observe connection metadata in either managed or self-hosted deployments.
- **Post-quantum cryptography.** Sessions currently use X3DH. A migration to post-quantum key exchange (PQXDH / ML-KEM) is on the roadmap to defend against "harvest-now, decrypt-later" attacks on archived ciphertext.
- **Large conversations are not benchmarked here.** The signed roster is limited to 100 devices. The separate default ceiling of 1,000 unexpired device publications includes pending and offline devices. Signal encrypts per recipient device, and mesh calls increase connection count and media bandwidth with group size. The tests do not establish practical group capacity at these limits.
- **Account-conversation unlinkability is not provided.** Exact creation passes can link accounts and rooms, and encrypted invitations expose an account invitation graph. Blind creation credentials require a separate design and migration.
- **Signalling scale needs measurement.** Both the Web and native Flutter clients support configurable Reverb WebSocket notifications with HTTP polling as a fallback. Socket notifications trigger retrieval from the authoritative HTTP queue; they do not carry decrypted

content. Deployed throughput, latency, failure recovery and practical concurrency require load testing.

- **Network-layer metadata.** IP addresses exist at the network layer and may also be stored in sessions or logs. Documented mitigations require deployment verification.
- **Operational security is shared.** Passing self-checks confirm only the checks performed. They do not establish architecture correctness, independent auditing, authentic release delivery, secure hosting or endpoint security.
- **Web screenshots, exported files and shared links.** A Web application cannot guarantee screenshot/recording prevention. Clipboard/OS sharing and downloaded copies are outside relay deletion. Local technical journals are filtered, but their timing/operation metadata and any manually exported copy have a separate privacy boundary.
- **Background calls and screen sharing.** APNs/PushKit and FCM wake adapters exist and are disabled by default until configured. Current production provider setup and delivery were not verified for this revision. An OS answer from a locked device waits for a fresh vault unlock and the matching Signal call invitation before media starts. macOS has local notifications for an active process, without a remote process-launch adapter. Screen sharing can be sent from supported desktop browsers and macOS; mobile screen broadcasting is not implemented. Mesh calls incur one connection per participant pair, so unlimited fluid capacity is not claimed.
- **Screenshots and recording.** Android's native window uses `FLAG_SECURE` to request exclusion from screenshots and non-secure displays. This is an OS protection with device/version limits, not an absolute guarantee. iOS public APIs do not prevent an ordinary screenshot: Apple's notification arrives after it is taken. Privacy masking during continuous capture or background snapshots reduces exposure but does not establish screenshot prevention. Physical-device behaviour remains to be verified. See Android's guidance, Apple's screenshot notification and Apple's capture guidance.

We consider this honesty a feature, not a weakness. It is how serious security work is done.

---

12. Compliance and governance

- **Company formation:** the company will be incorporated in Qatar.
- **Data inventory:** message ciphertext and public keys coexist with usernames, optional recovery email, invitations, signed admission records, account sessions, billing records, optional native push subscriptions, technical diagnostics and support enquiries. Encryption of content does not remove this metadata or establish compliance by itself.

- **Legal transparency:** the company's legal notice, terms, privacy policy, and liability disclaimer are published in the application in multiple languages.

12.1 Data minimisation as compliance strategy

The implementation separates client-encrypted content from service metadata. The practical inventory and deletion boundaries are:

- The ordinary relay does not hold Signal message-decryption keys. It still retains queued ciphertext and metadata. The application key can decrypt server-encrypted recovery emails; Stripe billing identifiers are readable in the application database.
- Guests can join without an account. Account holders supply a username and may supply a recovery email. Current encrypted invitations reveal the sender/recipient account graph; legacy invitation records can additionally retain room routing until cleanup. Creation-pass use, relay requests, sessions, payments and operational timing can expose account-room associations. Account deletion removes live account/vault/session/invitation records through its implemented cleanup; payment-provider records, support mail, backups and independent room histories need separate treatment.
- Room expiry and purge reduce live relay retention. Account, billing, session and backup lifetimes are distinct from room lifetimes.

These are technical properties and limits, not a certification or legal assessment of the deployed service.

---

13. Summary for partners and stakeholders

1. **The relay does not hold message decryption keys.** This assumes authentic clients, correct implementations and uncompromised endpoints.
2. **The server does process account and invitation metadata.** Optional recovery email is server-encrypted; invitations and creation passes can link accounts and conversations.
3. **Application data expires.** Deletion timing depends on server jobs and device execution; external copies, exports and backups cannot be remotely erased.
4. **Multiple layers of encryption** combine: messages (Signal), files and call frames (AES-256-GCM), transport (HTTPS/WSS and WebRTC DTLS-SRTP), local storage and selected server fields. Stripe billing identifiers remain readable in the database.
5. **Users can compare peer identities** through safety numbers and run the built-in configuration/runtime self-checks. The self-checks do not certify the whole system.

6. **The configured CI checks PHP, Web, Flutter and relay interoperability.** Native release builds/signatures and physical-device media checks remain separate; the workflow configuration and automated tests do not cover every behaviour or deployment condition.
7. **We are honest about limits** - browser support, post-quantum roadmap, network metadata - because credible security is verifiable, not absolute.

> ***In one sentence:*** *The clients encrypt message content before relay delivery. Practical confidentiality also depends on verified recipients, authentic releases, sound implementations and secure devices.*

---

14. Glossary

A plain-language reference for the terms used in this document.

Term	Meaning
End-to-end encryption (E2EE)	Content is encrypted on the sender's device to recipients' keys. Confidentiality from intermediaries depends on authentic clients, verified recipients, correct implementations and secure endpoints.
Blind message relay	A relay that lacks the recipients' message-decryption keys. It still observes public keys, ciphertext and routing metadata; the wider service processes account and invitation data.
Signal Protocol	A protocol family for secure messaging. Hush uses community TypeScript and Dart implementations of X3DH and the Double Ratchet, not Signal's official library.
X3DH	The handshake that lets two devices agree on a shared secret the first time they communicate.
Double Ratchet	A mechanism that advances message and session keys; forward secrecy depends on deleting earlier keys and uncompromised implementations.
Forward secrecy	Protection of earlier ciphertext after old keys have been deleted. Retained or skipped keys, plaintext history and compromised endpoints limit the protection in practice.
AES-256-GCM	A strong, standard symmetric encryption algorithm used for files, calls, and local storage.
Safety number	A short fingerprint of both parties' identity keys that users compare to detect impersonation.
Ephemeral	Access expires after a configured lifetime, followed by cleanup of application records when the server/device cleanup runs. It does not erase every external copy.
Metadata	Data *about* a message (who, when, size) rather than its content. We minimise it.

Term	Meaning
TTL (time-to-live)	The room access lifetime. Scheduled deletion may occur later and other service records can have different retention.
PBKDF2 / HKDF	Standard functions that turn a password into keys; PBKDF2 makes each guess slow, HKDF splits the result into independent keys.
Vault	A conversation list encrypted on-device. Strict accounts use an independently random master and a separate recovery key; older password-derived vaults require explicit migration. Ordinary server-held keys do not open the vault, and compromised clients remain a risk.
Creation pass	A short-lived signed plan token without an account-ID field. The service can correlate its exact value at account-side issuance and relay-side use.
STUN / TURN	Helper servers that let real-time calls connect across firewalls; TURN can also relay media.
CSP (Content-Security-Policy)	A page-defined browser policy that restricts script loading and execution. It cannot authenticate a policy or code supplied by a compromised origin.
Subresource Integrity (SRI)	A browser check of an asset against the hash declared by the page. An origin that replaces both the page and hash can still deliver malicious code.
TOTP	Time-based one-time passwords (authenticator-app codes) used for admin two-factor login.

---

15. Frequently asked questions

Can Hush read my messages? The relay does not hold message or attachment decryption keys. Authentic client delivery and verified peers are essential. Recovery emails are encrypted with a server-held key, so their confidentiality guarantee is different.

Do I need to create an account? An account is required to create conversations or receive a username invitation. It uses a username and password, without a phone number or mandatory email. Link guests can join without an account, after owner admission; native guests still use a local password for their device storage.

Can Hush see my list of conversations? Strict vaults are encrypted on your device with an independently random master. Ordinary server-held keys cannot open them; another device requires an authorized master or your saved recovery key. Older password-derived vaults require explicit migration. Invitation metadata and creation-pass correlation can separately expose associations, and compromised clients remain a risk.

What happens to my messages after a conversation? The API refuses access at room expiry. Queued messages are removed on acknowledgement or by purge; expired room records and files are removed by the next successful cleanup. Device history is removed when the client executes expiry cleanup. Exports, other participants' copies and backups are outside that guarantee.

What if someone gets the room link? The owner must accept link entrants before they can exchange messages. Safety numbers are hidden by default; clicking a participant opens a separate dialog for optional out-of-band comparison, without code validation. Changed identities block sending until the owner signs their admission.

Do conversations have a password or require code confirmation? No conversation password or code-comparison validation is required. A link entrant waits for the owner's Accept/Reject decision. A targeted username invitation can be admitted automatically by the owner's online client. Account and native local-vault passwords serve a separate purpose and remain part of authentication and encrypted storage.

What happens when I close and reopen a native app? The local vault starts locked. Configured, available biometrics require a fresh OS proof, with password unlocking available. Returning from the app's file picker is treated as a temporary system interaction so it can finish; a later real background transition requires unlocking again. The Apple protected-item migration needs the existing vault password once before rearming biometrics and preserves conversations.

Is this as secure as Signal? We use community implementations of X3DH and the Double Ratchet. This does not establish equivalence to Signal's official implementations, audited product or post-quantum protocol. Security assurance requires protocol review, independent auditing and operational testing.

Can you hand my data to authorities? Available data may include queued ciphertext, public keys and admission records, accounts, optional recovery emails, sender/recipient invitation metadata and any retained legacy room routing, subscription information, sessions, support mail and operational logs. Current invitation room details are client-encrypted, but creation-pass and connection correlation remain possible. The encrypted account vault and Signal message plaintext are not readable using ordinary server-held keys alone.

What are the limitations I should know about? Calls require working frame encryption on the actual browser/device; unsupported runtimes are refused. Post-quantum key exchange is not implemented. IP addresses and timing remain observable; Tor or forced TURN can change who sees an endpoint address, not eliminate all metadata. Sections 7 and 11 describe these limits.

How much does it cost? The current Free configuration creates 30-day conversations. Pro enables configurable durations, supported calls and deletion before expiry, with source defaults of US$99 monthly or US$999 annually and a seven-day eligible first-subscription trial requiring a payment method. Confirm the website's configured prices before subscribing. On both plans, each recipient device recorded at upload can download once, after confirmation and local decryption. Native clients acknowledge after the save panel reports success; the Web client acknowledges download initiation. The server copy is removed after the final expected acknowledgement or at expiry. Renewal can be stopped or an eligible pending cancellation reversed from Web settings; native apps explain the website billing path and refresh the account's plan.

Can my password or a support request recover all my old messages? The account password grants sign-in access. A strict vault needs an authorized local master or the separately saved recovery key; email resets do not recover that key. The vault synchronizes the conversation list and capabilities, while message histories remain device-local. Neither support nor a new sign-in reconstructs another device's deleted history.

Does the technical journal send my conversations to Hush? The Web journal records filtered operation metadata locally and offers manual copy/export/clear controls; it does not automatically upload messages or logs. Separate server traces record selected request/controller metadata by default, with private bounded files and a disable option. Contact-form messages and any export deliberately sent to support have separate handling.

How can I verify these claims myself? Run the built-in security self-checks in your browser and review the published documentation. Their totals and results reflect the current runtime and selected checks; they do not replace independent review, signed-release verification or physical-device testing.

This document describes the reviewed source tree as of 8 October 2026. It distinguishes implemented behaviour, tests rerun for this revision, earlier native validation records and unverified deployment/device properties. It is not an independent security audit, protocol proof or certification of a distributed binary.