Security Whitepaper

Threat model, architecture, and what our servers can — and cannot — see

Version 1.0 · June 2026 · secretus.app
Contact: security inquiries — support@secretus.app

1. Executive summary

Secretus is a zero-knowledge platform for sharing secrets — credentials, keys, and sensitive files — between people and teams. Every cryptographic operation runs inside the sender's and recipient's browsers using the W3C Web Crypto API. Plaintext, encryption keys, and key material never leave the device. Our servers relay opaque ciphertext and public-key material only; they are architecturally incapable of reading user secrets, and this document explains why.

The design goal is simple to state: **a full compromise of Secretus infrastructure — servers, databases, storage, and the operators themselves — must not reveal a single user secret.** Sections 4 and 5 detail the encryption modes and the exact data visible to each component; section 6 walks through the threat model adversary by adversary.

2. Design principles

Principle	What it means in practice
Zero knowledge	Decryption keys are generated in the browser and shared out-of-band (URL fragment or P2P key agreement). The server stores ciphertext it cannot decrypt.
End-to-end encryption	Only the sender and the intended recipient can read a secret. The signaling server relays public keys and encrypted blobs, never plaintext.
Browser-native cryptography	All primitives (ECDH P-256, ECDSA, AES-256-GCM, HKDF-SHA-256) come from the Web Crypto API — no third-party crypto bundles to supply-chain-attack.
One-time delivery	Secrets self-destruct on first read. Async reads are atomic (DynamoDB conditional delete); live sessions are blacklisted after completion.
Post-quantum readiness	Maximum-security mode runs hybrid key agreement: ECDH P-256 + ML-KEM-768 (NIST FIPS 203), defending against harvest-now-decrypt-later.
Minimal metadata	Audit entries store SHA-256 hashes of secret IDs, never the IDs themselves, and expire automatically after 90 days.

3. System architecture

Secretus consists of three components. The static web application is served from a CDN (CloudFront + S3, EU). The signaling server — a hardened Node.js WebSocket/HTTPS service in AWS eu-central-1 (Frankfurt) — brokers session setup and stores encrypted async payloads. Secret transfer in live mode happens over a direct, encrypted browser-to-browser WebRTC channel; the server never sits between the two parties' plaintext.

Component	Role	Handles secrets?
Web application (CDN)	Delivers the open, inspectable JavaScript application that performs all cryptography locally.	Only inside the user's browser
Signaling server	Relays prekey bundles (public keys) and WebRTC offers/answers; issues one-time session tokens; stores async ciphertext metadata in DynamoDB and payloads in S3; mints short-lived TURN credentials; writes the audit log.	Ciphertext only — never keys or plaintext

Component	Role	Handles secrets?
WebRTC P2P channel	Direct browser-to-browser data channel (DTLS-encrypted) carrying the Signal-Protocol-encrypted secret. TURN relays are used when NAT/firewalls block direct paths; they forward already-encrypted packets.	Double-encrypted in transit (Signal Protocol inside DTLS)

4. Encryption modes

4.1 Live peer-to-peer — Signal Protocol

Live sharing implements the Signal Protocol specification: X3DH key agreement followed by a Double Ratchet session. Each party generates an identity key pair (ECDH P-256 + ECDSA signing), a signed prekey, and a one-time prekey in the browser. The X3DH exchange performs four Diffie-Hellman operations whose outputs are combined through HKDF-SHA-256 into a shared secret; the Double Ratchet then derives per-message AES-256-GCM keys with fresh random IVs, giving forward secrecy — compromising any single message key reveals nothing about past or future messages. Message numbers are bound into key derivation to prevent replay.

In maximum-security mode the X3DH handshake is hybridised with ML-KEM-768 (NIST FIPS 203): the classical ECDH shared secret and the ML-KEM encapsulation are both fed into HKDF, so an attacker must break *both* elliptic-curve and lattice cryptography to recover the session key.

4.2 Async one-time secrets — AES-256-GCM

When the recipient is offline, the browser generates a random AES-256-GCM key, encrypts the payload locally, and uploads only the ciphertext. The decryption key is placed in the share link's URL fragment (`#async_…`) — browsers never transmit the fragment to any server, so the key exists only in the link itself. Retrieval is a single atomic DynamoDB conditional delete: the first reader receives the ciphertext and simultaneously destroys it, making a second read impossible by construction. Unread secrets expire via TTL.

4.3 Team Split — Shamir's Secret Sharing

Team Split divides a secret into *n* shares with a recovery threshold of *t* (k-of-n), using Shamir's Secret Sharing over GF(256) computed entirely in-browser. Fewer than *t* shares reveal mathematically nothing — not one bit. Each share is then delivered through the channels above, so no single teammate, channel, or server ever holds enough material to reconstruct the secret.

5. What our servers can and cannot see

This is the heart of the zero-knowledge claim. The left column is everything the infrastructure processes or stores; the right column is what remains exclusively client-side. The table is verifiable: the web application ships as inspectable JavaScript, and live system behaviour can be observed at secretus.app/status.

We CAN see (and store)	We CANNOT see — ever
Ciphertext of async secrets (opaque AES-256-GCM blobs in S3/DynamoDB)	**Plaintext** of any secret, file, or voice message
Public keys: identity, signed prekey, one-time prekey bundles relayed during session setup	Private keys or the AES keys derived from them — generated and held only in browser memory
WebRTC signaling: SDP offers/answers and ICE candidates (connection metadata, IP addresses)	URL fragments (`#async_…`) — the async decryption key never reaches us by browser design
Session lifecycle: session IDs, timestamps, completion/blacklist state	Content carried over the P2P data channel (Signal Protocol ciphertext inside DTLS)
Optional secret labels (user-supplied, max 128 chars) and account email for registered users	Shamir shares or the reassembled secret — splitting and recovery are in-browser
Audit events: event type, hashed secret ID (SHA-256, truncated), timestamp; auto-deleted after 90 days	Recipient identities for anonymous link sharing

Consequence: a subpoena, a database dump, a rogue employee, or a full server compromise yields ciphertext and connection metadata — not secrets. We cannot be compelled to produce what we never possess.

6. Threat model

6.1 Assets and security goals

Primary asset: the user's secret (confidentiality and one-time integrity). Secondary assets: key material, session metadata, account data. Goals: confidentiality against all infrastructure parties, forward secrecy in live mode, single-use delivery, and minimal, short-lived metadata.

6.2 In-scope adversaries and mitigations

Adversary	Capability	Mitigation
Network attacker (passive)	Observes all traffic	TLS 1.2+ everywhere; payloads additionally end-to-end encrypted (Signal Protocol / AES-256-GCM), so TLS interception alone yields nothing
Network attacker (active, MitM)	Intercepts and modifies traffic	Signed prekeys (ECDSA) authenticate key bundles; AES-GCM authenticated encryption rejects tampering; WebRTC uses DTLS with fingerprint verification via the signaling channel
Compromised signaling server	Full read/write on relay, DB, storage	Zero-knowledge design: server holds ciphertext and public keys only. Cannot decrypt async payloads (key in fragment) or P2P traffic (keys never transit)

Adversary	Capability	Mitigation
Malicious TURN operator	Relays P2P packets when direct connection fails	TURN sees DTLS-encrypted packets carrying Signal-Protocol ciphertext — two independent encryption layers; credentials are short-lived HMAC tokens
Replay / session-reuse attacker	Captures a share link or session traffic	One-time atomic reads; per-message keys bound to message numbers; completed sessions blacklisted server-side; session tokens single-issuance with short TTL
Quantum adversary (future)	Records traffic today, decrypts after CRQC exists	Hybrid ML-KEM-768 + ECDH key agreement in maximum-security mode (NIST FIPS 203)
Brute-force / scraping attacker	Guesses secret IDs or hammers the API	Server-generated UUIDv4 secret IDs (122 bits of randomness), strict per-IP and per-connection rate limiting on HTTP and WebSocket

6.3 Out of scope

No end-to-end system can protect against: a compromised endpoint device (malware, keyloggers on the sender's or recipient's machine), a malicious recipient who legitimately receives and then leaks a secret, a user who posts the share link — including its key-bearing fragment — over an insecure channel, or coercion of the communicating parties themselves. Browser or operating-system zero-days that break Web Crypto API guarantees are likewise outside the model. We recommend sharing links and any verification words over separate channels.

7. Infrastructure and operational security

Control	Implementation
Hosting & data residency	AWS eu-central-1 (Frankfurt). Static frontend on S3 + CloudFront; signaling server on hardened EC2
Process hardening	Signaling server runs as an unprivileged user (no root); TLS certificates delivered via deploy hook with least-privilege file access
Transport security	HTTPS/WSS only; HSTS; strict origin allow-listing on both HTTP CORS and WebSocket upgrade
Abuse prevention	Per-IP rate limits (HTTP and WebSocket), payload size caps, server-generated session IDs, one-time session-token issuance
Least-privilege IAM	Instance role scoped to the specific DynamoDB tables and S3 prefix it needs — no wildcard resource access
Auditability	Append-only audit log with hashed identifiers, 90-day automatic expiry (TTL); users can review their own event history in-app
Transparency	Public live status page (secretus.app/status) with real-time checks and uptime history; this whitepaper; inspectable client-side code

8. Data retention and privacy

Async ciphertext is deleted at first read or at TTL expiry, whichever comes first. Live-mode secrets are never stored at all — they exist transiently on the P2P channel. Audit metadata (hashed IDs, event types, timestamps) expires automatically after 90 days. Account data is limited to authentication essentials (email, hashed credentials via AWS Cognito) and plan tier. All processing occurs in the EU (Frankfurt), and the product is operated with GDPR obligations in mind: data minimisation is structural, not policy-based — the most privacy-sensitive data (your secrets) is data we architecturally never receive.

9. Responsible disclosure

We welcome security research against your own sessions and accounts. If you believe you have found a vulnerability, email **support@secretus.app** with reproduction steps. We commit to acknowledging reports promptly, keeping you informed through remediation, and crediting researchers who wish to be named. Please do not access other users' data or degrade service availability while testing.

This document describes the system as deployed in June 2026 and is updated as the architecture evolves. The authoritative description of behaviour is always the shipped client code, which runs unobfuscated in your browser.