📖 Zero-Knowledge File Storage Architecture for CJIS Clients: Key Ideas and Proposed Solutions

1. The problem (one paragraph)

CJIS clients want to store files in our web app such that **we — and our cloud provider — cannot read them**, even in a full server breach or under subpoena. Encryption must happen in the user's browser before upload, and the experience must feel like a normal upload/download: no keys to manage, no plugins to install. The twist: pure zero-knowledge conflicts with CJIS *availability* rules (agencies must be able to recover records when an officer leaves or loses credentials), so the design needs a recovery path that isn't a vendor backdoor.

2. Proposed approach — the five big ideas

1. **Envelope encryption, two tiers.** Each file gets a fresh single-use AES-256 key (**DEK**). The DEK is then "wrapped" (encrypted) with the public key of each authorized user (**KEK** pair per user). Server stores only ciphertext + wrapped DEKs — envelopes it cannot open. Sharing = re-wrap the DEK for the new person; no re-encrypting the file.
2. **Browser does all the crypto; server is a blind broker.** A Web Worker (background thread, first-party JS) encrypts the file in chunks and uploads **directly to S3 via presigned URLs** — file bytes never touch our API. Our API handles only the control plane: auth, issuing URLs, storing wrapped keys and metadata.
3. **Keys feel invisible to the user.** The user's private KEK is stored server-side *encrypted*; at login, a **passkey (WebAuthn PRF — fingerprint/face)** derives the unlock secret in the device's secure hardware and decrypts it into browser memory only. Passphrase as fallback. Works from any device (blob roams; secret doesn't). Nothing persists in the browser after logout.
4. **Agency-held escrow for recovery.** Every DEK is *also* wrapped to a per-agency **KMS key in a separate AWS account**. Recovery = audited break-glass: two-person approval + MFA, alarms on every use, tenant notified. Solves the CJIS availability mandate without giving us routine access.
5. **AWS GovCloud, three accounts.** Workload (app), Escrow (recovery keys), Security (immutable audit). No single role can both reach data and unwrap it.

3. What this buys us / what it costs us

Wins	Costs / consequences
DB/S3/server breach yields only ciphertext	No server-side search, previews, AV scanning, dedup — inherent, must be sold as such
Insiders and cloud provider have no plaintext path	The JavaScript we serve becomes the trusted computing base (CSP/SRI/pipeline rigor required)
Blast radius of one leaked key = one file	Compromised user endpoint remains out of scope (agency's CJIS endpoint controls)
Upload/download bandwidth scales with S3, not our servers	Key ceremonies (attestation, escrow, rotation) add operational surface

Strong story for CSA audits (recovery + audit trail)	FIPS question may force a WASM crypto module (cost + 2–10× slower than WebCrypto)

4. The decisions we need to align on first (agenda)

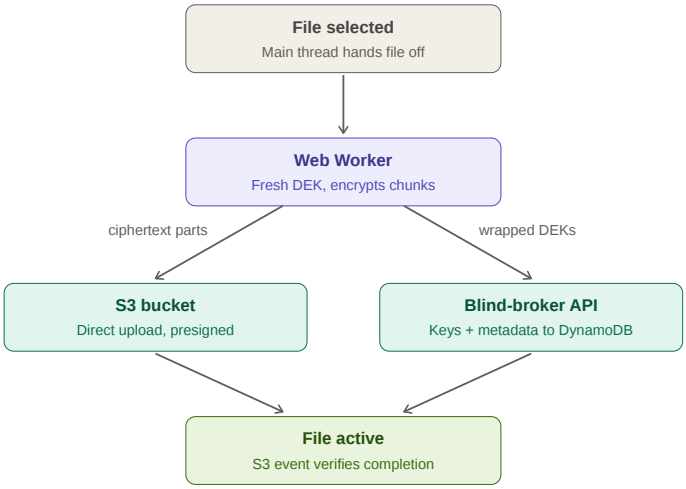
1. **FIPS interpretation (the money question):** do target CSAs accept browser WebCrypto, or demand a FIPS 140-3 validated module *in the browser* (WASM build of wolfCrypt/similar)? One written question to the CSA settles it; gates a vendor contract and real perf/isolation trade-offs.
2. **Key authenticity:** how do clients trust a public key the server hands them? Proposal: tenant-admin attestation (agency signs each user's public key at onboarding). Enough, or do clients want key transparency logs?
3. **Escrow granularity & governance:** one KMS key per agency? Who holds break-glass approval? What happens on multi-agency shared files?
4. **KEK algorithm:** RSA-4096 (matches KMS escrow envelope) vs ECDH P-384 (smaller/faster, divergent envelope format).
5. **Deletion semantics:** S3 versioning means real deletion = **crypto-shredding** (destroy all wrapped DEKs incl. escrow). Are we comfortable that shred = unrecoverable-by-design? Legal-hold interaction?
6. **Fleet reality check:** do agency-managed devices support passkeys / allow syncing? Determines how prominent the passphrase fallback and new-device flow are.

5. Explicitly parked for later (don't solve today)

Client-side encrypted search; native mobile; post-quantum wrapping migration; per-part performance tuning; DR targets; personnel-screening scope question; incident-reporting formats per state.

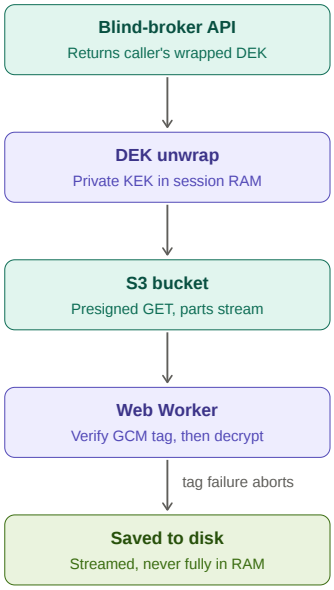
6. Strawman flows (60-second versions)

Upload: browser encrypts chunks with fresh DEK → chunks go straight to S3 (presigned) → wrapped DEKs (users + escrow) and metadata go to our API → API verifies with S3, activates file.



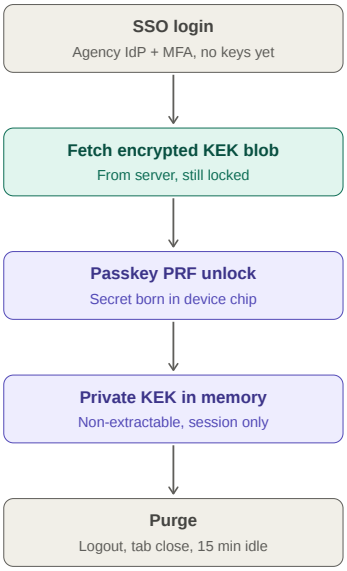
Upload. Ciphertext and keys travel separate paths and converge only at file activation: encrypted parts go directly to S3 via presigned URLs (the file bytes never transit our servers), while the API receives only wrapped DEKs and metadata. The raw DEK and the plaintext never leave the Web Worker.

Download: API returns caller's wrapped DEK → private KEK (in session RAM since login) unwraps it locally → ranged GETs stream ciphertext straight from S3 → worker verifies each GCM tag, decrypts, streams to disk; any tamper aborts.



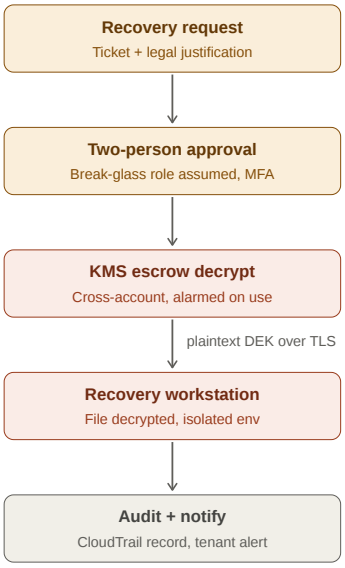
Download. The key arrives before the data: one small API call returns the caller's wrapped DEK, then bulk transfer streams directly from S3. Every part must pass its GCM tag check before any plaintext is released; a single failure aborts the download as a tamper event. Note the absence of any "server decrypts" step in either flow — no component ever holds both the data and the means to read it.

Login: SSO → fetch encrypted private-key blob from server → fingerprint (passkey PRF) unlocks it in memory → purge on logout/idle. Server never sees any secret.



Two things travel the network (a JWT and a locked blob); the thing that opens the blob never does — it is born in the device's secure chip at each login. First login on a new device is the same flow via synced passkey or passphrase, plus a one-time "enroll this device's passkey" step (Figure 7 in the full doc).

Recovery (break-glass): ticket → two-person approval → KMS decrypt in escrow account → alarms + audit + tenant notification.



Break-glass recovery. The only path to plaintext without a user key: gated by two-person approval and MFA, executed against the escrow account's KMS key (which alarms on every use), performed on an isolated recovery workstation, and fully recorded in CloudTrail with tenant notification. See OQ-10 for the recovery-workstation standard

7. Proposed next steps

1. Send the FIPS + personnel-screening questions to one friendly CSA in writing (unblocks the biggest unknowns in parallel).
2. Spike: Web Worker + WebCrypto chunked AES-GCM + presigned multipart against a sandbox bucket (~2–3 days, derisks the whole client pipeline).
3. Second session: walk the full design doc with these decisions penciled in.

---

erything here is expanded — with threat model, schemas, API contract, lifecycle, compliance mapping, and 15 open questions — in the full design doc. This page is deliberately the 20% needed to argue about the right things first.

Ev