How Keylay Works

encrypted multisig coordination through a hostile relay

Gordian Developers · v0.7 (security review R3) · keylay.org

01 · THE PROBLEM

Coordination is the leaky step

- **Setup:** participants exchange key material and agree on the wallet descriptor (BSMS). **Every spend:** a PSBT visits each signer and returns.
- In practice those files travel over **email and messaging apps**.
- **Content leak** — a descriptor reveals every address and balance the wallet will ever have. Forever.
- **Metadata leak** — the channel reveals who coordinates with whom, and when.
- For journalists, activists, NGOs, diaspora organizers: "these five people exchanged wallet files on Tuesday" is dangerous by itself.

01 · THREAT MODEL

The relay is the adversary

- Assume the relay **records everything**, tries to read contents, tries to **substitute handshake keys**, replays captured ciphertext. The protocol must hold anyway.
- Guarantee 1 — the relay never learns **plaintext**.
- Guarantee 2 — no key substitution without **already knowing the code**.
- Guarantee 3 — a code compromised later **does not unlock past sessions**.
- Guarantee 4 — private signing keys **never enter Keylay**. Coordination layer, not a wallet.

Out of scope, stated honestly: endpoint malware · compromised browser · insecurely shared code · identity ("knows the code," never "is Alice")

02 · ONE CODE, TWO JOBS

One secret, two opposite derivations

session code

10 chars · pruned 31-char alphabet · rejection-sampled · ~50 bits · shared out-of-band

↓ SHA-256, truncated to 32 hex

↓ PBKDF2-SHA256 · 300,000 iterations · salt 'keylay-v1-hmac'

CHANNEL NAME — PUBLIC

The WebSocket room label. Routing, nothing else.

Also the design's sharpest weakness: a fast hash of a guessable secret (W1).

HMAC KEY — SECRET

Authenticates the handshake. Slow on purpose: attacker pays 300k iterations per guess.

Fixed salt = domain separation, not a true salt — bootstrap circle (W2).

The hinge: the PBKDF2 output **authenticates, never encrypts** — confidentiality never depends on anything derived from the code.

03 · THE SEALED HELLO

Fresh keys, stamped

one message each way

```
hello = {
  pubkey : base64( my ephemeral X25519 public key ),
  sig    : HMAC( hmac-key, that base64 string )
}  // field named "sig" — an HMAC tag, not a signature
```

- **Ephemeral**: keypairs are not identities — they live minutes, then burn.
- The seal closes the classic relay MitM: a substituted key **fails verification loudly**. This one property is what the whole PBKDF2 branch was built to buy.
- The seal proves "someone who knows the code" — deliberately not *which* someone.

04 · THE SESSION KEY

Computed twice, transmitted never

on verifying the peer's hello

```
shared bits = X25519( multiply: my private scalar × their public point )
session key = HKDF( shared bits, info = sorted( pubA, pubB ) )
my private key = null   // immediately — this line is the whole point
```

- No key-delivery step exists, so there is **no key-delivery step to attack**.
- HKDF's `info` binds the key to **these two participants** — splicing fails structurally.
- Nulling the privates buys **forward secrecy**: a code leaked next month opens future sessions, never the recorded past.
- Honest bound: computational, not information-theoretic — the transcript still contains the public keys (W12).

04 · THE CHAIN

One line explains the whole design.

code → **HMAC key** → authenticates → pubkeys → **X25519 multiply** → **HKDF**(sorted pubs) → **AES-256-GCM key** → privates **nulled**

The code touches encryption only through authentication — never as key material. **That separation is forward secrecy.**

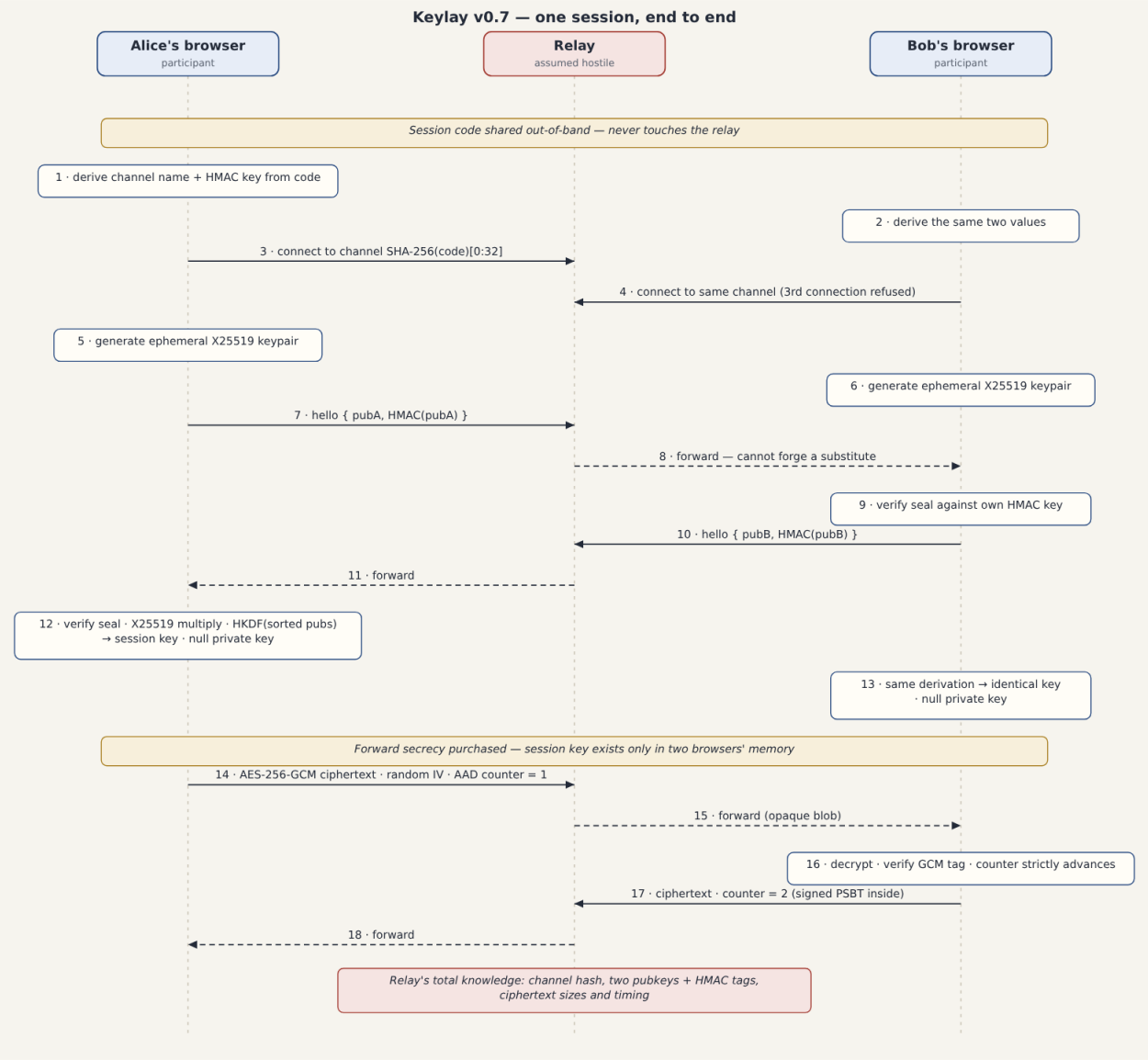
05 · PROTECTING MESSAGES

Three attacks, three named answers

- All payloads — PSBTs, BSMS files, blobs — under **AES-256-GCM**: authenticated encryption, tampering fails loudly.
- Random 12-byte **IV** per message, sent openly — identical plaintexts never repeat as ciphertext.
- **AAD** = `'keylay-v1|' + counter` — the replay counter rides inside the integrity envelope; receiver enforces strictly ascending numbers.
- **Read** → blocked by encryption. **Tamper** → blocked by the GCM tag. **Replay** → stale number dies at the door.

Those are all three moves available to a relay that doesn't know the code.

One session, end to end



07 · LIVE DEMO

The diagram you just saw, running

1

Create session

app.keylay.org — a random session code is generated in the browser.

2

Share out-of-band

The code travels by voice — out-of-band, never through the relay in raw form.

3

Join from phone

Sealed hellos exchange; two ephemeral keypairs derive one session key.

4

Send a descriptor

A BSMS file crosses encrypted; the replay counter advances.

5

Air-gap bridge

Animated QR from a hardware signer, captured frame by frame, relayed byte-identical.

6

End session

The tab closes — the session key ceases to exist anywhere.

08 · WEAKNESSES, RAISED BEFORE YOU ASK

W1 — the one that matters most

W1 The channel name is a fast-hash oracle on the code

HIGH · ANALYSIS FLAW

Crack tables assume every guess pays 300k PBKDF2 iterations. But the channel name is bare SHA-256(code) — an attacker with relay visibility tests candidates at raw hash speed. A ~50-bit code drops from "~58 years" to roughly **GPU-days**.

BOUND IT Forward secrecy holds regardless — a cracked code decrypts nothing recorded. Oracle audience: relay operator, server compromisers, any TLS-terminating front — *not* passive network observers, who see only DNS/SNI.

FIX (0.8.0) One PBKDF2 run, HKDF-expanded under two labels — both outputs behind the stretching. Residual: the name stays a deterministic fingerprint; reuse stays linkable.

08 · WEAKNESSES, RAISED BEFORE YOU ASK

W2, W3 — design debt and the scope boundary

W2 Hand-rolled PAKE with a static salt

MODERATE · DESIGN DEBT

The handshake approximates a password-authenticated key exchange — a solved problem (SPAKE2, CPace). A standard PAKE retires the salt bootstrap, a reflection edge, *and* W1's oracle in one migration.

W3 Code knowledge is membership — no identity layer

MODERATE · BY DESIGN, OWNED

"So it reduces to the security of a Signal message?" — **Yes, at session start.** Every introduction protocol bottoms out in a prior trust anchor; Keylay's is deliberately yours, not a vendor's. Backstop: hardware screens verify what an interloper cannot forge. Rule to enforce in UX: *fresh code between sittings*.

One more sentence for this room: recorded transcripts are harvest-now-decrypt-later vulnerable at the X25519 step — and descriptors are sensitive **forever** (W12; hybrid ML-KEM fix path).

09 · STANDARDS MAP

Nothing novel at either layer

STEWARD	STANDARD	IN KEYLAY V0.7
Blockchain Commons	UR family — Uniform Resources, bytewords, registry types, fountain-coded animated QR	`receive-side today` capture + reassembly + raw-UR relay; no encode, no CBOR decode yet
BIP process	PSBT (BIP-174) · BSMS (BIP-129) · descriptors (BIPs 380–386)	`payloads` transported byte-identical
Coinkite	BBQr	`full encode + decode`
IETF / NIST	X25519 · HKDF · HMAC · PBKDF2 · AES-GCM · SHA-256	`all protocol crypto` browser-native Web Crypto only

09 · ASKS

What this room can answer

- **The UR-encode gap** — reference libraries are Node-first; a single-file, auditable browser app resists a bundling step. A browser-friendly path would change what ships next.
- **Review our UR usage** — legacy `crypto-*` type names vs current registry; multipart handling — before habits harden into a de facto spec.
- **Envelope / GSTP / dCBOR** — is the Gordian stack the right substrate for the planned session-protocol message layer, rather than inventing one?

The relay is the adversary, the code never encrypts, the keys burn at handshake — and everything it carries is someone else's open standard. Exactly how it should be.