The Hesskey Backbone

Developer API — build identity, orgs, events & tickets on Hesskey

Read `/api/v1/*` · *Write* `/api/action` · *canonical base* `foundation.hesskey.org`

Reference: the Hesskey web app at /web-app/ is the first consumer · generated 2026-06-11

The two surfaces

Hesskey is an identity + object backbone you build on. Your app **reads** public state over a plain JSON API, and **writes** by composing an *intent* the user's phone approves and signs — your app never touches keys. The browser/SDK is always a viewport with a session, never a wallet.

Surface	Method	Trust	For
Read — /api/v1/*	GET	node-trusted JSON (T08) or verify-yourself (T09)	render identity, orgs, events, tickets
Write — /api/action	POST	phone-approved + phone-signed	mint / edit / grant — the phone is the authorizer

1 · Auth — session delegation (no keys in your page)

A user signs in once; the Hesskey extension (or @hesskey/web-sdk) holds a short-lived, origin-scoped Session Delegation Credential (SDC, ≤24 h, phone-signed, on-chain revocable) and mints a per-request Session Attestation Token (SAT). You pass `{sdc, sat}` as the `auth` envelope on writes. Verification is trustless: `verify_delegated()` replays the signature chain against on-chain state (T09).

- Silent sign-in: `HessKeyAuth({trust:'local'}).createSession()`.
- QR fallback: render the session QR; the phone scans + approves.
- eID card sign-in (desktop reader): `POST /api/card-sessions` → card signs behind the government PIN dialog → IS2 proof (chain-pinned BRCA + OCSP).

2 · Reads — GET /api/v1/* (clean JSON)

CORS-enabled. Org *discovery* is backed by the canonical node's request log, so query the canonical base (foundation.hesskey.org) for org/event lists; per-object reads work on any node.

Endpoint	Returns
GET /api/v1/orgs?did=<did>	{orgs:[{id, claims:{name}, my_role, …}], count} — spaces the DID owns/manages
GET /api/v1/orgs/{id}/events	{events:[{id, claims:{title,venue,start,capacity}, reentry, grants}]}
GET /api/v1/orgs/{id}/members	on-chain Owners + Managers
GET /api/v1/orgs/{id}/treasury	keyless org wallet {address, balance}
GET /api/v1/orgs/{id}/invites	pending person-membership invites
GET /api/v1/events/{id}/tickets	tickets minted under an event
GET /api/v1/holdings?key=<hex>	tickets held under an owner key

Object semantics live in the off-chain VC (`claims`); on-chain holds only what needs consensus. Held tickets are deliberately not publicly enumerable by DID. **Trustless option (T09):** read raw chain storage via `POST /rpc` and decode `pallet-object` yourself — the web app ships `chain-reads.js` as a worked example.

3 · Writes — POST /api/action (phone-approved intents)

Your app composes a structured intent; the verifier validates the session, classifies a tier, builds a canonical signing envelope and queues it for the phone. The phone renders the approval from the parsed call (never a server-supplied string), runs the presence gate, signs and finalizes. You never see a key or a signature.

```
POST /api/action
{ "did": "<32-hex>",
  "intent": { "call_index": 13,
              "args": { "object": 7, "principal": {"Person":"<32hex>"}, "cap_bits": 16 } },
  "auth":   { "sdc": "<b64>", "sat": "<hex>", "origin": "https://your-site" } }
-> 200 { "status":"queued", "tiers":[3], "envelopes":[ ...exactly what the phone signs... ] }
```

Envelope binds origin, composing-session id, nonce, expires_at, chain_state_anchor, params_hash. Pass `intents:[…]` to batch — but tier-3 actions are never batchable.

Tier	Examples	Ceremony
1 — routine	update_claims, set_reentry	optional time-boxed grant; light tap
2 — sensitive	mint, mint_batch, assign, revoke	per-action phone tap (presence high)
3 — authority	grant/revoke_capability, transfer, owner-change	single-action, eID face re-match (top), never batched

Object call indices (pallet `object`): mint 0, update_claims 1, transfer 2, revoke 3, set_reentry 11, grant_capability 13, revoke_capability 14, mint_batch 15, assign 16. Capability bits: Manage 0x01, CreateEvents 0x02, CreateTickets 0x04, EditTickets 0x08, Scan 0x10, Treasury 0x20.

4 · Build on Hesskey — quickstart

```
import { HessKeyAuth } from 'https://foundation.hesskey.org/shared/hesskey-sdk.js';
const auth = new HessKeyAuth({ verifierApiUrl:'https://foundation.hesskey.org',
  domain: location.hostname, authType:'siop_vp', trust:'local' });
await auth.createSession();                        // silent or QR
auth.onVerified(async ({ result }) => {
  const did = result.did.replace(/^did:hesskey:/, '');
  // READ
  const { orgs } = await (await fetch(
    `https://foundation.hesskey.org/api/v1/orgs?did=${did}`)).json();
  // WRITE - grant Scan to door staff (Tier 3; phone face-match + signs)
  const sat = await window.__hesskey.session.signChallenge(
    { origin: location.origin, challenge });
  await fetch('https://foundation.hesskey.org/api/action', { method:'POST',
    headers:{'Content-Type':'application/json'},
    body: JSON.stringify({ did,
      intent:{ call_index:13, args:{ object: orgs[0].id,
               principal:{Person:doorStaffDid}, cap_bits:0x10 } },
      auth:{ sdc: sat.sdc, sat: sat.sat, origin: location.origin } }) });
});
```

That's the whole backbone: sign in, read public state, propose actions the user's phone signs. No keys, no custody, no server you must trust for integrity. Full design: docs/design/web-app-building-block.md.