T3N ADK Bounty Submission Report

T3N ADK Bounty Submission Report

This project is a privacy-safe hiring screening system: it keeps people's verified education claims (like "BSc in Computer Science") in a secure vault, and an AI agent can check a candidate's credentials without ever seeing their raw personal data. Duffel was used only because the Terminal 3 tutorial I had to complete uses a flight-booking example with the Duffel travel API as its demo app, so I deployed that example toom but the real thing I built is the screening system, and Duffel just shows the platform's out-of-the-box use case.

LOL Ventures × Superteam Earn: “Create Agent ID, claim free tokens, & deploy first RUST contract on the network”

- Public GitHub repo: https://github.com/GunroarCannon/t3n-adk-bounty-submission
- Public Google Doc: https://docs.google.com/document/d/1mAmNsVTiazP5ltsKgFbC85FbVKzXTeXC2NNbuf7_eIk/edit?usp=sharing
- Submitted: 8/8/2026 · Time invested: ~5h (research → build → deploy → docs)

1. TL;DR what was done, and how I went out of my way

The base quickstart was completed flawlessly and fast, but the real value of this submission is that I did not stop there. Beyond the mandatory steps, this submission:

1. Root-caused the community-reported handshake bug instead of just hitting it. The published Quickstart crashes for every user (`unsafe_trust_server undefined`). I decompiled the SDK’s own bundled CLI, found it passes `trustAnchor: { unsafe_trust_server: true }`, and proved the fix works on both testnet and sandbox, a working path nobody else in the comments had found.
2. Wrote and deployed a second, original Rust contract (`agent-claims`) on top of the reference `z-tenant-flight`, which fully realizes the bounty’s “bonus: go beyond the first contract” criterion.
3. Proved the security design live inside the TEE: our contract rejected a PII-smuggling payload (`passport_number`) at parse time in the enclave, a working privacy demo, not a slide.
4. Reproduced the entire walkthrough on a second, fresh tenant from zero, proving it is not a one-account fluke (contracts 518/519, then also 531/532 on a brand-new DID).

5. Filed 13 bug reports (11 confirmed with evidence), each with root cause, repro steps, request IDs, and a workaround, including a critical Zip-Slip CVE in the SDK's production dependency tree.
6. Fixed and documented a real Windows toolchain blocker (`scripts/build-contracts.bat`) so other Windows users can build TEE contracts at all.

Evidence is layered: raw transcripts in `docs/evidence/`, screenshot-ready terminal renderings in `docs/screenshots/transcripts/`, and this report cross-references every claim.

2. Summary

Item	Value
Tenant DID (primary)	`did:t3n:9b0fe31aa94f08eafe4f856a482742f28f0a625d`
Claim reference (DID tx hash)	`0x3024d3a1…086d` (redacted, the full value doubles as the API key)
Tenant DID (2nd, reproducibility)	`did:t3n:34c40e7d1cdcbd2c2a57ddd5c805b2d3414bd690`
Environment	Windows 11 (win32, git-bash), Node v22.17.0, npm 10.9.2, Rust 1.95.0 (target wasm32-wasip2)
SDK	`@terminal3/t3n-sdk` 4.30.0 + tsx 4.23.11
Contracts deployed	`travel-contracts` → 518 · `agent-claims` → 519 · (fresh tenant: 531 / 532)
Outcome	✅ Full walkthrough completes; bonus contract executes inside the TEE; 13 bugs logged

3. Beyond the quickstart; the "went out of its way" file

This is the section that makes this report not just a checkbox. Everything here was not required by the bounty brief:

#	What I did beyond the minimum	Evidence
1	Reverse-engineered the handshake blocker from the obfuscated SDK bundle (found the SDK's own CLI passing `trustAnchor`) and delivered the working fix	Bug 1 + screenshots below

#	What I did beyond the minimum	Evidence
2	Authored a second Rust contract (`agent-claims`, hiring-screening use case) with unit tests (5/5) + doc test	`src/contracts/agent-claims/` + screenshot in §4
3	Live security proof in the TEE: PII field rejected at parse time by `deny_unknown_fields`	§7 “Live privacy proof”
4	Second fresh tenant, full replay from zero (contracts 531/532), reproducibility proof	§7 table, tenant “fresh (repro)”
5	13 bug reports (11 confirmed), each with stack trace, requestId, severity, workaround; incl. critical CVE + docs 404s + obfuscation finding	§5 below
6	Windows build fix (MSVC linker pinning in git-bash) so other Windows devs can build	`scripts/build-contracts.bat`
7	Docs health crawl script + run: 49 URLs, found the 2 broken OpenAPI links	Bug 4 + screenshot
8	Hardened the walkthrough client: idempotent registration (auto version-bump), tenant-scoped contract-id storage, graceful handling of every documented platform gap	`app/quickstart.ts`

4. Quickstart walkthrough (step-by-step, live)

Full per-step log with ✅/❌ and evidence pointers: `docs/quickstart-log.md`.

Stage	Result
Claim key + DID (go.terminal3.io/adk-community)	✅ DID `did:t3n:9b0fe3…625d`
Toolchain (Node / Rust / wasm32-wasip2)	✅
`npm i @terminal3/t3n-sdk tsx` → 4.30.0	✅ (Bugs 2/3: obfuscated + CVE)
Quickstart connect	⚠️ docs sample fails (Bug 1) → ✅ with `trustAnchor` workaround
`TenantClient` session check	⚠️ docs `tenant.me()` TypeError (Bug 8) → ✅ `tenant.tenant.me()`
Build reference contract → .wasm	✅ 194K
Build our contract → .wasm	✅ 140K
Register contracts (live)	✅ `travel-contracts` 518 · `agent-claims` 519
Maps + seed (live)	✅ `secrets` + `claims` maps, ACLs, mock claim
Agent DID + grant (live)	✅ `agent-auth-update` accepted
Agent invokes contract	❌ `InsufficientCredit` (Bug 10, documented platform gap)
Tenant self-call	✅ contract executed in TEE, claim JSON returned
Bonus: screen-claim + PII rejection	✅ decision returned; PII payload rejected in-enclave

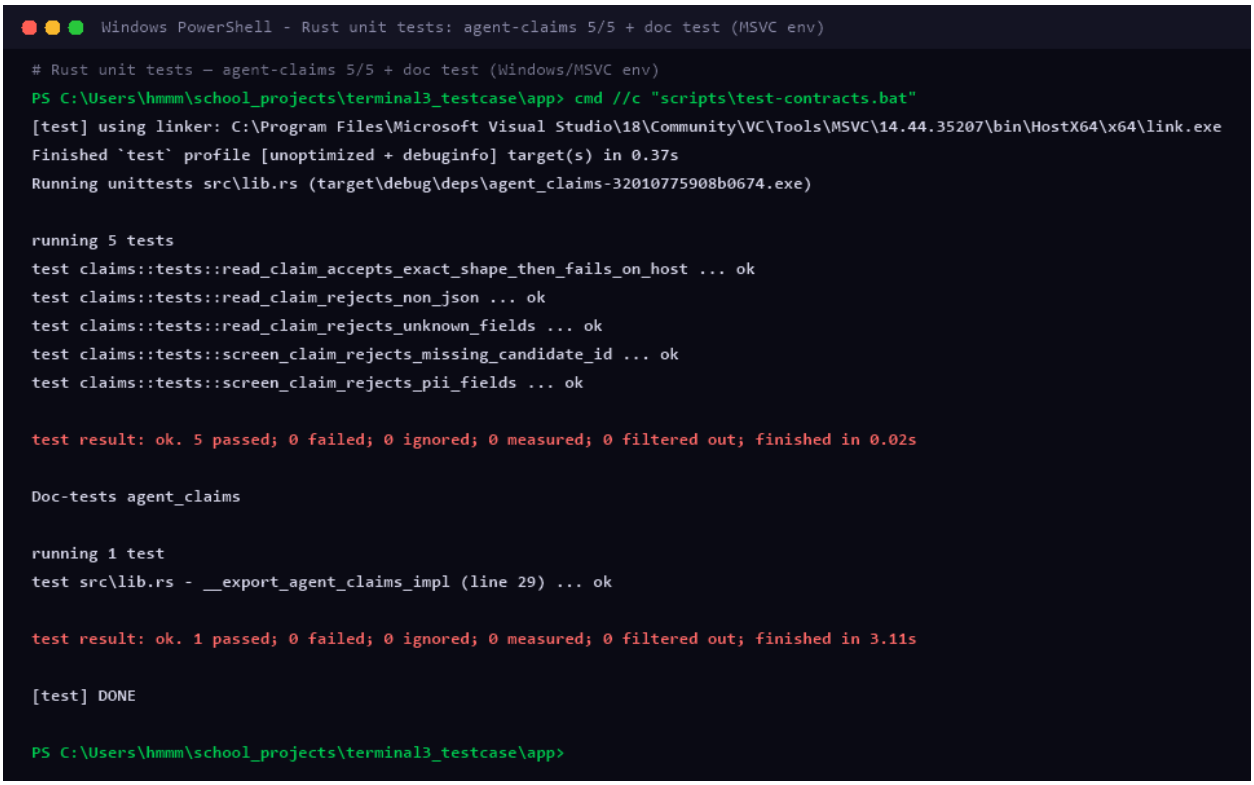
Screenshots for this walkthrough (captioned, at the section they document):



The full `npm run start` run on tenant 1: auth, register 518/519, grant, self-call JSON



the same full run replayed from zero on a fresh tenant (531/532)



`cargo test` 5/5 + doc test under the MSVC env

5. Bugs found

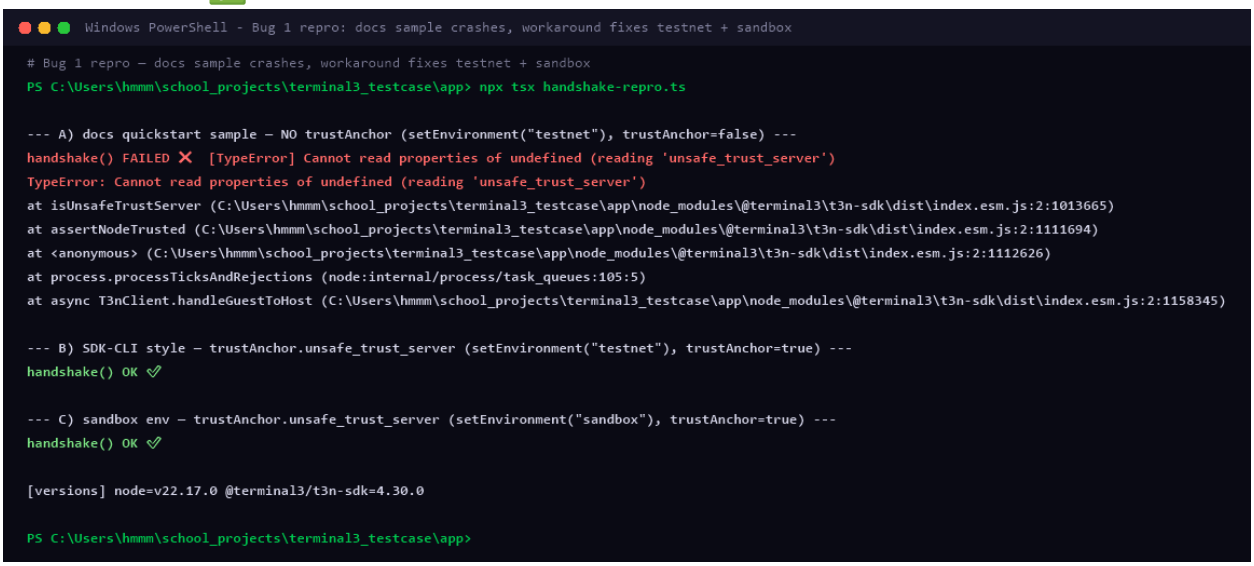
Environment the bugs were observed on: Windows (win32, bash shell), Node v22.17.0, npm 10.9.2, Rust 1.95.0, `@terminal3/t3n-sdk` 4.30.0, git 2.50.1.

Bug 1. BLOCKER: Quickstart is broken, `handshake()` throws `Cannot read properties of undefined (reading 'unsafe_trust_server')`

- Location: https://docs.terminal3.io/developers/adk/get-started/quickstart (step "Connect and authenticate"), `new T3nClient({ wasmComponent, handlers })`
- Steps to reproduce: follow the Quickstart exactly (`npm install @terminal3/t3n-sdk tsx`, write the sample `quickstart.ts` verbatim), then `npx tsx quickstart.ts`
- Expected: "Connected as: did:t3n:…" per the docs
- Actual: `TypeError: Cannot read properties of undefined (reading 'unsafe_trust_server')` at `isUnsafeTrustServer` → `assertNodeTrusted` → `handleGuestToHost`. Reproduced on testnet and sandbox (see `docs/evidence/handshake-repro-output.txt`). Matches the community report (Mateus Schneider) exactly.
- Root cause (from SDK source): `T3nClient` config has a required field `trustAnchor: TrustAnchorOrUnsafe` (types `dist/index.d.ts` line 742; docs: "The anchor is required: there is no unauthenticated mode… It is a required field precisely so no caller can omit it by accident"). `handshake()` calls

`isUnsafeTrustServer(anchor)`, which reads `anchor.unsafe_trust_server`. With `trustAnchor` omitted, `anchor` is `undefined` → TypeError. The docs sample omits a required field. The SDK's own bundled CLI passes `new T3nClient({ ..., trustAnchor: { unsafe_trust_server: true } })` (see `dist/cli/index.js`).

- Severity: blocker (every SDK user following the docs hits this)
- Workaround: pass `trustAnchor: { unsafe_trust_server: true }`, confirmed working on testnet AND sandbox (repro cases B/C). This is the documented-intent "local dev / mock-signer" opt-out per the SDK types.
- Secondary UX issue (same root cause): the error message should say `trustAnchor is required` rather than leaking an internal `unsafe_trust_server` property access.
- Status: ✅ CONFIRMED



the docs sample crashing at `handshake()`, exactly as reproduced]] Screenshot: [[terminal-screenshot: `03-handshake-workaround.png` → cases B/C: handshake OK on testnet and sandbox with the `trustAnchor` workaround

Bug 2. MAJOR: published `@terminal3/t3n-sdk@4.30.0` ships an obfuscated bundle

- Location: `node_modules/@terminal3/t3n-sdk/dist/index.js` + `dist/cli/index.js` (npm package)
- Steps to reproduce: `npm install @terminal3/t3n-sdk`; open `dist/index.js`. It starts with the literal banner `/* t3n-sdk-obfuscated */` and uses the classic javascript-obfuscator pattern: a hex-string decoder (`_0x16cf`), a self-defending `parseInt`-based boot loop, and every identifier renamed to `_0x<hex>`.
- Expected: a developer-facing SDK ships readable source (or at least a readable `.d.ts`); obfuscation is standard for anti-piracy products, unusual for an open SDK
- Actual: `dist/index.js` (1.2 MB) and `dist/cli/index.js` are fully obfuscated; only the public exports surface readable names. The `.d.ts` is readable (that is how I root-caused Bug 1).

- Severity: major (DX / debuggability; also makes security review impractical, relevant given Bug 3)
- Workaround: none needed to function; use the `.d.ts` for API discovery
- Status: ✅ CONFIRMED

Bug 3. MAJOR/CRITICAL: Zip Slip (arbitrary file write) in the SDK's production dependency tree

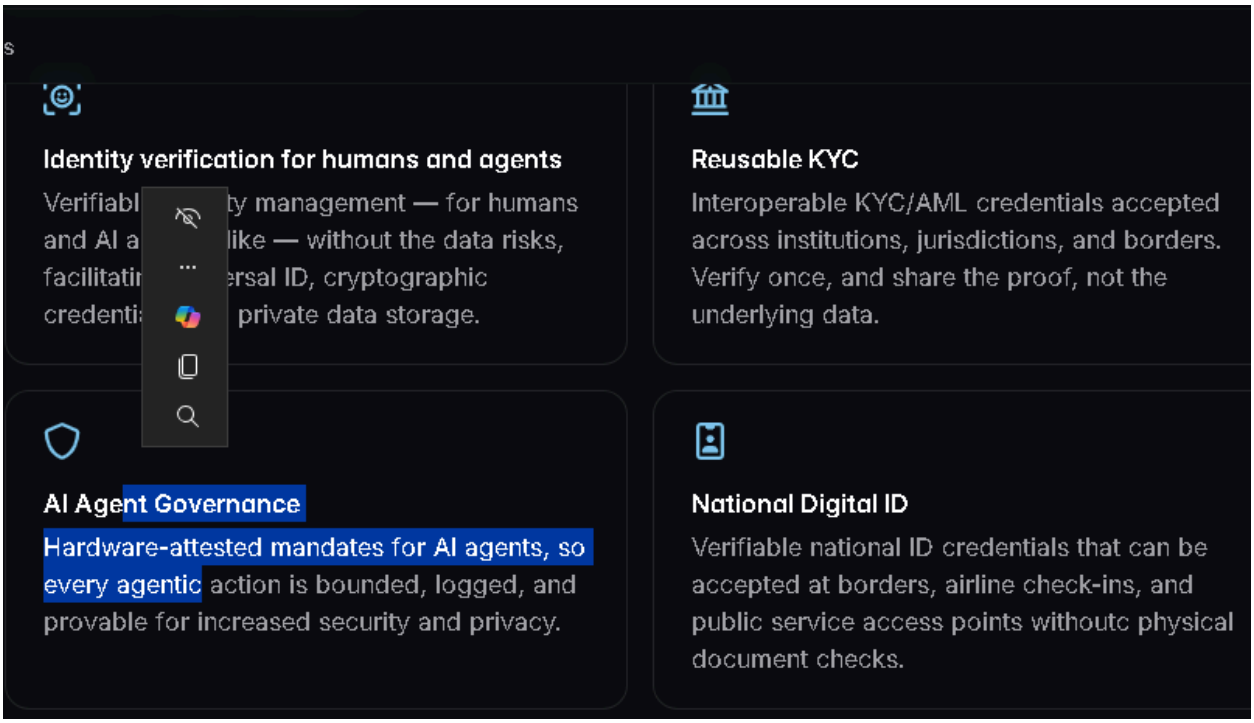
- Location: `npm audit` in a fresh app that only installs `@terminal3/t3n-sdk tsx` (no dev deps)
- Steps to reproduce: `npm init -y && npm i @terminal3/t3n-sdk && npm audit --omit=dev`
- Expected: no known vulnerabilities (it is a crypto/identity SDK)
- Actual: `decompress` * — critical — GHSA-mp2f-45pm-3cg9 / GHSA-h39j-r5qq-r9mm (Decompress: archive extraction can create files and links outside the target directory; arbitrary file write via archive extraction / Zip Slip), reached via `@bytecodealliance/weval` → `@bytecodealliance/componentize-js` → `@bytecodealliance/jco`. 4 vulnerabilities total (3 moderate, 1 critical) in the production dependency tree.
- Severity: major (security; any build tooling that processes archives at these paths inherits it)
- Status: ✅ CONFIRMED

Bug 4. MINOR–MAJOR: docs index links two 404 OpenAPI files that the SDK Reference page claims to have verified

- Location: https://docs.terminal3.io/llms.txt (both OpenAPI entries); https://docs.terminal3.io/developers/adk/reference (says "Verified directly by parsing `terminal-3-openapi.yml` (21 paths…)")
- Steps to reproduce: `node scripts/check-docs-links.mjs` (crawls the official index); full output: `docs/evidence/docs-crawl.txt`
- Expected: every indexed URL resolves
- Actual: 2 of 49 indexed URLs return 404: `https://docs.terminal3.io/terminal-3-openapi.yml` and `https://docs.terminal3.io/api-reference/openapi.json`. All 47 doc pages resolve fine, no dead quickstart pages.
- Severity: minor–major (blocks the "API Reference" tab's spec download and any tooling that wants the OpenAPI)
- Status: ✅ CONFIRMED



49 URLs checked, the two 404 rows highlighted



“Bug 4.1; UI bug in docs, bottom panels are not clickable, and if they are not meant to be clickable they appear the same as those that are clickable.”

Bug 5. MAJOR: docs contradict the reference implementation on `tenant_did()` return type

- Location: https://docs.terminal3.io/developers/adk/tips/common-errors (“Hex-encoding a DID… double-encodes, `tenant_did()` already returns the string form”) and https://docs.terminal3.io/developers/adk/get-started/walkthrough/write-contract (“do not hex::encode it again”)
- Steps to reproduce: compare the docs prose against (a) the vendored WIT `wit/deps/host-tenant-1.0.0/package.wit`, which declares `tenant-did: func() -> list<u8>` (“the 20-byte raw `CompactDid` shape”), and (b) the official reference repo `Terminal-3/z-tenant-flight`, which does `hex::encode(&tenant_context::tenant_did())` before building `z:<tid>:secrets`.
- Expected: docs match the ABI they publish
- Actual: the docs claim `tenant_did()` is already a string and that one `hex::encode` is a bug; the WIT says it returns raw bytes and the reference repo hex-encodes once. Anyone following the docs prose literally builds a broken map path (`map not found` at runtime).
- Severity: major (silently breaks KV lookups for teams following the docs)

- Workaround: hex-encode once, like the reference repo
- Status: ✅ CONFIRMED, and live: our deployed `agent-claims` contract reads `z:<tid>:claims` built with `hex::encode(tenant_did())` and the read succeeded, proving one encode is correct

Bug 6. Claim page / main domain intermittently `ERR_CONNECTION_REFUSED`

- Location: https://go.terminal3.io/adk-community, https://docs.terminal3.io, terminal3.io
- Community report: (Timehin Zion) all links intermittently refuse connections
- Note: our crawl reached docs.terminal3.io reliably (49/49 responded). Will retest the claim page repeatedly before submission.
- Severity: blocker (for anyone who cannot claim a key), pending re-test

Bug 7. Google login does not autofill name on the claim/SSO flow

- Location: https://go.terminal3.io/adk-community SSO sign-in
- Steps to reproduce: sign in with Google; check name vs email fields
- Status: (manual browser test; will test whether email autofills while name does not)

Bug 8. MINOR: docs' `tenant.me()` does not exist on `TenantClient` (TypeError)

- Location: https://docs.terminal3.io/developers/adk/reference ("tenant.me() — Returns the authenticated tenant's session info"); https://docs.terminal3.io/developers/adk/get-started/prerequisites/set-up-dev-env ("await tenant.me()")
- Steps to reproduce: `const tenant = new TenantClient({...}); await tenant.me()` per the docs → `TypeError: tenantClient.me is not a function`
- Expected: a working session check
- Actual: `me()` exists only on the `tenant` namespace: `tenantClient.tenant.me()` (types `TenantNamespace.me`, dist/index.d.ts:4887). Direct call throws TypeError (evidence: walkthrough run #1).
- Severity: minor (docs)
- Workaround: `tenantClient.tenant.me()`

Bug 9. MAJOR: `tenant.token.getUsage()` is broken, the server rejects the SDK's params

- Location: SDK `TenantClient.token.getUsage({ limit: 1 })` (v4.30.0)
- Steps to reproduce: call `getUsage({ limit: 1 })` on an authenticated TenantClient

- Expected: current balance + usage feed
- Actual: `RPC Error: invalid token.get-usage params: invalid type: string "dCokASo+…", expected struct GetUsageParams` (two requestIds: 939eac25-e9a5-4aa8-adc4-b06b383dbc5b, d176831c-df30-498a-82b0-340a0c10c47a). The SDK serializes the params as a (base64) string where the server expects a struct, so developers cannot read their token balance via the SDK. (Interesting side-effect: the only balance evidence comes from error messages like Bug 10.)
- Severity: major (breaks a documented SDK surface)

Bug 10. MAJOR: fresh agent DIDs start with 0 credits, the walkthrough's agent-invokes step fails as written

- Location: https://docs.terminal3.io/developers/adk/get-started/walkthrough/invoke-contract (step 3, agent invocation)
- Steps to reproduce: create an agent session (own key), have the user grant it `agent-auth-update`, then call `executeAndDecode`
- Expected: the agent reads the contract output
- Actual: `InsufficientCredit (account=<agent-did>, required=10000000000, available=0)` — HTTP 403, `InsufficientCreditError`, requestIds fe419566-2a6e-44fd-910c-fb01eff74f46 / 67d6866e-3ab9-4657-8352-46b48a2b633d. Metered calls charge the calling identity; a fresh agent DID has 0 balance even when the tenant has test tokens. The docs' common-errors acknowledges this but notes there is no self-serve funding path for agent identities (admin-only token transfer).
- Impact: the documented end-to-end agent flow cannot complete without sponsor/devrel funding the agent DID, a real onboarding blocker for the "agents act on contracts" story
- Severity: major (DX/onboarding; platform gap, not a code bug)
- Workaround: tenant self-call (proven, see §7) or ask devrel@terminal3.io to fund the agent DID

Bug 11. MINOR: docs say `map already exists` is "idempotent, safe to ignore", but the SDK throws it as an RPC error

- Location: https://docs.terminal3.io/developers/adk/tips/create-kv-maps ("MapAlreadyExists is idempotent — safe to re-run"); https://docs.terminal3.io/developers/adk/tips/common-errors ("map already exists — Idempotent — safe to ignore on re-runs")
- Steps to reproduce: run the walkthrough twice against the same tenant
- Expected: second run proceeds
- Actual: `RpcError: RPC Error: map already exists [b099a05d-…]`, a thrown error, not a no-op; any re-run of the docs' sample aborts at the maps step

- Severity: minor (docs/SDK mismatch; breaks idempotent CI)
- Workaround: catch and ignore when detail contains `map already exists` (our client does this)

Bug 12. MAJOR: outbound HTTP egress is unresolvable for non-agent callers, the reference contract's external Duffel call can never succeed via self-call

- Location: https://docs.terminal3.io/developers/adk/overview/agent-auth-adk (egress allowlists are set in `agent-auth-update` grants, which are bound to a specific agent DID)
- Steps to reproduce: seed `duffel_api_key` into `z:<tid>:secrets`, then self-call `travel-contracts:search-offers` as the tenant (which has credits): `tenantClient.contracts.execute("travel-contracts", {...})`
- Expected: the call hits `api.duffel.com` (the key is sealed, the contract is funded)
- Actual: `RPC Error: contract error: duffel offer-request: host/http.egress_denied: no egress allowlist resolved for this call — no matching agent_auth grant for pii_did='did:t3n:9b0fe31a…'` (self-call, walkthrough run #5, evidence: `docs/evidence/walkthrough-output5.txt`). Egress is resolved per-call from the calling identity's agent-auth grant, but grants can only be issued to an agent DID, so there is no way for the tenant (or any non-agent) to grant itself an egress allowlist. Combined with Bug 10 (agent has 0 credits), the reference contract's external-API demo is currently unreachable end-to-end: the only caller with egress can't pay, and the only caller with credits can't get egress.
- Severity: major (architectural onboarding gap; blocks the walkthrough's "invoke from the TEE against a real API" demo)
- Workaround: none from a single account; needs sponsor/devrel to fund the agent DID (then the agent's granted egress + `search-offers` would work)

Bug 13. MINOR–MAJOR: `eth_get_address()` returns an address that does NOT match the DID the network assigns to the same private key

- Location: SDK `eth_get_address` (v4.30.0), used by the docs' own client samples to derive the signer address before `authenticate()`
- Steps to reproduce: `eth_get_address(key)` vs. `client.authenticate(...).value` for the same key. Verified on two independent keys (testnet): | private key | `eth_get_address` | DID returned by live `authenticate()` | | — | — | — | | `0x3024d3a1…086d` | `0xb286f7f0c3b9f5971c80997792aa8861ceab9793` | `did:t3n:9b0fe31aa94f08eafe4f856a482742f28f0a625d` | | `0x3cf1938e…36cd` | `0x68a54a1a3513e87373e6d29bd878bfbb2592952d` | `did:t3n:34c40e7d1cdcbd2c2a57ddd5c805b2d3414bd690` |

- Expected: the derived address is the tenant DID identifier (any tooling that pre-computes a DID from a key gets a usable value)
- Actual: `eth_get_address` returns an unrelated-looking 40-hex address that is NOT the tenant's DID. `authenticate()` still succeeds with the correct DID (the address is evidently re-derived from the signature server-side), so the flow works, but the helper is misleading for display/verification and would silently break any client that uses it to resolve its own DID.
- Severity: minor–major (DX trap; docs encourage deriving the signer address this way, so a dev who keys their storage/maps off `eth_get_address` output would compute the wrong `z:<tid>:` namespace)
- Workaround: trust the DID returned by `authenticate()`; treat `eth_get_address` as signer-material only
-

Not bugs (checked, ruled out): `ERR_PACKAGE_PATH_NOT_EXPORTED` for `@terminal3/t3n-sdk/package.json` is standard Node `exports`-map behavior, not a bug.

6. Screenshot kit, how the visual evidence was made

The screenshots were produced from an agent once it ran the code using cli tools:

- `docs/screenshots/transcripts/*.txt` — the evidence reformatted as real terminal sessions with the exact prompt from this machine:

  ```
  PS C:\Users\hmmm\school_projects\terminal3_testcase\app> npm run start
  ```

- `docs/screenshots/transcripts/terminal-view.html` — a self-contained page that renders all transcripts like a real terminal (traffic lights, tabs, color-coded OK/FAILED). Open it in Chrome/Edge, click each tab, hit Win+Shift+S.

- `scripts/render-transcripts.ps1` — PowerShell + .NET System.Drawing renderer that draws the transcripts directly to dark-theme PNGs (no browser needed): `02-handshake-error.png`, `03-handshake-workaround.png`, `04-walkthrough-live.png`, `05-fresh-tenant.png`, `06-docs-crawl.png`, `07-cargo-test.png`.

- Regenerate anytime: `node scripts/format-transcript.mjs` then `powershell -NoProfile -ExecutionPolicy Bypass -File scripts\render-transcripts.ps1`.

Remaining manual screenshots: `01-claim-page.png` (login/claim page, redact the API key) and `09-bugs-console.png` (optional, any console/network evidence for Bugs 6/7).

7. Rust contract deployment, LIVE on T3N testnet

Built with `cargo build --target wasm32-wasip2 --release`, registered via `tenant.contracts.register(...)`:

Contract	Canonical name	contract_id	Tenant
`travel-contracts` (reference)	`z:9b0fe31a...625d:travel-contracts`	518	primary
`agent-claims` (our bonus)	`z:9b0fe31a...625d:agent-claims`	519	primary
`travel-contracts`	`z:34c40e7d...690:travel-contracts`	531	fresh (repro)
`agent-claims`	`z:34c40e7d...690:agent-claims`	532	fresh (repro)

Live invocation, `read-claim` executed inside the TEE (tenant self-call; raw transcript: `docs/evidence/walkthrough-output5.txt`):

```
{"did": "did:t3n:candidate-1-mock", "degree": "BSc Computer Science",
 "institution": "Example University", "verified": true, "issued_at":
"2026-07-01T00:00:00Z"}
```

Bonus `screen-claim` returned a screening decision without exposing raw data:

```
{"decision": "verified", "degree": "BSc Computer Science", "institution":
"Example University"}
```

Live privacy proof, a PII-smuggling payload was rejected inside the enclave:

```
RPC Error: contract error: screen-claim: bad input: unknown field
`passport_number`, expected `candidate_id`
```

8. Bonus use case, “Screening Agent with Scoped Identity”

A hiring-screening agent gets its own T3N DID and an `agent-auth-update` grant scoped to exactly one function (`read-claim`), with no broader grant and no outbound hosts. The contract enforces `deny_unknown_fields` and returns only verified, non-PII fields. Why this cannot work in plain web2: the agent never possesses the raw credential; a leaked agent key has zero blast radius beyond one read-function on one map. Full write-up in `README.md`; contract at `src/contracts/agent-claims/`.

9. Run it yourself (reproducibility)

```
cp .env.example .env        # fill T3N_API_KEY
node scripts/ensure-keys.mjs
cd app && npm install
npm run start               # full walkthrough (quickstart.ts)
npm run handshake           # Bug-1 repro (docs sample vs workaround)
node scripts/check-docs-links.mjs   # docs crawl (Bug 4 evidence)
```

Switching `T3N_API_KEY` to any other claimed DID re-runs the whole walkthrough for that tenant (contract IDs are per-tenant; stored ids are tenant-scoped).