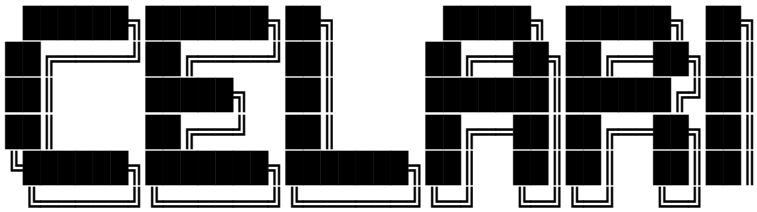
# CELARI WALLET

### Technical Report & Architecture Overview

### Aztec Network Ecosystem — February 2026

---



celāre (Latin) — to hide, to conceal

---

## Executive Summary

Celari is the **first WebAuthn/Passkey-native smart wallet on Aztec Network**. It replaces the entire seed phrase paradigm with biometric authentication — Face ID, fingerprint, Windows Hello — while leveraging Aztec's zero-knowledge architecture for complete transaction privacy.

**Core thesis:** The next billion crypto users will never write down 24 words. They'll tap their finger.

| Metric | Value |
|---|---|
| Network | Aztec v3 (Testnet deployed) |
| Signature scheme | ECDSA secp256r1 (P256) via Noir |
| Auth mechanism | WebAuthn/Passkey (hardware secure enclave) |
| Privacy model | Full — encrypted UTXOs, zero on-chain metadata |
| Extension | Chrome MV3, production-ready |
| Test coverage | 95 tests across 7 test suites |
| Codebase | 5,500+ lines across 42 files |
| Languages | Noir, TypeScript, JavaScript, CSS |

---

## 1. The Problem We Solve

### 1.1 Seed Phrases Are a UX Catastrophe

- **$100B+** lost to forgotten/stolen seed phrases (Chainalysis estimates)
- 71% of non-crypto users cite "too complex" as their #1 barrier (a16z 2025 survey)
- Every seed phrase is a phishing vector — one screenshot, one clipboard read, game over

### 1.2 Privacy Is Not Optional

- On transparent chains (Ethereum, Solana), every transfer is public
- Your employer sees your salary. Your ex sees your spending. Advertisers see everything
- Privacy is not about hiding — it's about basic financial dignity

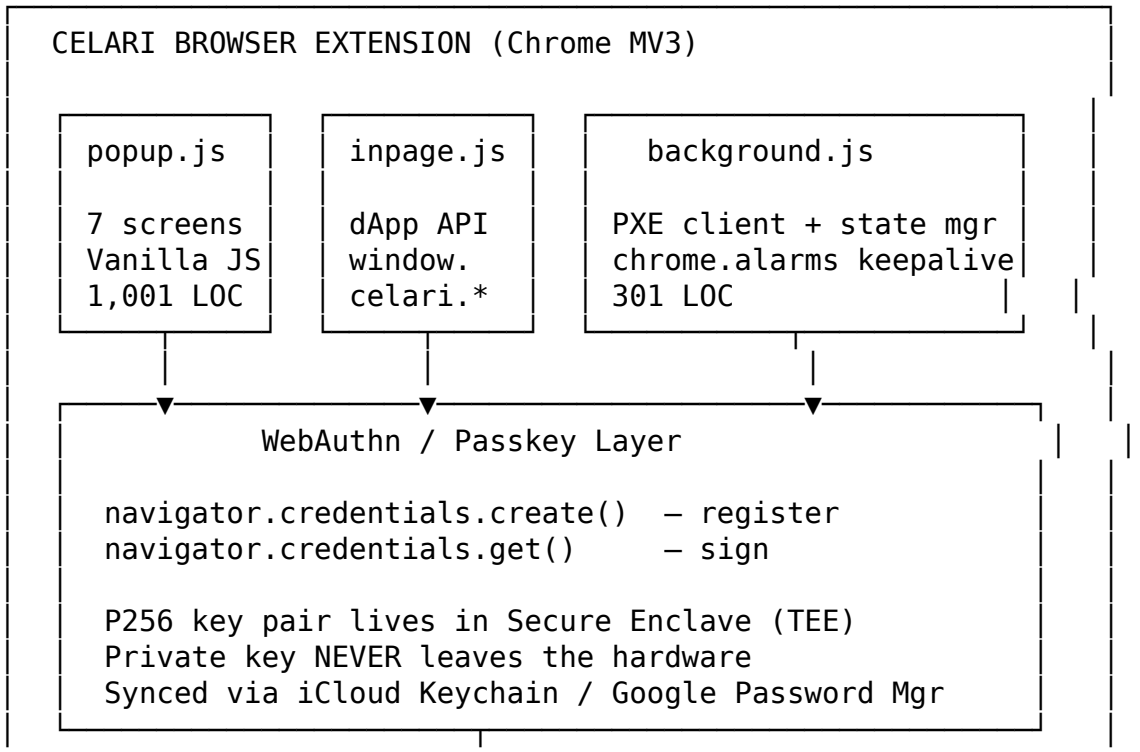
### 1.3 Our Answer

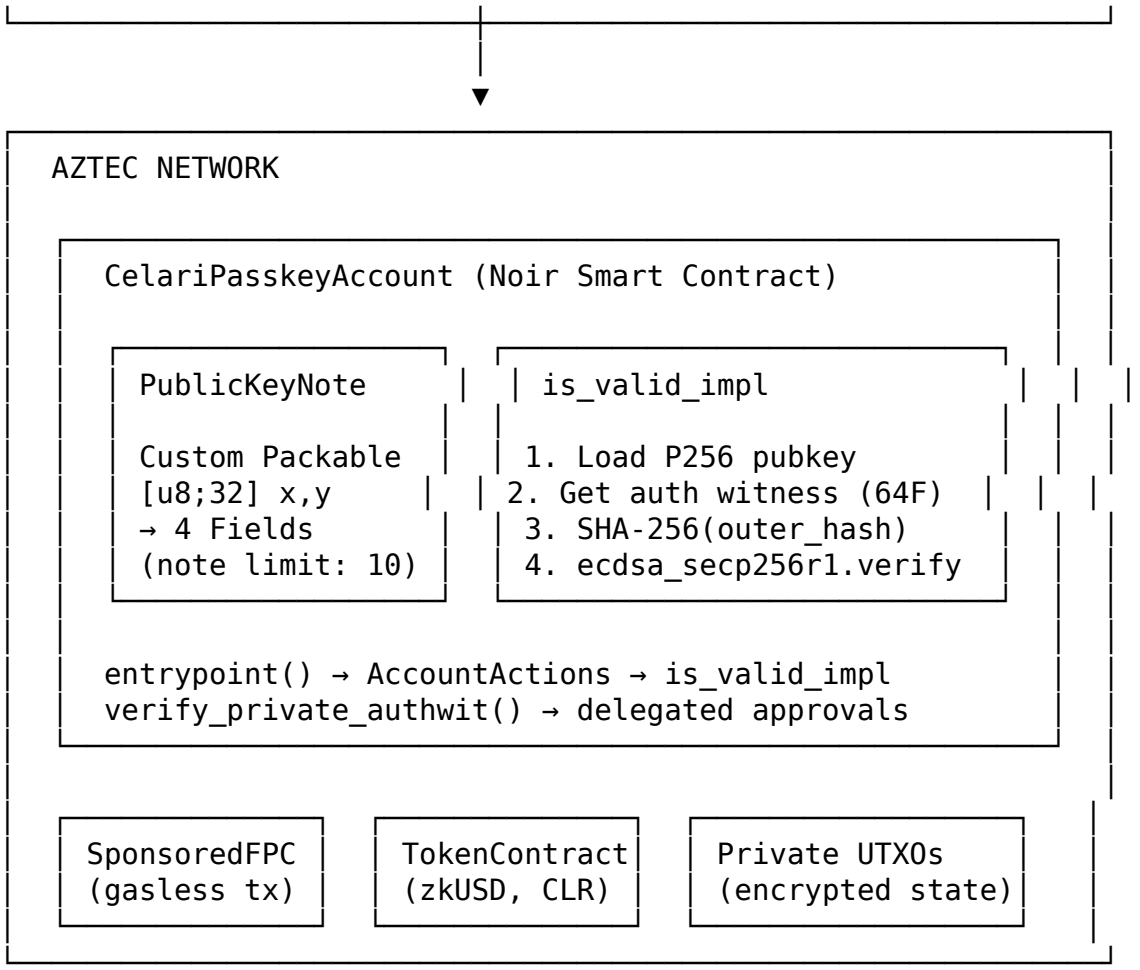
| Traditional Wallet | | Celari Wallet |
|---|---|---|
| 24-word seed phrase | → | Face ID / Fingerprint |
| Paper backup | → | iCloud / Google auto-sync |
| Software keys | → | Hardware secure enclave |
| Pseudonymous | → | Fully private (ZK proofs) |
| Phishable | → | Domain-bound, unphishable |

---

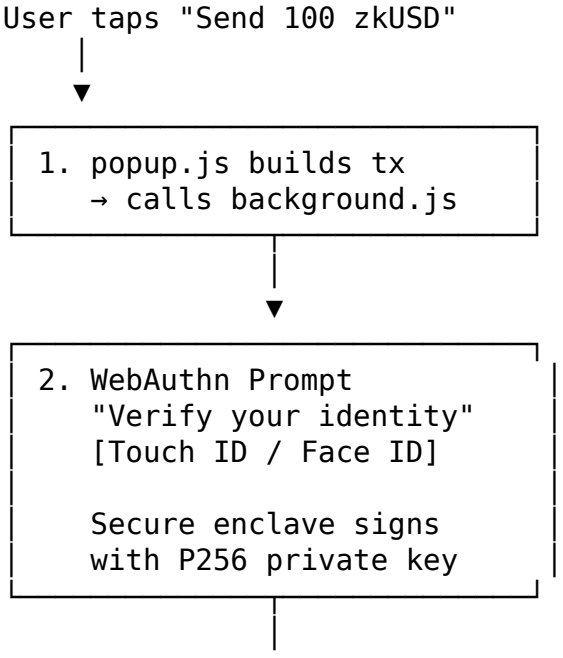
## 2. Architecture

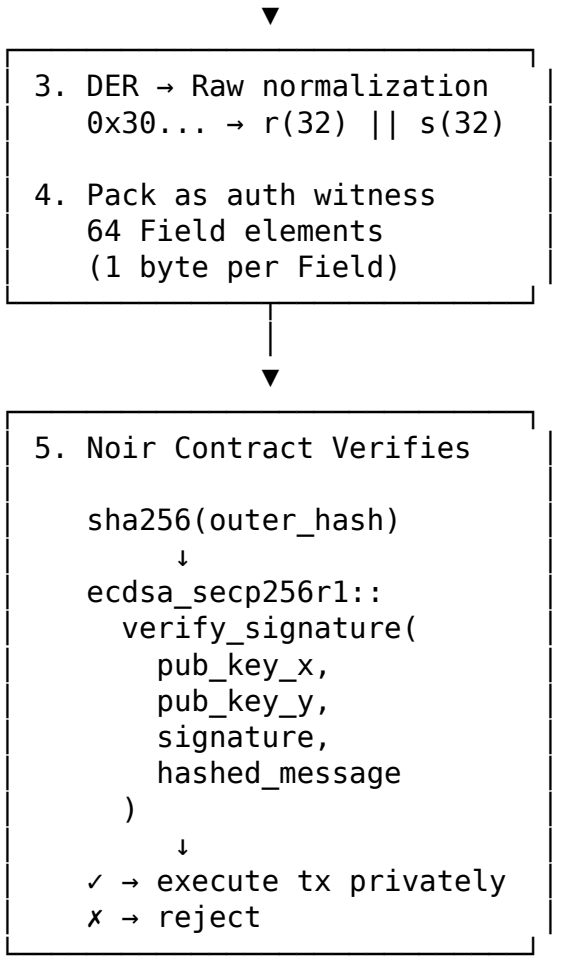
### 2.1 System Overview






### 2.2 Signature Verification Flow






---

## 3. Smart Contract Deep Dive

### 3.1 CelariPasskeyAccount (132 LOC, Noir)

The account contract is deliberately minimal — a clean implementation of P256 verification for Aztec's account abstraction framework.

**Key design decisions:**

| Decision | Rationale |
|---|---|
| `[u8; 32]` for key storage | P256 coordinates can exceed BN254 field modulus (~254 bits). Raw bytes prevent overflow |
| Custom `Packable` (4 Fields) | Aztec notes are limited to 10 Fields. Our encoding uses only 4 — room for future metadata |

| Decision | Rationale |
|---|---|
| `sha256(outer_hash)` before verify | Matches WebCrypto's `sign({hash: "SHA-256"})` — the browser and contract agree on the hash |
| 64-Field auth witness | Minimal layout: exactly 64 bytes of signature (r‖s), no padding waste |

### 3.2 PublicKeyNote — Custom Packable (61 LOC, Noir)

```
PublicKeyNote { x: [u8; 32], y: [u8; 32] }
         │
         ▼ pack()
[Field₀: x[0..31] as base-256]  ← 31 bytes fit in ~248 bits < 254-bit field
[Field₁: x[31] as single byte]  ← avoids 256^31 overflow
[Field₂: y[0..31] as base-256]
[Field₃: y[31] as single byte]
         │
         ▼ unpack()
PublicKeyNote { x: [u8; 32], y: [u8; 32] }
```

This encoding is lossless and uses only **4 of 10** available note Fields.

---

## 4. SDK & TypeScript Layer

### 4.1 Passkey SDK (`src/utils/passkey.ts` — 340 LOC)

Complete WebAuthn lifecycle management:

| Function | Purpose |
|---|---|
| `createPasskeyCredential()` | Register new passkey → P256 key in secure enclave |
| `signWithPasskey()` | Biometric auth → P256 signature |
| `normalizeP256Signature()` | DER → raw 64-byte format for Noir |
| `padTo32Bytes()` | Handle DER's variable-length integers |
| `bytesToHex()` / `hexToBytes()` | Encoding utilities |

### 4.2 Account Contract Bridge (`src/utils/passkey_account.ts` — 174 LOC)

| Class | Role |
|---|---|
| `CelariPasskeyAccountContract` | Extends `DefaultAccountContract` for Aztec's account abstraction |
| `PasskeyAuthWitnessProvider` | Browser context — triggers WebAuthn for each tx |

| Class | Role |
|---|---|
| `CliP256AuthWitnessProvider` | CLI context — uses PKCS8 private key directly |

### 4.3 Deploy Infrastructure

| Script | Purpose |
|---|---|
| `deploy_passkey_account.ts` | CLI-based account deployment |
| `deploy-server.ts` | HTTP API for one-click deploy from extension |
| `deploy-token.ts` | Token contract deployment |
| `mint-token.ts` | Token minting to deployed accounts |
| `lib/aztec-helpers.ts` | Shared: `setupSponsoredFPC()`, `generateP256KeyPair()` |

---

## 5. Browser Extension

### 5.1 Chrome MV3 Architecture

| Component | File | LOC | Role |
|---|---|---|---|
| Popup UI | `popup.js` | 1,001 | 7-screen wallet interface (vanilla JS) |
| Service Worker | `background.js` | 301 | PXE connection, state mgmt, `chrome.alarms` keepalive |
| Content Script | `content.js` | 57 | Page ↔ background bridge with type whitelist |
| Inpage Provider | `inpage.js` | 108 | `window.celari` dApp API |

### 5.2 dApp Integration API

```
// Any website can interact with Celari
const { address } = await window.celari.connect();
await window.celari.sendTransaction({ to, amount, token });
await window.celari.createAuthWit(messageHash);
const connected = await window.celari.isConnected();
window.celari.on("accountChanged", callback);
```

### 5.3 Security Measures

| Threat | Mitigation |
|---|---|
| Cross-origin messaging | `postMessage` uses `window.location.origin`, never `"*"` |
| Rogue content scripts | `sender.id === chrome.runtime.id` check in background |
| dApp message spoofing | Type whitelist: only `DAPP_CONNECT`, `DAPP_SIGN`, `GET_ADDRESS`, etc. |
| CORS on deploy server | Origin-checked whitelist (chrome-extension://, localhost) |
| Private key file leaks | `chmod 0600` + `.gitignore` with security warnings |
| MV3 service worker sleep | `chrome.alarms` instead of unreliable `setInterval` |

---

## 6. Test Suite

### 6.1 Overview

| Suite | Tests | Coverage Area |
|---|---|---|
| `passkey.test.ts` | 20 | bytesToHex, hexToBytes, DER normalization, padTo32 |
| `p256-crypto.test.ts` | 15 | Key generation, ECDSA sign/verify, auth witness packing |
| `cors.test.ts` | 18 | Origin validation, env var parsing, pattern matching |
| `extension.test.ts` | 18 | Sender validation, state management, account CRUD |
| `deploy-server.test.ts` | 14 | CORS headers, HTTP routing, error responses |
| `passkey_account.test.ts` | 8 | E2E: account creation, mint, transfer, payroll flow |
| `test.nr` (Noir TXE) | 2 | Contract deployment, public key storage |
| **Total** | **95** | |

### 6.2 Key Test Highlights

**P256 Crypto End-to-End Pipeline:**

`Generate P256 key → Sign message → Normalize DER → Pack witness → Verify signature`

This tests the complete path from key generation to what the Noir contract receives.

**CORS Security Matrix:**

`✓ chrome-extension://abcdef... → Allowed`

```
✓ http://localhost:3000        → Allowed
✗ https://evil.com             → Blocked
✗ http://localhost.evil.com    → Blocked
✗ data:text/html,...           → Blocked
```

### 6.3 CI/CD Pipeline

```
# .github/workflows/ci.yml
Steps: Install → TypeCheck → Lint → Format → Build Extension
Node: 22 | Cache: yarn | Trigger: push/PR to main
```

---

## 7. Why Aztec?

### 7.1 Technical Alignment

| Aztec Feature | How Celari Uses It |
|---|---|
| Account Abstraction | Custom account contract with P256 verification |
| Private State (UTXOs) | All balances encrypted — only the owner can decrypt |
| Noir Circuits | `ecdsa_secp256r1::verify_signature` — native P256 support |
| Auth Witnesses | Clean interface for passing signatures to contracts |
| SponsoredFPC | Gasless transactions for onboarding |
| TestWallet | Deploy server uses TestWallet for one-click deployment |

### 7.2 Why P256 Over secp256k1?

| | secp256k1 (Ethereum) | P256/secp256r1 (Celari) |
|---|---|---|
| Hardware support | None (software only) | Apple Secure Enclave, Android TEE, TPM 2.0 |
| WebAuthn compatible | No | Yes (FIDO2 standard) |
| Key extraction | Possible (MetaMask exports) | Impossible (hardware-bound) |

| secp256k1 (Ethereum) | P256/secp256r1 (Celari) |
|---|---|
| Cross-device sync | Manual seed backup | Automatic (iCloud/Google) |
| Phishing re-sistance | None | Domain-bound (origin checked) |
| User ex-pe-ri-ence | 24-word onboarding | One tap |

---

## 8. Project Metrics

### 8.1 Codebase

| Language | Files | Lines | Purpose |
|---|---|---|---|
| Noir | 3 | 264 | Smart contract + notes + tests |
| TypeScript | 14 | 3,130 | SDK + deploy scripts + unit tests |
| JavaScript | 5 | 1,525 | Extension (popup, background, content, inpage, build) |
| CSS | 1 | 696 | Extension styling |
| HTML | 2 | 265 | Popup + dApp example |
| Config | 8 | 160 | tsconfig, jest, eslint, prettier, CI |
| Markdown | 2 | 385 | README + review plan |
| Total | 35 | 6,425 | |

### 8.2 Engineering Quality

| Metric | Value |
|---|---|
| Test count | 95 (85 unit + 8 E2E + 2 Noir TXE) |
| Test suites | 7 |
| CI/CD | GitHub Actions (typecheck → lint → format → build) |
| Linting | ESLint + Prettier (enforced in CI) |
| Build system | esbuild (extension), aztec compile (contracts) |
| Security fixes applied | 27 (across 3 review passes) |
| Dead code removed | 1,394 lines |
| Code deduplication | Shared `aztec-helpers.ts` across 4 scripts |

### 8.3 Commit History

```
39b85a4  Celari Wallet Faz 1: Passkey auth + Aztec testnet deploy
c553579  Add comprehensive code review plan with 17 identified fixes
ff991ae  Apply 17 code review fixes (security, correctness, quality)
9fce5ad  Fix 10 issues from second code review pass
257e696  Add Phase D: ESLint, unit tests, esbuild, CI/CD pipeline
724a3d7  Add 65 new unit tests across 4 test suites (85 total)
```

---

## 9. Network Compatibility

| Network | Status | URL |
|---|---|---|
| Local Sandbox | Tested | `localhost:8080` |
| Aztec Devnet | Supported | `devnet-6.aztec-labs.com` |
| Aztec Testnet | **Deployed** | `rpc.testnet.aztec-labs.com` |

The extension supports one-click network switching. Accounts can be deployed via: 1. **One-click** — Deploy server (`yarn deploy:server`) handles everything 2. **CLI** — `yarn deploy:passkey` with auto-generated or provided keys 3. **JSON import** — Drop a `.celari-passkey-account.json` file

---

## 10. Roadmap

```
  COMPLETED                          IN PROGRESS                PLANNED
  ═══════════                        ═══════════                ═══════

  Phase 0                              Phase 2                  Phase 3
  Schnorr Prototype □                L1 ↔ L2 Bridge           Batch Payroll
  - Basic account                    - Portal integration     - batch_pay()
  - Token transfers                   - ETH deposit/withdraw   - CSV upload
                                 - Fee abstraction       - Employer dashboard
  Phase 1
  Passkey + Extension □              Phase 4                  Phase 5
  - P256 Noir contract             Crypto Debit Card       Mobile + Cross-
chain
  - Chrome MV3 extension         - Virtual card issuing   - React Native app
  - WebAuthn integration          - POS integration      - WalletConnect v2
   - Testnet deployment                 - Spending limits         - Multi-
chain bridge
   - 95 tests + CI/CD
   - Deploy server API
```

---

## 11. Technical Differentiators

### vs. MetaMask

- **No seed phrases** — hardware-bound keys, unextractable
- **Private by default** — Aztec's encrypted UTXOs vs. Ethereum's transparent state
- **Phishing resistant** — WebAuthn credentials are domain-bound

### vs. Argent (StarkNet)

- **True privacy** — Aztec offers protocol-level privacy; StarkNet is transparent
- **Standard crypto** — P256 is NIST-standardized, FIDO2-approved, hardware-supported worldwide
- **Simpler key management** — No guardian setup, no social recovery complexity

### vs. Other Aztec Wallets

- **First passkey-native** — All other Aztec wallets use Schnorr (secp256k1)
- **Custom Packable** — Efficient P256 key storage in 4 Fields (not 10)
- **Full SDK** — TypeScript SDK + deploy server + dApp integration API

---

## 12. For the Aztec Team

### What We Built On Your Stack

| Aztec Component | How We Use It |
|---|---|
| `ecdsa_secp256r1::verify_signature` | Core of our account contract — P256 verification in Noir |
| `AccountActions` + `DefaultAccountContract` | Clean account abstraction integration |
| `SponsoredFPC` | Gasless onboarding — users deploy accounts for free |
| `TestWallet` | Deploy server creates wallets without full node setup |
| `SinglePrivateImmutable` | Secure storage for P256 public key notes |
| `AuthWitness` | 64-Field layout for passing P256 signatures |
| `Noir TXE` | Contract unit testing framework |

### What We'd Love to See

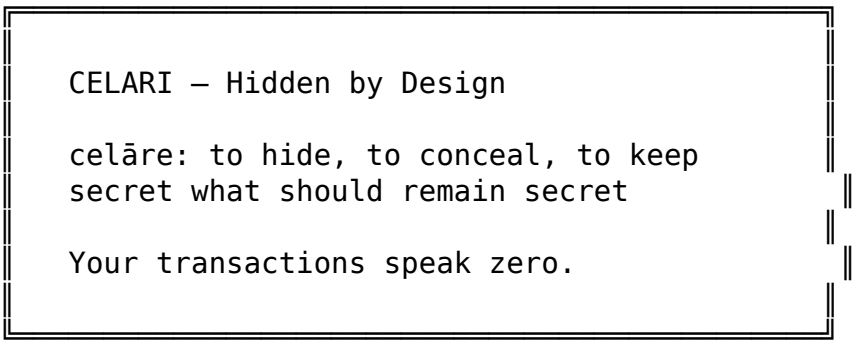
1. **P256 precompile cost reduction** — secp256r1 verification is expensive in Noir circuits. A native precompile would dramatically reduce proof generation time
2. **WebAuthn-aware SDK helpers** — `AuthWitness.fromP256Signature(derBytes)` would help other teams building passkey wallets

3. **PXE browser SDK** — A lightweight PXE client that runs in a service worker would enable real balance queries without a separate node
4. **Account migration standard** — As passkeys sync across devices, a standard for migrating account state between PXE instances would be valuable

### Integration Points for Ecosystem

```
// Any Aztec dApp can integrate Celari in 3 lines:
const { address } = await window.celari.connect();
const result = await window.celari.sendTransaction({ to, amount, token
 ↪  });
console.log("Private transfer complete:", result.txHash);
```

---




---

**Repository:** github.com/Kubudak90/celari-wallet **Built with:** Noir + Aztec SDK v3.0.2 + TypeScript + Chrome MV3 **Contact:** Kubudak90