Openloop

Openloop Connect
dApp Integration App Specification

Version v0.6.11 | Last updated August 8, 2026
Haudi Crypto, Inc. — Representative Director, Kazunori Asada

1. Overview

1.1 Product Name

Openloop Connect - Companion application for the Openloop hardware wallet

1.2 Purpose

Openloop Connect is the bridge software that connects the Openloop hardware wallet with dApps (decentralized applications) around the world.

Primary roles:

1. **dApp integration**: Receives and processes signing requests from over 600 dApps worldwide via the WalletConnect 2.0 protocol
2. **Secure signing**: Forwards received transactions and signing requests to the Openloop hardware wallet, where they are securely signed inside the SE050 secure element
3. **Multi-chain support**: Supports five chain families — Ethereum/EVM chains (EIP155), Bitcoin (BIP122), XRP, Solana, and TRON
4. **Seamless connection**: One-tap connection via QR code, Universal Link, and Deep Link
5. **Firmware updates**: OTA firmware updates for Openloop devices (main + recovery)
6. **Browser extension integration**: Connects to the Openloop device directly from a browser via a local WebSocket server and a Safari Web Extension

1.3 What Is WalletConnect

WalletConnect is an open protocol for connecting wallets and dApps.

- **End-to-end encryption**: Communication between the wallet and the dApp is fully encrypted
- **Chain-agnostic**: Supports many blockchains, including Ethereum, Bitcoin, and Solana

- **Industry standard**: Adopted by major wallets such as MetaMask, Trust Wallet, and Rainbow
- **600+ dApps supported**: Major DeFi/NFT platforms such as Uniswap, OpenSea, Aave, and 1inch are supported

WalletConnect is becoming the **de facto global standard** in the blockchain industry.

1.4 Supported Platforms

Platform	OS	Framework	Connection Method	Status
Desktop	Windows	Electron	USB / BLE	✓ Supported
Desktop	macOS	Electron	USB / BLE	✓ Supported
Desktop	Linux	Electron	USB / BLE	✓ Supported
Mobile	iOS	React Native (Expo)	BLE	✓ Supported
Mobile	Android	React Native (Expo)	BLE	✓ Supported

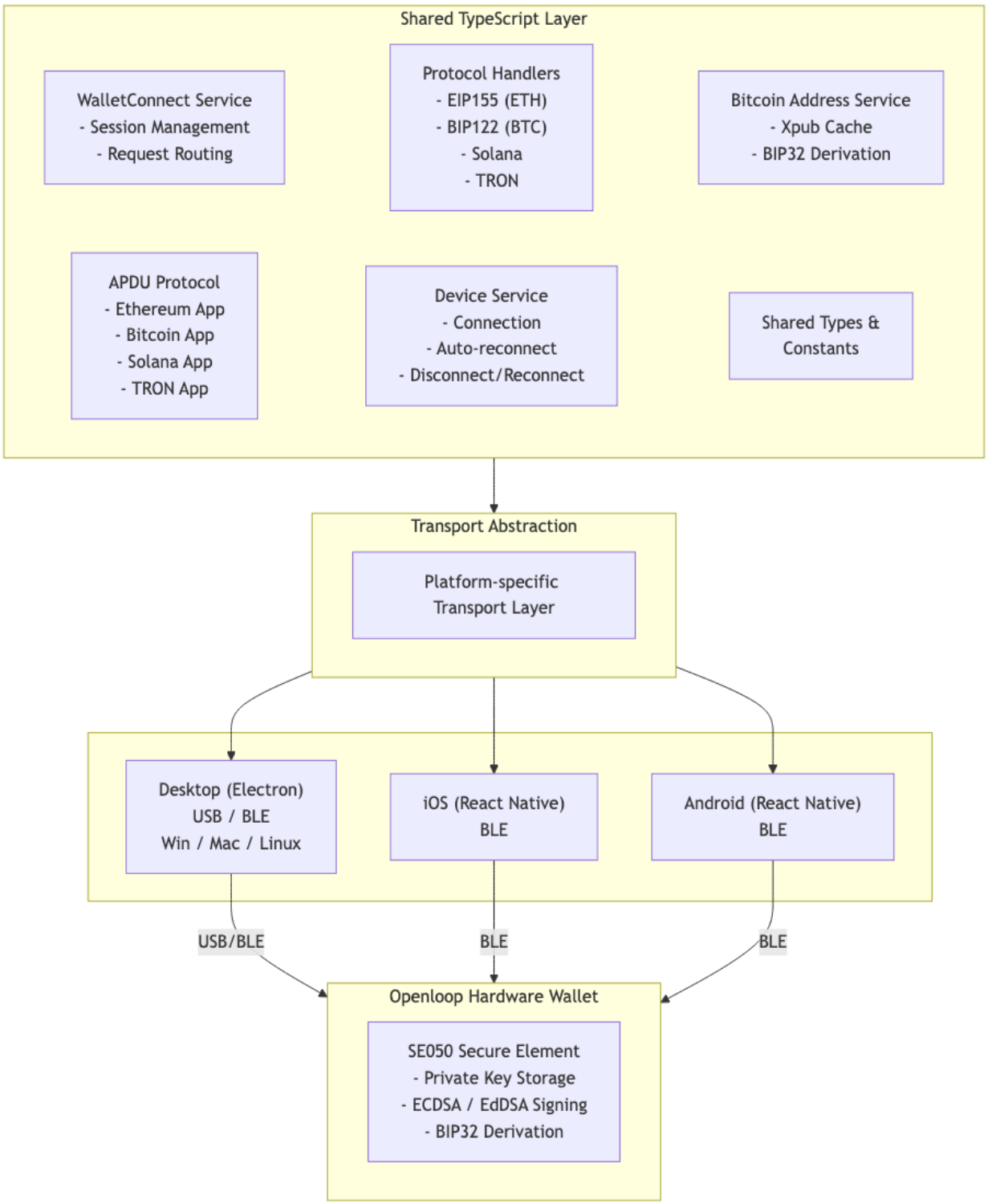
1.5 dApp Connection Methods

Connection Method	Description	Supported Platforms
Deep Link	Select “Openloop Connect” from a dApp’s wallet list to connect automatically (recommended)	All platforms
QR Code	Scan the QR code shown by the dApp	All platforms
Universal Link	Launch the app directly from an HTTPS link	iOS / macOS
Clipboard	Copy and paste a WalletConnect URI	All platforms
LocalWebSocket	A browser extension connects directly via ws://127.0.0.1	Desktop
Safari Web Extension	Connect directly over BLE from iOS Safari	iOS

2. Architecture

2.1 Cross-Platform Design

Openloop Connect adopts a cross-platform architecture that supports **five platforms from a single codebase**.



2.2 Technology Stack

Layer	Technology	Description
Shared logic	TypeScript	WalletConnect, APDU, protocol handlers

Layer	Technology	Description
Desktop UI	Electron + React	Windows / macOS / Linux
Mobile UI	React Native (Expo)	iOS / Android
BLE communication	Platform Native	Each OS's BLE API
USB communication	node-hid	Desktop only
OTA	Secure update	Firmware update protocol
LocalWS	ws (Node.js)	Browser extension bridge
Safari Extension	Swift + Manifest V3	iOS BLE extension
Android Background	HeadlessJsTaskService	Foreground service

2.3 Benefits of Sharing Code

1. **Development efficiency**: Implement the business logic once to support five platforms
2. **Consistency**: Identical WalletConnect handling across all platforms
3. **Maintainability**: Bug fixes and new features are immediately reflected on every platform
4. **Testing efficiency**: Quality is ensured through unit tests of the shared logic

2.4 Device Disconnect / Reconnect

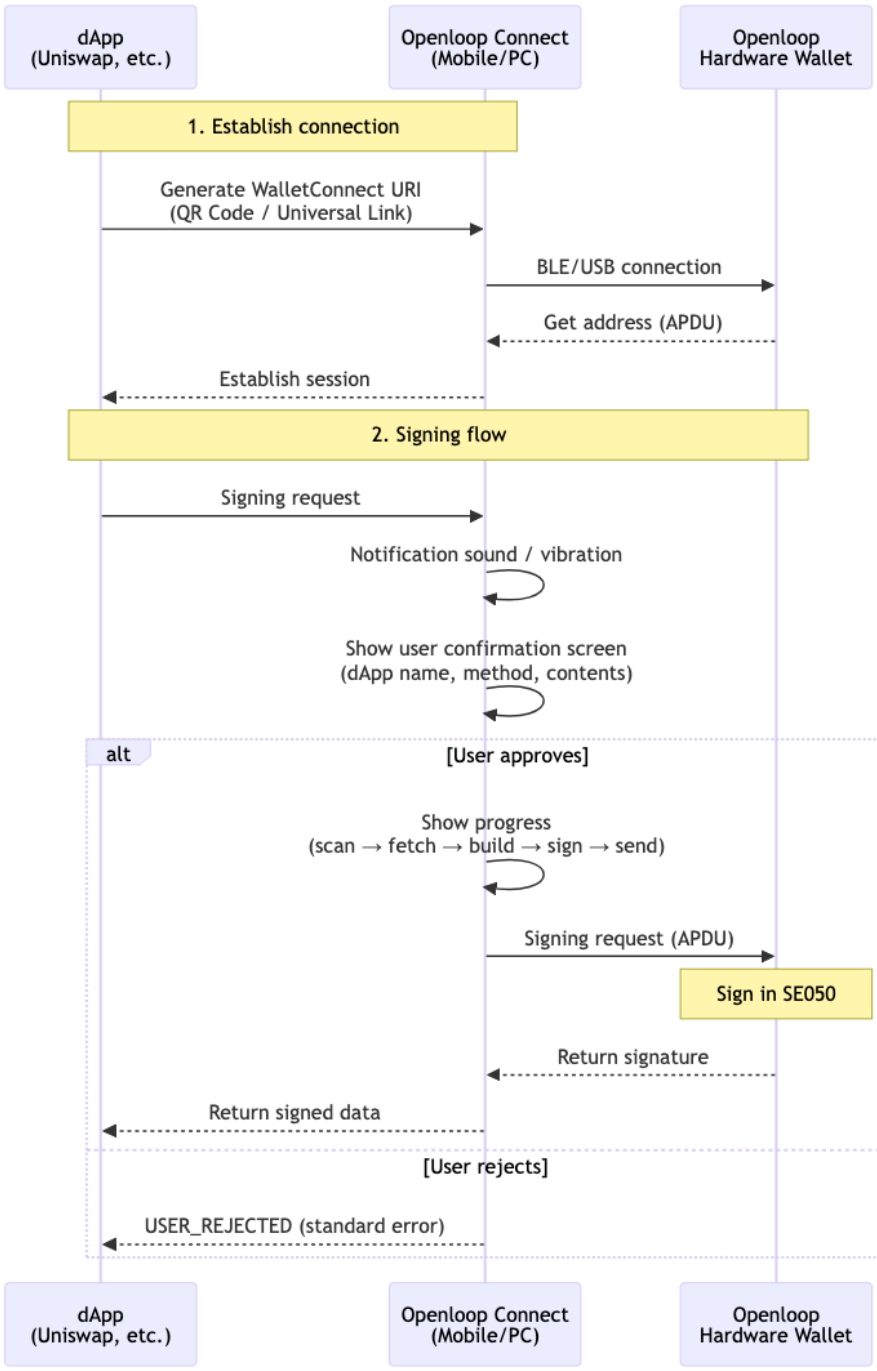
A feature that manually disconnects only the device while keeping the WalletConnect session alive, so that other apps (such as Ledger Live) can access the device.

Platform	Disconnect Method	Reconnect Method
Desktop (USB)	Disconnect button → USB HID close	Reconnect button → resume USB polling
Mobile (BLE)	Disconnect button → BLE disconnect	Automatic (lazy connect on the next operation)

- The WalletConnect / LocalWS session is maintained even while disconnected
- If a different device is detected on reconnect, the WC session is automatically disconnected
- Mobile: The disconnect button is disabled during a WC signing operation or while the WS lock is held

3. dApp Integration Flow

3.1 Connection Flow



Signing confirmation UI:

- Shows the dApp's icon, name, and URL
- Shows the method display name (signMessage, signPsbt, etc.) with multilingual support
- Shows the message/transaction contents in a scrollable area

- Warning text: “Only sign messages from sources you trust”
- Approve / Reject buttons
- While approving: spinner + staged progress status display
- Detects rejection on the device and automatically returns USER_REJECTED to the dApp

Notifications:

- Desktop: system notification sound
- Mobile: vibration (200 ms)
- Notifications are given for both connection requests and signing requests

3.2 Multi-Device Support

Openloop Connect supports the use of multiple Openloop hardware wallets.

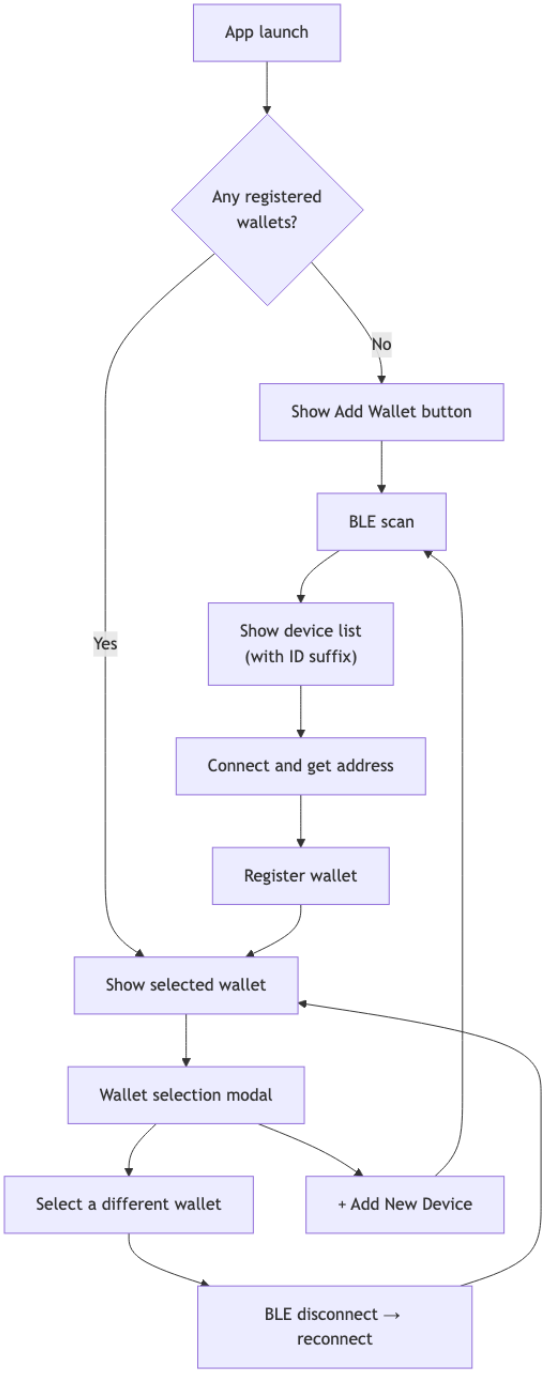
Mobile (BLE): Wallet Registration System

On mobile, you can “register” multiple Openloop devices and “select” which device to use.

Data structure:

```
interface RegisteredWallet {
  id: string          // Unique ID (UUID)
  deviceId: string    // BLE device ID
  deviceName: string  // "Openloop (E854)"
  address: string     // Ethereum address "0x..."
  createdAt: number   // Registration timestamp
  lastUsedAt: number  // Last-used timestamp
}
```

Wallet management flow:



BLE mutual exclusion:

- Only one device can be connected at a time
- When switching wallets, the existing connection is closed before opening a new one
- A mutex prevents concurrent BLE operations

Device identification:

- The scan list shows the first four characters of the BLE device ID as a suffix
- Example: “Openloop (E854)”, “Openloop (D72C)”
- Already-registered devices are excluded from the scan list

Desktop (USB): Physical Switching

On desktop, the USB-connected Openloop device is automatically detected and used.

Multi-device use:

1. Unplug the current device from USB
2. Plug in a different device via USB
3. Openloop Connect automatically detects the new device
4. Connect to the dApp with the new device's address

Detection method:

- USB VID/PID check only (no communication with the device)
- Avoids conflicts with other wallet apps (MetaMask, Rabby, etc.)
- URI input is enabled only when a device is detected

Item	Mobile (BLE)	Desktop (USB)
Registration required	✓ Required	✗ Not required
Switching method	In-app selection	Physically swap the connection
Simultaneous connections	1 device	1 device
Reconnect	Automatic (selected device)	Automatic (connected device)

4. WalletConnect Support

4.1 Protocol Version

Item	Value
WalletConnect	2.0
Relay Server	wss://relay.walletconnect.com

4.2 Supported Namespaces

Namespace	CAIP-2 Identifier	Description
eip155	eip155:1, eip155:11155111, ⋯	Ethereum/EVM chains
bip122	bip122:000000000019d6689c085ae165831e93, ⋯	Bitcoin chains

Namespace	CAIP-2 Identifier	Description
solana	solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp, ⋯	Solana
tron	tron:0x2b6653dc, ⋯	TRON

5. EIP155 (Ethereum/EVM) Support

5.1 Supported Chains

Chain	CAIP-2 Identifier
Ethereum Mainnet	eip155:1
Ethereum Sepolia	eip155:11155111
Other EVM chains	Supported dynamically by Chain ID

5.2 Supported Methods

Method	Description	Status
eth_sendTransaction	Send a transaction	✓
eth_signTransaction	Sign a transaction	✓
eth_sign	Sign a message (deprecated)	✓
personal_sign	Sign a message (EIP-191)	✓
eth_signTypedData	Sign typed data	✓
eth_signTypedData_v3	Sign typed data v3	✓
eth_signTypedData_v4	Sign typed data v4 (EIP-712)	✓
wallet_getCapabilities	Query wallet capabilities (EIP-5792)	✓

5.3 EIP-5792 Wallet Call API

The `wallet_getCapabilities` method is supported. It is used when a dApp queries the wallet's advanced capabilities (atomic batch, paymaster, etc.).

Response format:

```
{
  "0x1": {},
  "0xaa36a7": {}
}
```

As a hardware wallet, it returns no special capabilities (empty objects). The dApp falls back to the standard `eth_sendTransaction`.

5.4 Supported Events

Event	Description
chainChanged	Chain change notification
accountsChanged	Account change notification

6. BIP122 (Bitcoin) Support

6.1 Supported Chains

Chain	CAIP-2 Identifier	Genesis Hash (first 32 chars)
Bitcoin Mainnet	bip122:000000000019d6689c085ae165831e93	000000000019d6689c085ae165831e93
Bitcoin Testnet3	bip122:000000000933ea01ad0ee984209779ba	000000000933ea01ad0ee984209779ba

6.2 Supported Methods

Method	Description	Status
getAccountAddresses	Get the list of BIP32-derived addresses	✓
signPsbt	Sign a PSBT (BIP-174)	✓
signMessage	Sign a message (BIP-137)	✓
sendTransfer	Send Bitcoin	✓

6.3 getAccountAddresses

Specification:

- Always include index 0
- Include all addresses that have UTXOs
- Include at least two unused addresses
- Gap limit: stop after 20 consecutive unused addresses

Implementation:

1. Get the account xpub (m/84'/coin'/0') from the firmware
2. Derive child keys locally with BIP32 (non-hardened)
3. Check UTXOs/history via the mempool.space API
4. Scan receive/change in parallel

Performance:

- Scan time: about 8 seconds (111 addresses)
- API concurrency: 4 (to avoid rate limits)

6.4 signPsbt

Specification:

- Receives a Base64-encoded PSBT
- Reverse-maps address → path from signInputs
- Executes signing in the firmware
- Returns the signed PSBT as Base64

Processing flow:

1. Reverse-map the path from the address cache
2. On a cache miss, search using local derivation
3. Send the PSBT + path information to the firmware
4. Retrieve the signed PSBT

6.5 signMessage

Supported formats:

Format	Description	Status
ECDSA (BIP-137)	Traditional Bitcoin message signing	✓

Format	Description	Status
BIP-322	Generic message signing (P2WPKH witness format)	✓

ECDSA (BIP-137):

- Hash: SHA256(SHA256("18Bitcoin Signed Message:" + varint(len) + message))
- Signature: [V (35-38)][R (32B)][S (32B)]
- Return format: Base64-encoded

BIP-322:

- Builds a to_spend / to_sign transaction pair
- Returns the signature in witness format
- Return format: Base64-encoded

6.6 sendTransfer

Specification:

- The wallet builds the PSBT (fetch UTXOs → estimate fee → create PSBT)
- Executes signing in the firmware
- Broadcasts the signed transaction
- Returns the txid to the dApp

Processing flow:

1. Dust check (reject amounts under 546 sats)
2. Address scan (search for addresses with a balance)
3. Fetch UTXOs (mempool.space API)
4. Estimate fee (using halfHourFee)
5. Select UTXOs (largest first)
6. Build PSBT (attach BIP32_DERIVATION to the change address)
7. Sign in the firmware
8. Broadcast (mempool.space API)
9. Return the txid

---

7. Solana Support

7.1 Supported Chains

Chain	CAIP-2 Identifier
Mainnet Beta	solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp
Devnet	solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1

7.2 Supported Methods

Method	Description	Status
solana_signTransaction	Sign a transaction	✓
solana_signAllTransactions	Sign multiple transactions at once	✓
solana_signAndSendTransaction	Sign + broadcast	✓
solana_signMessage	Sign a message (off-chain)	✓

7.3 Technical Details

Ed25519 signatures. BIP44 path: m/44'/501'/account'/0'

8. TRON Support

8.1 Supported Chains

Chain	CAIP-2 Identifier
Mainnet	tron:0x2b6653dc
Shasta Testnet	tron:0x94a9059e

8.2 Supported Methods

Method	Description	Status
tron_signTransaction	Sign a transaction	

Method	Description	Status
		✓
tron_signMessage	Sign a message	✓

8.3 Technical Details

- secp256k1 (the same curve as Ethereum)
- BIP44 path: m/44'/195'/account'/0/0
- TX hash: SHA-256 (differs from Ethereum's keccak256)
- Broadcast via the TronGrid API

---

9. LocalWebSocket Server

A local WebSocket server that lets browser extensions (such as the Openloop Web SDK) connect to the hardware wallet through Openloop Connect.

9.1 Overview

Item	Value
Protocol	WebSocket (ws://)
Address	127.0.0.1 (localhost only)
Port	21320 (default)
Data format	JSON-RPC 2.0

9.2 Supported Platforms

Platform	Implementation
Desktop (Electron)	Node.js `ws` library
Mobile (Android)	WebSocket server inside a HeadlessJsTaskService
Mobile (iOS)	Via the Safari Web Extension (see §10)

9.3 Security

9.3.1 Network Boundary

- The server **listens on 127.0.0.1 only**. It is unreachable from external networks.
- The `Host` header rejects anything other than `localhost` / `127.0.0.1` (defense against DNS rebinding attacks).

9.3.2 Connection Policy (User-Controlled Toggles)

At the WebSocket handshake, the following are evaluated in order; if any of them is rejected, the connection is dropped with **HTTP 403 Forbidden**.

#	Check	Rejection Condition
1	`Host` header	Anything other than `localhost` / `127.0.0.1`
2	Optional token	A token is configured and the query `?token=...` does not match
3	**LocalWS toggle** or **PKCS#11 toggle**	The user has turned it OFF for the classification by origin (see 9.4)
4	**Origin allowlist**	A LocalWS Web Origin that is not in the registered list (when the list is empty, all are allowed)

The toggles and allowlist can be changed dynamically from the Openloop Connect settings screen, and changes **take effect from the next new connection. Established sessions are not disconnected (active-session protection)**.

9.3.3 Active-Session Protection

The `verifyClient` hook is called **only on a new handshake**. Therefore, even if the user flips a toggle during a connection:

- WebSockets that have already been upgraded continue as usual
- Only new `WebSocket()` calls are rejected with 403

This is a design decision to avoid suddenly breaking Adobe Acrobat in the middle of PDF signing, or Firefox in the middle of TLS client authentication (because the PKCS#11 library goes through the same server, the same protection applies to PKCS#11 sessions).

9.4 Connection Classification by Origin

The server classifies connections into two types based on the `Origin` header of the handshake.

Origin Value	Classification	Applicable Toggle	Allowlist
`openloop-pkcs11`	Bundled PKCS#11 library	**PKCS#11 toggle**	Not applied
`http://...` / `https://...` / other	Web dApp / SDK / native app	**LocalWS toggle**	Applied

The PKCS#11 library sends the fixed string `Origin: openloop-pkcs11`. This makes it possible to control PKCS#11 access independently of the Web-facing toggle and allowlist (turning PKCS#11 OFF does not block dApp connections from the Web, and vice versa).

9.5 Origin Allowlist (for Web Connections)

The user can register allowed origins on the Openloop Connect settings screen. dApp developers must have their origin registered before connecting.

9.5.1 Origin Format

An Origin follows the WHATWG URL Standard's Origin (RFC 6454 §4) and is a string combining the following three elements:

```
<scheme>://<host>[:<port>]
```

- **scheme**: `http` or `https`
- **host**: a domain name (including IDN) or an IP address
- **port**: normalized to the scheme default (http:80, https:443) when omitted

Examples:

Input	Stored Origin
`https://dapp.example.com`	`https://dapp.example.com`
`https://dapp.example.com/path`	`https://dapp.example.com` (path discarded)
`https://dapp.example.com:8443`	`https://dapp.example.com:8443`
`https://www.example.com`	`https://www.example.com`
`https://example.com`	`https://example.com` (distinct from the previous row)

> ⚠ An Origin **does not include the path**. `https://example.com/dapp` and `https://example.com/admin` are treated as the same Origin, so registering it makes both connectable. dApp providers should use a domain where user content and the dApp do not coexist (or separate them by subdomain).

9.5.2 Matching Rules

- When the list is **empty**: accept any Web Origin (legacy / first-run behavior)
- When **one or more** origins are registered: only **exact matches** are allowed. Non-matching origins get 403
- Wildcards (`*.example.com`, etc.) are **not supported**
- `Origin: null` (sandboxed iframe / `file://`, etc.) is subject to exact matching, so it passes only when the list is empty

9.5.3 Preventing Origin Spoofing by the Browser

The browser does not allow JavaScript to rewrite the `Origin` header of a WebSocket handshake. Therefore, a malicious web page cannot impersonate another site's Origin. Note, however, that **native processes** can set an Origin freely, so the allowlist is only a protection against browser-originated dApps.

9.6 Integration Steps for dApp Developers

1. The dApp attempts to connect to `ws://127.0.0.1:21320`
2. If the handshake fails (`403 Forbidden`), guide the user to check the following:
   - Whether Openloop Connect is running
   - Whether "Local WebSocket server" in the settings screen is **ON**
   - Whether the site's own Origin is registered under "Allowed sites" in the settings screen (if the list has entries)
3. The dApp should **handle connection failures gracefully** (there will naturally be users who do not have Openloop running)
4. After a successful connection, send and receive APDUs over JSON-RPC 2.0 as described in 9.1 / the API reference

The retry interval, caching, etc. on connection failure are entirely up to the dApp's design. Openloop Connect does not show a connection-approval UI (allowlist registration is a one-time operation).

---

10. Safari Web Extension (iOS)

A Safari Web Extension for connecting to the Openloop hardware wallet over BLE from iOS Safari. The industry's first Safari BLE extension for a hardware wallet.

10.1 Architecture

Component	Technology	Role
Extension Process	Swift + CoreBluetooth	BLE communication
Content Script	JavaScript	Inject the Web3 provider
Background Script	JavaScript (Manifest V3)	Message routing
Popup	HTML/CSS/JS	User UI
Container App	React Native (Expo)	BLE permission management

10.2 Communication Flow

dApp → Content Script → Background → Native Messaging → Swift BLE → Openloop

10.3 Limitations

- iOS Safari only (macOS Safari is not supported)
- Requires installing the Container App (Openloop Connect for iOS)
- BLE permission is granted for the first time from the Container App

---

11. Android Background Operation

The Android version uses a foreground service to keep the WebSocket server and the BLE connection alive even when the app is in the background.

11.1 Implementation

Component	Description
WsServerService	HeadlessJsTaskService + PARTIAL_WAKE_LOCK + WIFI_LOCK
Notification	Runs as a foreground service via a persistent notification
Permissions	

Component	Description
	FOREGROUND_SERVICE, FOREGROUND_SERVICE_CONNECTED_DEVICE, WAKE_LOCK

11.2 Behavior

- The WS server responds even when the app is in the background
- Maintains the BLE connection and processes signing requests
- The iOS version does not support background WebSocket due to OS limitations (complemented via the Safari Extension)

12. OTA Firmware Update

Openloop Connect provides the ability to update the Openloop device's firmware wirelessly or over USB.

12.1 Supported Transports

Transport	Platform
USB	Desktop
BLE	Mobile (iOS/Android)

12.2 OTA Targets

Target	Description
Main firmware	The device's core features (wallet, passkey, etc.)
Recovery firmware	FW for recovery mode (OTA Recovery Mini)

12.3 Protocol

Item	Specification
Chunk verification	Split the transferred data and verify integrity
Final verification	SHA-256 hash
Signature verification	RSA-3072 (device side)

12.4 OTA Flow

1. Check the version
2. Start the update
3. Transfer chunks (integrity verification)
4. Complete (SHA-256 verification)
5. RSA-3072 signature verification on the device → apply and reboot

12.5 HID-Preferred Strategy

On desktop, the HID transport is used preferentially.

---

13. Communication Protocol

13.1 BLE Communication

Service UUIDs:

Item	UUID
Service	13D63400-2C97-0004-0000-4C6564676572
Notify Characteristic	13D63400-2C97-0004-0001-4C6564676572
Write Characteristic	13D63400-2C97-0004-0002-4C6564676572

Frame protocol:

An industry-standard BLE frame-splitting protocol.

Item	Value
Frame prefix	0x05
First frame header	5 bytes (PREFIX + INDEX + SIZE)
Continuation frame header	3 bytes (PREFIX + INDEX)
MTU	247 bytes

13.2 USB Communication (Desktop Only)

Mode	VID	PID	Notes
Native mode	0x303A	0x8341	

Mode	VID	PID	Notes
			Espressif VID / Openloop PID
Compatibility mode	0x2C97	0x1011	Compatible VID / Nano S-compatible PID

- Interface: HID
- The mode can be switched from the device settings screen

HID response corruption detection:

- Detects corrupted responses and retries automatically (up to 3 times; errors out on failure)

13.3 Automatic Reconnect

Because the BLE connection may drop during BIP122's getAccountAddresses (about 8 seconds), automatic reconnect is performed in the following methods:

- signPsbt()
- signPsbtWithPaths()
- getAccountXpub()
- signBitcoinMessage()

---

14. APDU Protocol

APDU distinguishes command groups by their CLA (class byte). Within the same CLA, the INS (instruction byte) is shared across currencies and purposes, and the target currency is determined by the request parameters (path, data body). Therefore, the INS values shown separately for "Ethereum," "Solana," "TRON," etc. in the following sections are not different commands when the CLA is the same; they are the same INS reused per currency.

14.1 Industry-Standard / Extended Commands (CLA=0xE0)

A set of industry-standard commands based on Ledger compatibility, with Openloop's own extensions added (ERC20/NFT information provision, EIP-712 streaming, EIP-7702, etc.). The CLA is 0xE0 in common, and the INS is shared via currency determination.

Handshake / read:

INS	Command	Description
0x01	GET_VERSION	Get the firmware version
0xD8	OPEN_APP	Switch apps (Safe/MetaMask compatible)

Ethereum (EVM):

INS	Command	Description
0x02	GET_PUBLIC_KEY	Get the public key / address
0x04	SIGN	Sign a transaction
0x08	SIGN_PERSONAL_MESSAGE	Sign a message (EIP-191)
0x0C	SIGN_EIP712	EIP-712 signing
0x14	PROVIDE_ERC20_TOKEN_INFORMATION	Provide ERC20 token information
0x16	PROVIDE_NFT_INFORMATION	Provide NFT collection information
0x1A	EIP712_STRUCT_DEFINITION	EIP-712 struct definition (streaming)
0x1C	EIP712_STRUCT_IMPLEMENTATION	EIP-712 struct implementation (streaming)
0x1E	EIP712_FILTERING	EIP-712 display filtering
0x34	SIGN_AUTHORIZATION	EIP-7702 authorization signing

XRP: Address retrieval and transaction signing share the same INS as Ethereum (0x02 / 0x04); the request's path and body cause it to be processed as XRP.

Bitcoin (Ledger-compatible flow):

INS	Command	Description
0x40	GET_WALLET_PUBLIC_KEY	Get the public key / address
0x42	GET_TRUSTED_INPUT	Generate a Trusted Input (UTXO verification)
0x44	UNTRUSTED_HASH_TX_INPUT_START	Start signing-hash computation
0x48	UNTRUSTED_HASH_SIGN	Sign an input
0x4A		

INS	Command	Description
	UNTRUSTED_HASH_TX_INPUT_FINALIZE	Finalize signing-hash computation

14.2 Solana APDU Commands (CLA=0xE0)

As part of CLA=0xE0, dedicated Solana INS values are assigned.

INS	Command	Description
0x05	GET_PUBKEY	Get the public key (Ed25519)
0x06	SIGN_MESSAGE	Sign a transaction
0x07	SIGN_OFFCHAIN_MESSAGE	Sign an off-chain message

14.3 TRON APDU Commands (CLA=0xE0)

Part of CLA=0xE0. GET_ADDRESS / SIGN_TX share the same INS as the EVM family (0x02 / 0x04) and are processed as TRON based on the request contents.

INS	Command	Description
0x02	GET_ADDRESS	Get the TRON address (Base58Check)
0x04	SIGN_TX	Sign a transaction (SHA-256)
0x08	SIGN_PERSONAL_MESSAGE	Sign a message

14.4 Bitcoin v2 Commands (CLA=0xE1)

Read-only extended-public-key and master-fingerprint retrieval. The signing commands (register/get address/sign PSBT) are not implemented in this version; Bitcoin signing uses the Ledger-compatible flow in 14.1 and the Openloop PSBT flow in 14.5.

INS	Command	Description
0x00	GET_EXTENDED_PUBKEY	Get the extended public key (xpub)
0x05	GET_MASTER_FINGERPRINT	Get the master fingerprint

14.5 Openloop Extended Commands (CLA=0xF0)

INS	Command	Description
0x08	SIGN_PERSONAL_MESSAGE	Sign a message (CLA_OPENLOOP version, with chain_id)
0x0C	SIGN_EIP712	EIP-712 signing (CLA_OPENLOOP version, with chain_id)
0x70	SIGN_PSBT	Receive PSBT data and start signing
0x71	GET_SIGNED_PSBT	Get the signed PSBT
0x72	GET_ACCOUNT_XPUB	Get the account xpub
0x73	SIGN_MESSAGE	Sign a message (BIP-137)
0xA0	GET_DEVICE_INFO	Get device information

14.6 Common Commands (CLA=0xB0)

INS	Command	Description
0x01	OPEN_APP / GET_APP_INFO	Query the current app (Lc=0) / switch apps (Lc>0)
0xA7	GET_BATTERY_STATUS	Get the battery status

14.7 Protocol Details

For details, see `BIP122_signPsbt_Implementation_Spec.md`.

15. File Structure

```
openloop-connect/
├── packages/
│   ├── shared/                      # Shared TypeScript code
│   │   └── src/
│   │       ├── apdu/                # APDU protocol
│   │       │   ├── ethereum-app.ts
│   │       │   ├── bitcoin-app.ts
│   │       │   ├── solana-app.ts        # Solana APDU
│   │       │   └── tron-app.ts          # TRON APDU
│   │       ├── bitcoin/             # Shared Bitcoin logic
│   │       │   ├── address-cache.ts     # Address ↔ path mapping cache
│   │       │   ├── address-service.ts   # Gap-limit scan / address reverse lookup
│   │       │   ├── bip32-derive.ts      # BIP32 public key derivation
│   │       │   ├── bip322.ts            # BIP-322 message-signing utilities
│   │       │   ├── tx-service.ts        # PSBT build / UTXO fetch / broadcast
│   │       │   └── index.ts
│   │       ├── ethereum/            # Ethereum-related utilities
│   │       │   ├── index.ts
│   │       │   ├── mini-ethers.ts
│   │       │   ├── transaction.ts
│   │       │   └── typed-data.ts
│   │       ├── codec/               # Encoding
│   │       │   └── base58.ts            # Base58 codec
│   │       ├── ota/                 # OTA protocol
│   │       ├── walletconnect/       # WalletConnect
│   │       │   └── namespace-builder.ts  # Namespace builder
│   │       ├── sign-handler/        # Shared signing handlers
│   │       ├── i18n/                # Shared multilingual strings
│   │       │   ├── wc-strings.ts        # WalletConnect-related (EN/JA)
│   │       │   └── index.ts
│   │       ├── transport/
│   │       │   └── interface.ts     # Transport abstraction interface
│   │       ├── constants.ts         # Constants / mapErrorToI18nKey()
│   │       ├── ws-protocol.ts       # WS protocol definitions
│   │       ├── index.ts
│   │       └── types.ts
│   │
│   ├── desktop/                     # Desktop app (Electron)
│   │   └── src/
│   │       ├── main/                # Main process
│   │       │   ├── hid-service.ts       # USB HID communication (node-hid)
│   │       │   ├── ws-server.ts         # LocalWebSocket server
│   │       │   ├── ota-service.ts       # Desktop OTA
│   │       │   ├── device-service.ts    # Device disconnect/reconnect management
│   │       │   ├── walletconnect-service.ts
│   │       │   ├── index.ts             # Signing hold / confirmation / progress management
│   │       │   └── utils.ts
│   │       ├── preload/
│   │       │   └── index.ts
│   │       └── renderer/            # Renderer process (React)
```

```
│   │           ├── App.tsx               # Includes the signing confirmation modal
│   │           ├── i18n/                 # Desktop-specific + shared i18n
│   │           └── main.tsx
│   │
│   └── mobile/                       # Mobile app (React Native)
│       ├── App.tsx
│       ├── ios/
│       │   └── SafariExtension/      # Safari Web Extension
│       └── src/
│           ├── transport/            # BLE transport
│           │   ├── BleTransport.ts
│           │   └── index.ts
│           ├── services/             # Mobile-specific services
│           │   ├── device.ts             # BLE connection management
│           │   ├── wallet-storage.ts     # Wallet registration / selection
│           │   ├── walletconnect.ts
│           │   ├── ota-service.ts        # Mobile OTA
│           │   └── ws-server.ts          # Mobile WebSocket server
│           ├── screens/              # UI screens
│           │   ├── ConnectScreen.tsx     # dApp connection (main screen)
│           │   ├── SignScreen.tsx        # Signing confirmation
│           │   ├── ScanDeviceScreen.tsx  # BLE scan
│           │   └── WalletAddedScreen.tsx # Registration complete
│           ├── i18n/                 # Mobile-specific + shared i18n
│           └── types/
│               └── navigation.ts
│
└── docs/
    ├── OpenloopConnect_Spec_2026.md            # This specification
    └── BIP122_signPsbt_Implementation_Spec.md  # BIP122 implementation spec
```

---

16. Vision

16.1 Significance of WalletConnect Support

WalletConnect is the de facto standard protocol of the blockchain industry, adopted by **more than 600 dApps**.

By fully supporting WalletConnect, Openloop Connect achieves:

- **Instant ecosystem participation**: Openloop can be used with major dApps such as Uniswap, OpenSea, Aave, and 1inch
- **No per-dApp work required**: When a new dApp adds WalletConnect support, it becomes usable with no additional development on the Openloop side
- **Industry-standard compliance**: dApp integration capability on par with major hardware wallets

16.2 Expanding Native Mode

Openloop Connect is the primary execution platform for native mode (VID: 0x303A / PID: 0x8341), and the connection methods it supports keep expanding.

Native-mode connections via Openloop Connect:

Connection Method	Description	Platform
WalletConnect	Receive signing requests from 600+ dApps	All platforms
LocalWebSocket	A browser extension connects directly via ws://127.0.0.1	Desktop / Android
Safari Web Extension	Connect directly over BLE from iOS Safari	iOS

As a result, **the use cases that native mode alone can cover are expanding rapidly** — WalletConnect-enabled dApps, browser extensions supporting the Openloop Web SDK, Web3 dApps on Safari, and more.

For the full picture of the native-mode strategy (including WebHID), see the Openloop product specification.

16.3 Future Outlook

1. **Growth of the WalletConnect ecosystem**: The number of supported dApps keeps rising
2. **Growth of Bitcoin dApps**: BIP122 support opens the door to the Bitcoin DeFi/Ordinals market
3. **Expansion of native features**: Advanced security features unique to a hardware wallet
4. **Multisig support**: Joint signing with multiple Openloop devices

---

17. References

17.1 WalletConnect

Document	URL
WalletConnect 2.0 Specs	https://specs.walletconnect.com/
WalletConnect Cloud	https://cloud.walletconnect.com/
Sign API	https://docs.walletconnect.com/api/sign

Document	URL
React Native SDK	https://docs.walletconnect.com/advanced/walletconnectmodal/about?platform=react-native
Electron Integration	https://docs.walletconnect.com/advanced/walletconnectmodal/about?platform=web

17.2 CAIP (Chain Agnostic Improvement Proposals)

Standard	Description	URL
CAIP-2	Blockchain ID Specification	https://github.com/ChainAgnostic/CAIPs/blob/main/CAIPs/caip-2.md
CAIP-10	Account ID Specification	https://github.com/ChainAgnostic/CAIPs/blob/main/CAIPs/caip-10.md
CAIP-25	Wallet JSON-RPC Methods	https://github.com/ChainAgnostic/CAIPs/blob/main/CAIPs/caip-25.md

17.3 Bitcoin BIP122

Document	URL
BIP122 Scope (WalletConnect)	https://docs.walletconnect.com/advanced/multichain/rpc-reference/bitcoin-rpc
CAIP-122 (signMessage)	https://github.com/ChainAgnostic/CAIPs/blob/main/CAIPs/caip-122.md

17.4 Ethereum EIP155

Document	URL
EIP155 Scope (WalletConnect)	https://docs.walletconnect.com/advanced/multichain/rpc-reference/ethereum-rpc
EIP-191	https://eips.ethereum.org/EIPS/eip-191
EIP-712	https://eips.ethereum.org/EIPS/eip-712
EIP-5792 (Wallet Call API)	https://eips.ethereum.org/EIPS/eip-5792

17.5 Bitcoin Standards

Standard	Description	URL
BIP-32	HD Wallets	https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
BIP-84	Native SegWit	https://github.com/bitcoin/bips/blob/master/bip-0084.mediawiki
BIP-137	Message Signing	https://github.com/bitcoin/bips/blob/master/bip-0137.mediawiki
BIP-174	PSBT	https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki

17.6 Solana

Document	URL
Solana WalletConnect Scope	https://docs.walletconnect.com/advanced/multichain/rpc-reference/solana-rpc
Solana Web3.js	https://solana-labs.github.io/solana-web3.js/
Ledger Solana App	https://github.com/LedgerHQ/app-solana

17.7 TRON

Document	URL
TronGrid API	https://www.trongrid.io/
Ledger TRON App	https://github.com/AurelienMusic/app-tron
TIP-191 (signMessage)	https://github.com/tronprotocol/TIPs/blob/master/tip-191.md

17.8 OTA / Firmware

Document	URL
CBOR (RFC 8949)	https://www.rfc-editor.org/rfc/rfc8949
FIDO2 / WebAuthn	https://fidoalliance.org/fido2/

18. Change History

Version	Date	Contents
v0.1.0	2026-01-18	Initial version (EIP155 + BIP122 support)
v0.2.0	2026-01-18	Major revision: added desktop support, detailed architecture, added vision, Mermaid diagram support
v0.2.2	2026-01-19	Multi-device support: mobile BLE wallet registration system, desktop USB physical switching
v0.3.0	2026-02-14	Desktop BIP122 support, shared Bitcoin service (moved to shared)
v0.3.1	2026-02-14	Added signing confirmation UI (desktop), standardized error responses, shared i18n module
v0.3.4	2026-02-15	EIP-5792 wallet_getCapabilities support, notification sound/vibration, unified error messages, fixed sendTransfer spec, added BIP-322 note
v0.3.6	2026-02-17	Solana WalletConnect support, published on App Store / Play Store
v0.3.10	2026-02-17	APDU sharing (CLA_OPENLOOP), TRON WalletConnect support
v0.3.12	2026-02-25	Unified Desktop/Mobile UI, fixed GitHub Actions CI
v0.4.1	2026-03-11	OTA USB HID support, LocalWebSocket server, Safari Web Extension, Android background
v0.4.3	2026-03-19	Device disconnect/reconnect, restored Android foreground service
v0.6.10	2026-05-13	User-controlled toggles for LocalWS / PKCS#11, Origin allowlist, PKCS#11 lib Origin

Version	Date	Contents
		identification convention (§9.3-9.6 major revision)
v0.6.11	2026-07-07	Aligned the §14 APDU spec with the implementation: added CLA=0xE1 (xpub/master fingerprint), BTC signing flow (42/44/48/4A), EIP-712 streaming (1A/1C/1E), EIP-7702 (0x34), ERC20/NFT (14/16), handshake commands (GET_VERSION/OPEN_APP/ GET_DEVICE_INFO/ GET_BATTERY_STATUS); reorganized per-currency CLA notation

Openloop Connect - Haudi Crypto, Inc.