ZOOBC (ZBC) · INTEGRATION SPECIFICATION

# Hardware Wallet Signing Specification

Prepared for hardware-wallet firmware and companion-app engineers.

| | |
|---|---|
| REVISION | DATE |
| Version 2.1 | 10 September 2026 |
| SIGNATURE SCHEME | DERIVATION PATH |
| Ed25519 over SHA3-256 | m/44'/883'/account' |
| SLIP-44 COIN TYPE | VERIFIED AGAINST |
| 883 \| ZBC | ZooBC node v0.4.0 · 7ad161a4 |

Every statement in this document was checked against ZooBC node v0.4.0 and against transaction vectors a live chain has accepted. Source files are cited so a reviewer can re-check any line.

# Contents

# 0 One-page summary

| | |
|---|---|
| **Signature scheme** | **Ed25519** (RFC 8032, pure, no prehash variant, no context string). libsodium `crypto_sign_ed25519_detached`. 64-byte signature, 32-byte public key. |
| **What is signed** | `Ed25519.sign(sk, SHA3-256("ZBC-TX" ‖ genesis_hash(32) ‖ unsigned_transaction_bytes))`. The 32-byte digest **is** the Ed25519 message. See §3. |
| **Chain binding** | The digest covers the chain's **genesis block hash**, so a signature made for one chain cannot be replayed on another. The wire format is unchanged; the genesis hash is not a transaction field. |
| **Signing tag** | `"ZBC-TX"`, six ASCII bytes `5a 42 43 2d 54 58`. No terminator, no length prefix. |
| **Hash function** | **SHA3-256 (FIPS 202)**. Not Keccak-256, not SHA-256. |
| **Key derivation** | BIP-39 mnemonic → seed (PBKDF2-HMAC-SHA512, 2048 rounds) → **SLIP-0010 Ed25519**, path **`m/44'/883'/account'`** (three hardened levels; no change/index levels). |
| **SLIP-44 coin type** | **883** (registered: `883 \| ZBC \| ZooBC`). |
| **Account (on-wire)** | 36 bytes: `u32le(0)` ‖ `pubkey(32)`. Hex form is what the JSON API takes. |
| **Address (display)** | `ZBC_` + 56 base32 chars in 7 groups of 8, e.g. `ZBC_L2HLFDOM_VKKKTEXX_C2P2M6LG_EB6ZNKSV_356SJUWW_5QPVHDE7_EFJA3PEX`. 3-byte checksum = `SHA3-256(pubkey ‖ "ZBC")[0..3]`. |
| **Transaction version byte** | **`0x01`, unchanged.** It was not bumped for chain binding and is not a version discriminator. See §3.3. |
| **Native unit** | 1 ZBC = 100 000 000 atomic (8 decimals). Amounts and fees are `u64` little-endian atomic units. |
| **Byte order** | Everything on the wire is **little-endian**, except SLIP-0010 child indexes (big-endian, per the standard). |
| **Transaction id / hash** | `tx_hash = SHA3-256(unsigned_bytes ‖ signature)`; `tx_id = int64le(tx_hash[0..8])`. Unchanged; the genesis hash enters only the signing digest. |
| **Replay protection** | Chain binding (above), plus hash uniqueness, plus a **3600 s** inclusion window. There is no nonce. See §5.4. |
| **Minimum fee** | 0.025 ZBC base (2 500 000 atomic) × network fee scale + size rent. Ask the node: `GET /api/v1/blockchain/estimate-fee`. |
| **Recommended Tier 1 types** | **1 SendZBC**, **11 TransferToken**, **4 ApprovalEscrow** (§6). |
| **Chain discovery** | `GET /api/v1/node/info` returns `genesis_hash`, `signing_version: 2` and `signing_tag: "ZBC-TX"`. Read it from the endpoint you will broadcast to. See §8.3. |

## 0.1 What changed since draft 1

Draft 1 documented node v0.3.2, under which a signature was portable across ZooBC chains. Transactions are now bound to the chain they are signed for. Four things changed, and two of them are breaking for anyone who built against draft 1.

| # | CHANGE | BREAKING? |
|---|---|---|
| 1 | The signing digest is now `SHA3-256("ZBC-TX" ‖ genesis_hash ‖ unsigned_bytes)` (§3) | **Yes** |
| 2 | **ApprovalEscrow (type 4)** body is 36 bytes, `approval u32le` ‖ **escrowed transaction hash (32)**, replacing the 12-byte form that carried an 8-byte id (§6.1, §7.3) | **Yes** |
| 3 | `GET /api/v1/accounts/<a>/escrows` returns `transaction_hash` per escrow, the value that goes in that body (§9.1) | No, additive |
| 4 | `GET /api/v1/node/info` returns `signing_version` and `signing_tag`; on the archival/gateway path it now also reports the mirrored node's real `version` instead of a `0.1.0` placeholder (§9.1) | No, additive |

Everything else a wallet touches is byte-for-byte unchanged, verified by diffing the files that define it: the 11-field envelope, the body layout of every type other than ApprovalEscrow, the type set (still 56), the fee formula and `estimate-fee`, account bytes and address encoding, the 17 recipient account types, every other route and response shape in §9, the 3600 s inclusion window, the 60-minute mempool expiry, the tip − 3 confirmed read, and the escrow block including its trailing `multi_party` byte.

**Nothing implements the v0.3.2 construction.** Testnet moves to v0.4.0 imminently and mainnet will launch on v0.4.0 or later, so a hardware wallet should implement §3 only. The legacy digest appears in this document exactly twice: in this paragraph, and as the "legacy signature" line in each §10 vector, where it is printed solely so an implementer can confirm they are *not* producing it.

# 1 Cryptographic primitives

## 1.1 Ed25519

- Standard Ed25519 as in RFC 8032 / libsodium. The node verifies with `crypto_sign_ed25519_verify_detached` (`src/crypto/signature.cpp:68`), reached through a dispatcher that selects the algorithm from the sender's account type (`src/crypto/signature.cpp:416`).
- No Ed25519ph, no Ed25519ctx, no domain-separation prefix **inside Ed25519**. The message handed to Ed25519 is the 32-byte SHA3-256 digest defined in §3. Domain separation is done by the `"ZBC-TX"` tag inside the SHA3 preimage, not by an Ed25519 context.
- Public key = 32 bytes. Signature = 64 bytes (`R ‖ S`). Secret = the 32-byte SLIP-0010 leaf, expanded per RFC 8032.
- A ZBC-type sender always uses Ed25519. Other sender account types select other algorithms through the same dispatcher (sr25519 for Polkadot, secp256k1 for ETH/Tron and BTC/Ripple), but the digest handed to them is the same 32 bytes (§10 scope note in §5.6).

## 1.2 SHA3-256

- FIPS 202 SHA3-256 (OpenSSL `EVP_sha3_256`). Used for: the signing digest, the transaction hash, the address checksum, and the escrowed-transaction hash in an ApprovalEscrow body.
- Do not substitute Keccak-256 (Ethereum's variant); the padding differs and every signature will fail with the node's bare `"Invalid transaction signature"`.

## 1.3 Key derivation (BIP-39 + SLIP-0010)

Reference: `zoobc-signer/extension/lib/zoobc-crypto.js` (`mnemonicToSeed`, `deriveZbcSeed`), byte-identical to the reference wallet; node-side equivalents in `src/crypto/slip10.cpp`.

1. `seed = PBKDF2-HMAC-SHA512(password = NFKD(mnemonic), salt = NFKD("mnemonic" ‖ passphrase), 2048 iterations, 64 bytes)`.
2. `I = HMAC-SHA512(key = "ed25519 seed", data = seed)`; `k = I[0..32]`, `c = I[32..64]`.
3. For each index in `[44, 883, account]`, all hardened: `data = 0x00 ‖ k ‖ u32be(index + 0x80000000)`; `I = HMAC-SHA512(c, data)`; `k = I[0..32]`, `c = I[32..64]`.
4. The final `k` is the 32-byte Ed25519 seed. `pubkey = Ed25519.publicKeyFromSeed(k)`.

There are **no** change or address-index levels. Account 0 is `m/44'/883'/0'`, account 1 is `m/44'/883'/1'`, and so on.

Key derivation is **not** affected by chain binding. One key set serves every ZooBC chain; only the signature differs.

## 1.4 Account bytes and display address

- **Account (wire form):** `u32le(account_type) ‖ payload`. For a ZooBC key: `00 00 00 00` ‖ pubkey(32) = 36 bytes. The JSON API takes this as 72 hex characters.
- **Display address** (`src/crypto/slip10.cpp:595-650`, `ZoobcAddress::Encode`): 1. `buf[35] = pubkey(32) ‖ "ZBC"` (three ASCII bytes). 2. `h = SHA3-256(buf)`; overwrite `buf[32..35] = h[0..3]`. 3. `s = base32(buf)` with the RFC 4648 alphabet `A-Z2-7`, no padding → exactly 56 characters. 4. Output `"ZBC" + "_" + s[0..8] + "_" + s[8..16] + …` (7 groups). Canonical length 66.
- **Decoding** accepts underscores, dashes, spaces or no separators, any case; the checksum must be recomputed and compared. The body must be exactly 56 base32 characters.
- The 3-byte SHA3 checksum above is the definition the code and the chain use; implement from it.

## 1.5 Account types (recipient parsing)

The 4-byte type prefix decides the payload length that follows it (`src/util/transaction_util.cpp:466-515`, `GetAccountPublicKeyLength`). A device only ever **signs** as type 0, but it must **parse and display** any type as a recipient.

| TYPE | NAME | PAYLOAD BYTES | DISPLAY FORM (NATIVE) |
|---|---|---|---|
| 0 | ZBC | 32 (Ed25519 pubkey) | `ZBC_...` (§1.4) |
| 1 | BTC legacy (ambiguous) | 20 | hex |
| 2 | Empty | **0** — the 4-byte marker `02 00 00 00` alone means "no recipient" | "(none)" |
| 3 | Estonia eID | 32 | hex |
| 4 | ETH | 20 | `0x` + hex (EIP-55 optional) |
| 5 | BTC P2PKH | 20 | Base58Check `1...` |
| 6 | BTC P2SH | 20 | Base58Check `3...` |
| 7 | BTC P2WPKH | 20 | bech32 `bc1q...` |
| 8 | BTC P2WSH | 32 | bech32 `bc1q...` |
| 9 | BTC Taproot | 32 | bech32m `bc1p...` |
| 10 | DataSet object (`ZBS_`) | 32 | `ZBS_...` (same encoding as ZBC with prefix `ZBS`) |
| 11 | Solana | 32 | base58 |
| 12 | Polkadot | 32 | SS58 |
| 13 | Cardano | 28 (blake2b-224 key hash) | bech32 `addr1...` |
| 14 | Ripple | 20 | base58 `r...` |
| 15 | Tron | 20 | base58check `T...` |
| 16 | Tezos | 20 (blake2b-160) | base58check `tz1...` |

A standard-edition firmware may render non-ZBC recipients as `<type name> + hex` and still be safe; what matters is that the **entire** payload is shown (see §7.2).

# 2 Unsigned transaction wire format

Source of truth: `src/util/transaction_util.cpp:144-270`, `TransactionUtil::GetTransactionBytes`. Same layout in the reference wallet (`buildTx`).

**The wire format did not change in v0.4.0.** The genesis hash is not a field; it is prepended only when the signing digest is computed (§3). No length or offset below is affected by chain binding.

| # | SIZE | FIELD | NOTES |
|---|---|---|---|
| 1 | 4 | `transaction_type` u32le | §6 catalog |
| 2 | 1 | `version` | always `0x01` (the node rejects anything else, `transaction_validator.cpp:77-79`) |
| 3 | 8 | `timestamp` u64le | Unix seconds, > 0 |
| 4 | 36 | `sender` | `00000000` ‖ pubkey(32) for a ZBC signer |
| 5 | 4 or 4+n | `recipient` | 36 bytes for ZBC; `u32le(type)` ‖ payload for others (§1.5); **`02 00 00 00` alone** when there is no recipient |
| 6 | 8 | `fee` u64le | atomic units |
| 7 | 4 | `body_length` u32le | |
| 8 | n | `body` | per-type layout, §6 |
| 9 | 4 or var | `escrow` | **no escrow:** the 4 bytes `02 00 00 00` (written as `i32le(2)`, i.e. the Empty account type). **With escrow:** see §2.1 |
| 10 | 4 | `message_length` u32le | |
| 11 | m | `message` | plain text ≤ 256 bytes for ordinary txs (`include/zoobc/util/input_validator.h:23`) |

`signed_bytes = unsigned_bytes ‖ signature(64)`. That is also what the chain hashes: `tx_hash = SHA3-256(signed_bytes)` (`transaction_util.cpp:436-441`, `:446-458`). `tx_id = int64le(tx_hash[0..8])` (`:461-467`). Neither the tag nor the genesis hash enters `tx_hash`.

## 2.1 Escrow block (field 9 when an escrow is attached)

`src/util/transaction_util.cpp:961-1000`, `GetEscrowBytes`.

| SIZE | FIELD |
|---|---|
| 36 | `approver` account (`00000000` ‖ pubkey for a ZBC approver) |
| 8 | `commission` u64le (atomic ZBC paid to the approver) |
| 8 | `timeout` u64le — **absolute** Unix timestamp; must be > tx timestamp (`transaction_validator.cpp:170-182`) |
| 4 | `instruction_length` u32le |
| var | `instruction` (UTF-8 text) |
| 1 | `multi_party` flag — `0x00` for a normal escrow |
| (only if multi_party = 1) | `cosig_len` u32le ‖ cosig ‖ `escrow_request_id` i64le |

The trailing `multi_party` byte is mandatory. Omitting it, or writing the no-escrow marker with a different value, produces only `"Invalid transaction signature"` on the node. §10 vector 2 pins this byte under the new digest.

## 2.2 Worked example (chain-validated, §10 vector 0)

SendZBC of 1 ZBC to self, fee 0.05 ZBC, timestamp 1700000000, no escrow, no message:

```
01000000                type = 1 (SendZBC)
01                      version
00f1536500000000        timestamp 1700000000
00000000 5e8eb28d…2152  sender   (type 0 + pubkey)
00000000 5e8eb28d…2152  recipient (type 0 + pubkey)
404b4c0000000000        fee 5 000 000
08000000                body length 8
00e1f50500000000        body: amount 100 000 000
02000000                no-escrow marker
00000000                message length 0
```

That is 113 bytes. The signing preimage prepends 38 more:

```
5a42432d5458                            tag "ZBC-TX"           6 bytes
090ab3c7…a61878                         genesis hash          32 bytes
0100000001…00000000                     the 113 bytes above  113 bytes
                                                             ---------
                                                             151 bytes
```

`digest = SHA3-256(preimage) = f37e5340…4644fc`; signature `9a3b044d…03bb07`; `tx_hash = SHA3-256(unsigned ‖ signature) = f00a6692…ce78b0`. A live v0.4.0 node returned HTTP 202 and that exact hash.

# 3 Signing procedure (device side)

```
input : account_index, genesis_hash(32), unsigned_bytes
1. k      = SLIP10(m/44'/883'/account_index')           (§1.3)
2. pub    = Ed25519.pub(k); self = 00000000 ‖ pub
3. parse unsigned_bytes per §2; every length must land exactly at the end
4. require version == 1 and sender == self               (§7.1)
5. decode body per type; render §7.2; wait for approval
6. digest = SHA3-256("ZBC-TX" ‖ genesis_hash ‖ unsigned_bytes)
7. sig    = Ed25519.sign(k, digest)                      -- digest is the message
8. return sig (64), and tx_hash = SHA3-256(unsigned_bytes ‖ sig) (32) for the host to cross-check
```

Use streaming SHA3 so that step 6 does not require the whole transaction in RAM. Feed the 6 tag bytes, then the 32 genesis bytes, then the transaction; the header is 100-ish bytes and Tier 1 bodies are ≤ 36 bytes, but an escrow instruction may be long (§5.3).

## 3.1 The tag

`"ZBC-TX"` is six ASCII bytes, `5a 42 43 2d 54 58`. **No NUL terminator and no length prefix.** A firmware that terminates the string, or writes seven bytes, produces a valid-looking signature that fails on chain with the generic `"Invalid transaction signature"` and no further diagnostic. This is the single most likely implementation error in the whole document, so test it against §10 vector 0 before anything else.

The tag exists to separate transaction digests from everything else a ZooBC key can be asked to sign (§3.3), so no raw-signed blob can double as a transaction digest.

Preimage assembly is `src/util/transaction_util.cpp:471-486` (`TransactionUtil::SigningDigest(bytes, genesis)`); the tag constant is `include/zoobc/util/transaction_util.h:123`. The process-wide overload that reads the node's own genesis is `:488-498`, and it **refuses** rather than falling back to the legacy digest when the chain identity is unknown (`:490-495`).

## 3.2 The genesis hash travels with the transaction

The host supplies the 32 bytes; the device does not look them up. Signing is therefore **chain-agnostic**: one code path serves testnet, devnet, mainnet and any private or parallel ZooBC chain, and no chain table is compiled into the firmware for correctness.

The security consequence must be stated plainly, because it is easy to over-claim:

- Chain binding stops **replay across chains**. A signature made for chain A will not verify on chain B. That is what it was built for and it does that completely.
- It does **not** stop a compromised host from naming the wrong chain. The device signs for whatever hash it is handed.

So a firmware table of known genesis hashes is still wanted, but for one purpose only: **displaying a trustworthy network name**, falling back to "UNKNOWN CHAIN" when the hash is not recognised (§7.2).

**Do not refuse unrecognised chains by default.** Devnet's hash changes at every relaunch, and private or parallel chains are a normal thing on ZooBC, so a hard refusal makes the device useless for integration and for legitimate deployments. If a vendor wants that lock, make it an explicit user setting, off by default.

## 3.3 Version is not a discriminator

The `version` byte stays `0x01`; it was not bumped for chain binding. A v0.4.0 node has exactly one rule and verifies every transaction against the chain-bound digest, rejecting any other version value outright (`transaction_validator.cpp:77-79`). A transaction signed with the legacy v0.3.2 digest is simply `"Invalid transaction signature"`, HTTP 400.

The discriminator a **host** needs lives in `GET /api/v1/node/info` as `signing_version`, not in the transaction (§8.3). A device needs no discriminator at all: it implements §3 and nothing else.

### 3.4 Raw message signing (a separate, dangerous command)

Some ZooBC flows sign bytes **without** the SHA3 prehash, plain `Ed25519.sign(k, bytes)`. Chain binding does **not** apply to them. Neither is a hardware-wallet flow:

- **Proof-of-ownership inside NodeRegistration** is verified raw over a 72-byte message using the **node's** key from `node.conf` (`transaction_validator.cpp:2120`), not a wallet key, so a device is never asked for it.
- **FeeVoteReveal** signs vote-info bytes raw in the CLI (`tools/fee-vote-reveal.cpp:47`); the reveal's `voter_signature` is stored (`src/transaction/transaction_executor.cpp:7224`) and the body is parsed at admission (`transaction_validator.cpp:316-330`). Node-owner governance flow.
- Off-chain "prove you hold this key" (`window.zoobc.signMessage` in the reference signer).

> **NORMATIVE**
>
> **Rule for the device:** a raw-sign command must never sign an input of exactly 32 bytes, and should only sign inputs it can display as text. A 32-byte raw input could be the SHA3 digest of a transaction the user never saw, which turns the message command into blind transaction signing. Refusing length 32 closes that hole with no chain change.

Chain binding does not weaken or replace this rule, and does not add a new length to refuse. The tag and the genesis hash live inside the SHA3 preimage, not inside what Ed25519 sees, so the Ed25519 message for a transaction is still exactly 32 bytes and a raw input of any other length still cannot be one.

# 4 Multisignature (Tier 2, for completeness)

Participants of a multisig account co-sign the **inner** transaction, using the same chain-bound digest as §3: `Ed25519.sign(k, SHA3-256("ZBC-TX" ‖ genesis_hash ‖ inner_unsigned_bytes))`. The inner tx's sender is the multisig account, not the device's own key (`src/transaction/multisignature_service.cpp:617-622`, `SigningDigest` then `VerifySignature`).

One subtlety: the **key under which the chain collects signatures** for a pending transaction stays the bare `SHA3-256(inner_unsigned_bytes)` (`src/transaction/transaction_executor.cpp:5683`, `SignatureInfo.transaction_hash`). It identifies the pending transaction; it is not something anyone signs. Do not confuse it with the digest.

The multi-party escrow co-signer signs the same chain-bound digest of the enclosing transaction (`src/transaction/transaction_executor.cpp:4257`).

A firmware that adds multisig later needs a variant of SIGN_TX whose sender check (§7.1 step 3) is replaced by "show the multisig sender in full and warn". The wrapper transaction (type 5) itself is built by the host.

# 5 Chain rules the device cannot see but the host must respect

## 5.1 Fee (transaction_validator.cpp:2428-2515, CalculateMinimumFee)

`minimum = 2 500 000 × fee_scale / 1 000 000 + persistence_rent(body + message bytes) + escrow_rent(escrow bytes × lock duration)`. Live testnet: a plain SendZBC needs 2 500 003 atomic (0.02500003 ZBC). Over-payment is refunded for most types (honest-fee rule), so the host may add a small buffer. Type 0 (Empty) must declare a fee > 0. The device shows the fee as given; the host obtains it from `GET /api/v1/blockchain/estimate-fee` (§9). Unchanged in v0.4.0.

## 5.2 Timestamp

The device has no clock; the host supplies `timestamp`. The node requires `> 0`; the mempool keeps a tx for 60 min; a block may include it only while `block.timestamp ≤ tx.timestamp + 3600` (`src/core/block_validator.cpp:456`). A wrong timestamp costs a resubmission, never funds. The `estimate-fee` response carries the node's own clock, which is a good source.

## 5.3 Sizes

Body ≤ 65 536 bytes; plain message ≤ 256 bytes; dataset value ≤ 4096; escrow instruction is outside the body and only bounded by the node's request size (~128 KB).

**Recommended firmware caps.** Unsigned transaction ≤ 2 KB, message ≤ 256 B, escrow instruction ≤ 512 B, refusing anything larger with `TOO_LARGE`. Tier 1 transactions are under 300 bytes. Note that an ApprovalEscrow arrives with a second transaction attached for verification (§7.3), so the cap must cover both buffers. Larger payloads (DFS, storage) belong in software wallets.

## 5.4 Replay

There is no nonce. Three mechanisms bound a signature instead:

1. **Chain binding.** The genesis hash is inside the digest, so a signature is valid on exactly one chain (§3), and cannot be replayed onto another.
2. **Hash uniqueness.** A transaction is admitted at most once; `POST /api/v1/transactions` answers `duplicate: true` rather than an error when the node already holds it.
3. **The 3600 s inclusion window.** A block may include the transaction only within an hour of its timestamp.

So a signature is usable **once**, on **one chain**, and **within an hour**. Because there is still no ordering primitive, two transactions signed from the same account in the same second race; sign, broadcast and confirm before signing the next one.

## 5.5 Confirmed reads

Every account read the node serves is as of tip − 3 (`api_server.cpp:1454-1456`), so a balance can lag ~45 s behind a fresh confirmation.

## 5.6 What chain binding does not cover

Chain binding applies to **every one of the 56 transaction types** and to **every sender account type**: the node computes one digest and hands the same 32 bytes to whichever verifier the sender's type selects (`src/crypto/signature.cpp:416-470`).

The exception, which was also outside the v0.3.2 digest, is **foreign carrier transactions**, where the `message` field carries a natively signed Ethereum, Solana or Bitcoin transaction and the node verifies that native payload instead of the ZooBC bytes (`transaction_validator.cpp:1692-1738` MetaMask, `:1739-1783` Phantom, `:1784-1853` Bitcoin P2WPKH/P2PKH). Those are MetaMask, Phantom and Bitcoin-wallet flows, never a ZBC hardware-wallet flow. Block signatures, receipts and peer messages are not transactions and are out of scope entirely.

# 6 Transaction type catalog and tiering

Type id = `group + 256 × subtype` (`include/zoobc/common/types.h:19-90`). Fifty-six types exist. Body layouts below are verified against the node's parsers; "see tool" means the exact builder is `tools/<name>.cpp` in the node repo.

## 6.1 Tier 1 — recommended for the standard firmware STANDARD FIRMWARE

These three cover: pay ZBC (with the optional message and escrow envelope), pay any token (genesis coins and user tokens), and complete an escrow. Bodies are small and fixed.

| ID | NAME | BODY | RECIPIENT |
|---|---|---|---|
| 1 | **SendZBC** | `amount u64le` (8) | required, any type |
| 11 | **TransferToken** | `token_id i64le` (8) ‖ `amount i64le` (8) ‖ optional `fee_in_token u8` (17th byte, `0x01` = pay the fee in this token) | required |
| 4 | **ApprovalEscrow** | `approval u32le` (0 = approve, 1 = reject) ‖ **`escrowed_transaction_hash` (32)** = **36 bytes** | empty marker; sender is the approver |

Notes for the device:

- `token_id` is a signed 64-bit value derived from a hash; it can be negative. Genesis coins are ids 1-14 (ZBTC, ZETH, ZUSD, …), ZBC itself is id 0 and never appears in TransferToken. Each token has its own `decimals` (0-8) that the device cannot verify, so the recommended treatment is to accept the host's value as a display hint and label it as reported by the app; see §7.2.
- **ApprovalEscrow changed in v0.4.0.** The body was `approval` ‖ 8-byte id (12 bytes); it is now `approval` ‖ the escrowed transaction's 32-byte hash (36 bytes). The parser refuses any other length (`src/util/transaction_util.cpp:845-860`), the node derives the escrow's id from the hash, and it requires the stored transaction's hash to equal all 32 bytes (`transaction_validator.cpp:936-951`, `transaction_executor.cpp:5522-5533`).

The reason is a hardware-wallet one. A signer that sees only an 8-byte id cannot check what it is releasing, and an 8-byte id is findable by a 2^64 search against a fake escrow shown to the device. The full hash lets the device verify what it approves; see §7.3.

## 6.2 Tier 2 — small fixed bodies, cheap to add CHEAP TO ADD

| ID | NAME | BODY |
|---|---|---|
| 0 | Empty (message carrier / chat) | no body; payload is the `message` field; fee must be > 0 |
| 3 | SetupAccountDataset | `prop_len u32le` ‖ prop ‖ `val_len u32le` ‖ val; recipient = the account the property is set on |
| 259 | RemoveAccountDataset | same layout as 3 |
| 6 | LiquidPayment | `amount u64le` ‖ `complete_minutes u64le` |
| 262 | LiquidPaymentStop | `tx_id i64le` |
| 25 / 27 / 28 | JoinApp / ResignApp / ClaimAppTimeout | `app_id i64le` |
| 26 | AppMove | `app_id i64le` ‖ `move_len u16le` ‖ move |
| 52 | FundLongevity | `target_tx_id i64le` ‖ `amount u64le` |
| 53 | CancelLongevity | `target_tx_id i64le` |
| 260 | EscrowRequest | `proposed_sender`(36) ‖ `amount`(8) ‖ `approver`(36) ‖ `commission`(8) ‖ `timeout`(8) ‖ `instr_len`(4) ‖ instr ‖ `expiry`(8) |

### 6.3 Tier 3 — everything else BLIND-SIGN / EXPERT

| ID | NAME | BODY / NOTE |
|---|---|---|
| 2 / 258 / 514 / 770 | NodeRegistration / Update / Remove / Claim | node_pubkey(32) [\|\| account(36)] [\|\| locked_balance(8)] [\|\| POOWN(136)]; POOWN is verified with the node's own key (§3.4); node-owner flow |
| 5 | MultiSignature | wrapper: multisig info + signature info + inner tx; §4 |
| 7 / 263 | FeeVoteCommitment / Reveal | governance; node-owner flow |
| 8 / 264 / 520 | DFSCreateFile / Update / Delete | path_len \|\| path \|\| content_len \|\| content; up to 64 KB |
| 9 | AddPrepaidStorage | see tool |
| 10 / 12 / 13 / 14 | IssueToken / MintToken / BurnToken / FinanceToken | see tools |
| 15 / 16 / 17 | CreateTrigger / CancelTrigger / AttestEvent | see tools |
| 18 / 19 / 20 | CreateSwapOffer / Accept / Cancel | 18: give_token(8) give_amount(8) want_token(8) want_amount(8) expiry(8) [counterparty(36)] |
| 21 / 22 / 23 | CreateMarket / PlaceOrder / CancelOrder | order book; see tools |
| 24 | CreateApp | app_type(1) \|\| stake_token_id(8) \|\| stake_amount(8) \|\| seats(1) \|\| params_len(2) \|\| params \|\| … |
| 29 / 30 / 31 | ScheduledTransfer / CancelSchedule / ReassignSchedule | see tools |
| 32-35 | RegisterRelease / AuthorityPropose / AuthorityAccept / RevokeRelease | release governance |
| 36-38, 46-49 | Register/Heartbeat/Unregister Gateway, Archival, Relay | infrastructure registry |
| 39 | SettleApp | app_id(8) \|\| final_seq(4) \|\| move_count(4) \|\| [seat(1) move_len(2) move sig(64)]* |
| 40 / 41 | StoreFile / SubmitStorageProof | storage |
| 42-45 | TransferDataset / SetDatasetPolicy / AcceptDataset / DeleteDataset | dataset objects |
| 50 | SetTransactPolicy | 9-bit opt-out mask |
| 51 | SetConsensusParam | node-signed governance |

**Handling Tier 3 on a standard firmware.** The envelope parser (§2) is shared by all types, so the device can always show: type id and name, sender, recipient, fee, message, and `SHA3-256(body)` as a fingerprint. Recommendation: refuse Tier 3 by default and allow it only behind an explicit "expert / blind signing" setting, with the fingerprint shown, which is how other chains' apps handle unknown contract calls. A 56-entry name table costs about 1 KB.

# 7 What the device verifies and shows

## 7.1 Verification ("is this transaction correct, and is it really from me?")

1. **Structure.** Parse per §2. Each length prefix must be consistent and the buffer must end exactly after the message. Anything else: refuse.
2. **Version** must be 1.
3. **Sender must equal the device's own derived account** for the requested path. The device never trusts the sender bytes it was handed; it re-derives and compares all 36 bytes. Mismatch: refuse. This is the whole "where is it from" guarantee: the signature only proves control of the key, so the bytes must name that key.
4. **Type** must be in the supported set (or the expert mode is on, §6.3).
5. **Body length** must match the type's expected layout: **8** for SendZBC, **16 or 17** for TransferToken, **36** for ApprovalEscrow. Extra bytes: refuse (the node would accept a longer SendZBC body and ignore the tail, which is exactly how a hidden payload would ride along).
6. **Escrow timeout** must be greater than the tx timestamp (the node rejects otherwise).
7. **Genesis hash** must be exactly 32 bytes. The device does not validate it against a list (§3.2), but a wrong length is a malformed request.
8. Never accept a **pre-computed digest** to sign. The device hashes the bytes itself, tag and genesis included (§3.4).

## 7.2 Display (what the user approves)

Per screen, in this order, adapted from the chain-validated reference signer (`zoobc-signer/extension/lib/decode.js`):

| ROW | RULE |
|---|---|
| Network | name from the firmware's table for the supplied genesis hash; **"UNKNOWN CHAIN" plus the first 8 hex** if it is not in the table (§3.2) |
| Type | e.g. "Send ZBC", "Send token", "Approve escrow" |
| Amount | `amount / 1e8` with the ZBC symbol; for tokens see below |
| Recipient | **full** address, never elided. A middle-elided address is what a vanity-generated look-alike key defeats. Type 0 as `ZBC_...`, others per §1.5 or `<type> hex` |
| Fee | atomic / 1e8 ZBC |
| Total leaving the account | amount + fee, one number (only for ZBC amounts; a token amount and a ZBC fee are different units and must not be summed) |
| Message | shown if printable; otherwise "N bytes (binary)" + first 8 hex of SHA3-256 |
| Escrow (if present) | approver in full, commission, timeout as a date/time, instruction text |
| Token (type 11) | token id (signed decimal) and the **atomic** amount. If the host supplies `decimals/symbol` as display hints, show the scaled amount labelled "as reported by the app", because the device cannot verify them. |
| Escrow approval (type 4) | "APPROVE" or "REJECT" in large type, then the verified escrow detail per §7.3, then fee |

Timestamps need not be shown; they are not user-meaningful and the node enforces the window.

## 7.3 Verifying an escrow approval

New in v0.4.0, and the reason the ApprovalEscrow body grew to 36 bytes. The device can now show what it is releasing instead of an opaque id.

The host hands the device the **original escrow transaction** alongside the approval: its unsigned bytes and its signature. The device then:

1. computes `SHA3-256(escrow_unsigned_bytes ‖ escrow_signature)`;
2. requires it to equal the 32 bytes in the ApprovalEscrow body, byte for byte;
3. parses the escrow transaction per §2 and requires the **approver** inside its escrow block to equal the device's own account;
4. displays sender, recipient, amount, commission, timeout and instruction **from those verified bytes**.

If the hash does not match, refuse. A host that supplies a different transaction than the one being approved is caught at step 2, which is precisely the attack the 8-byte id could not stop.

This check is chain-agnostic: it hashes bytes the device was handed and compares them with bytes in the body. No chain state and no genesis hash is involved.

The host gets the hash to put in the body from `GET /api/v1/accounts/<a>/escrows`, which returns `transaction_hash` per escrow (§9.1).

# 8 Host ↔ device protocol (the "conversation")

The transport is the vendor's own USB/HID layer and its existing chunking; the content is what matters.

## 8.1 Commands

| COMMAND | REQUEST | RESPONSE |
|---|---|---|
| `GET_APP_VERSION` | — | firmware/app version, supported type set, max tx size |
| `GET_PUBLIC_KEY` | `account_index`, `confirm_on_device` (bool) | `pubkey(32)`, `address` (66-char string); when `confirm_on_device`, the device shows the full address for the user to compare with the website |
| `SIGN_TX` | `account_index`, **`genesis_hash` (32 bytes, mandatory)**, `unsigned_tx_bytes`, optional escrow-approval material (§7.3), optional display hints `{token_decimals, token_symbol}` | `signature(64)`, `tx_hash(32)`, or a typed refusal (`SENDER_MISMATCH`, `UNSUPPORTED_TYPE`, `MALFORMED`, `USER_REJECTED`, `TOO_LARGE`, `ESCROW_HASH_MISMATCH`) |
| `SIGN_MESSAGE` | `account_index`, `message_bytes` | `signature(64)` (raw Ed25519, §3.4 rules) |

**`genesis_hash` is now load-bearing, not a display hint.** In draft 1 the `network` parameter could be an enum, because it only chose a label. It is now inside the digest, so it must be the actual 32 bytes and a wrong value produces a signature that no chain will accept.

**Structured fields vs raw bytes.** The reference browser signer accepts only structured fields from the page and serializes them itself, so it can never be tricked into signing bytes it did not build. For a hardware device the equivalent is: the host sends the unsigned bytes and the device **parses them completely** and refuses anything it cannot account for (§7.1). Both designs give the same guarantee; parsing on the device is simpler for a firmware because the format is fixed-header + length-prefixed sections. Either way the device must compute the digest itself.

## 8.2 Sequence: website → companion app → device → chain


The host compares the node's `transaction_hash` with the device's `tx_hash`; they must be identical, otherwise the host altered the bytes after signing.

**Read `genesis_hash` and `signing_version` from the endpoint you will broadcast to**, since that is the chain that judges the signature.

### 8.3 Chain identity

The only network identifier ZooBC has is the **genesis block hash** (nodes compare it before peering; there is no network-id byte). It is now also what a signature is bound to (§3).

`GET /api/v1/node/info` returns it as `genesis_hash`, alongside two fields new in v0.4.0:

```
"genesis_hash": "090ab3c7…", "signing_version": 2, "signing_tag": "ZBC-TX"
```

`signing_version` absent means the node predates v0.4.0 and enforces the legacy unbound digest. A host that supports only current chains can treat its absence as "do not sign here".

The node records `block 0 . block_hash` at start-up (`src/main.cpp:712-717`, `include/zoobc/common/chain_identity.h`) and `/node/info` prints those same bytes (`api_server.cpp:4663-4666`, `json_serializer.cpp:295`, signing fields at `:296-300`). A node prints lower-case hex and the archival API upper-case, so decode case-insensitively and strip a leading `0x` if present; the bytes are the same either way.

**Where to read it from.** A gateway answers `/api/v1/node/info` from the archival service rather than from a node. Archival passes the two signing fields through from the node it mirrors, emitting them exactly when the node reports them (`archival/api_server.cpp:213-220`, learned at `:1864-1870`), from build `7ad161a4` onward. Read `signing_version` from a node endpoint, or from a gateway on that build or later; treat its absence as "ask a node" rather than as an answer.

# 9 Gateway / node API used by the host

All routes are `GET` unless stated, JSON responses, served by a node (`:8080`), a gateway (HTTPS front for a node, same paths) or an archival node (full history, read-only). Reads: archival → gateway → node. **Broadcast: gateway or node, never an archival endpoint.** Full list in `docs/API_REFERENCE.md` and `src/api/api_server.cpp`.

## 9.1 Reads

| ROUTE | PURPOSE | KEY FIELDS |
|---|---|---|
| `/api/v1/node/info` | chain identity and health | `genesis_hash`, **`signing_version`**, **`signing_tag`**, `height`, `version`, `synced`, `archival_node`, `wedge_reason` |
| `/api/v1/accounts/<ZBC_addr or 72-hex>` | balances (confirmed, tip−3) | `spendable_balance`, `balance`, `locked_balance`, `total_balance`, `pop_revenue`, `transact_policy` (atomic units) |
| `/api/v1/accounts/<a>/tokens` | token balances | `tokens[]` with per-token `decimals` (ZUSD 2, ZBTC 8, ZETH 9 for the bridge coins as configured; always read it, never assume 8) |
| `/api/v1/tokens/<id>` | token metadata | symbol, decimals, supply |
| `/api/v1/blockchain/fee-params` | fee constants | `min_fee`, `fee_scale`, `effective_min_fee`, `one_zbc`, storage rent params |
| `/api/v1/blockchain/estimate-fee?type=&body_length=&message_length=&escrow=&escrow_instruction_length=&escrow_lock_seconds=` | exact minimum fee for a proposed tx | `minimum_fee`, `one_zbc`, `timestamp` (node clock) |
| `/api/v1/accounts/<a>/escrows` | pending escrows to approve | **`transaction_hash`** (the 32 bytes for an ApprovalEscrow body, §7.3), amounts, counterparties (`api_server.cpp:2761`, field near `:2812`) |
| `/api/v1/accounts/<a>/history?limit=` | sent/received txs (node only; archival 404s it) | `transactions[]` |
| `/api/v1/accounts/<a>/movements?limit=` | value movements incl. non-tx credits | `movements[]` |
| `/api/v1/accounts/<a>/pubkey` | verified public key for any account type | `public_key`, `curve`, `verified`, `source` |
| `/api/v1/address/<any>` | parse any native address form into typed hex | hex, type |
| `/api/v1/transactions/<hash-hex>` | look up by **hash** (not numeric id) | tx |
| `/api/v1/transactions/<hash-hex>/status` | lifecycle | `status`: `staging` / `mempool` / `confirmed` (+`block_height`) / `not_found` (HTTP 404) |

## 9.2 Broadcast: POST /api/v1/transactions

Body (`api_server.cpp:764`, fields parsed from `:856`). **Unchanged in v0.4.0: no new field.** `transaction_body_bytes` is the **body only**; the node rebuilds the envelope from the named fields, prepends its **own** genesis hash and the tag, and recomputes the digest, so every field must equal what was signed.

```
{
  "version": 1,
  "timestamp": 1700000000,
  "transaction_type": 1,
  "fee": 5000000,
  "sender_account_address":    "000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152",
  "recipient_account_address": "000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152",
  "transaction_body_bytes": "00e1f50500000000",
  "signature": "9a3b044dca7c2f351e9d2b37dd6924e4260536c496207de87f0e1f1a31dd9f01cd6359e202e13964580a561a89
e26688749fe9cdbf12056ed05eb1fbcd03bb07",
  "message_hex": "…optional…",
  "escrow": { "approver_address": "…72 hex…", "commission": 0, "timeout": 1700003600, "instruction": "…" }
}
```

- `fee`, `commission`, `timeout` are JSON numbers (int64). `message` (text) is accepted instead of `message_hex`. Omit optional keys rather than sending them empty. Recipient for "none" is the hex `02000000`.
- Success: HTTP **202** `{"status":"success","duplicate":false,"message":"Transaction submitted successfully","transaction_hash":"…"}`. `duplicate:true` means the node already holds it (do not resend). 200 must be treated as success too.
- Failure: 400 with `{"success":false,"error":"…","code":400}` (signature, fee floor, structure), 503 under load (`"mempool backpressure"`, etc.). The staging pool checks the signature first (`transaction_staging_pool.cpp:147`); a tx that reaches a later error such as `"Sender account does not exist"` has already passed signature verification, which is a useful integration test on an unfunded key.

## 9.3 Diagnosing a rejected signature

A chain mismatch is **not** distinguishable from any other signature failure today. Both of these return HTTP 400 with the identical body:

```
{"success":false,"error":"Transaction validation failed: ValidationError: Invalid transaction
signature","code":400}
```

- a signature made with the legacy v0.3.2 digest;
- a signature made for a different chain's genesis hash.

So when a signature is rejected, work through the causes in this order, most likely first: the tag omitted or NUL-terminated (§3.1); Keccak-256 used instead of SHA3-256 (§1.2); a genesis hash that does not match the chain being broadcast to; the escrow's trailing `multi_party` byte omitted (§2.1); an ApprovalEscrow body still 12 bytes rather than 36 (§6.1).

## 9.4 Endpoints today

| CHAIN | GATEWAY (HTTPS) | NODE API | SIGNING | NOTE |
|---|---|---|---|---|
| testnet | `https://zoobc.network` | behind the gateway | v0.3.2 legacy, **moving to v0.4.0 imminently** | the integration target once it flips |
| devnet | `https://socialconnect.network` | `http://193.181.213.106:8080` | v0.4.0 | 11 nodes, faucet at `/faucet`; genesis hash changes at every relaunch |
| reference chain R | — | local | v0.4.0 | the chain the §10 vectors were signed for and accepted by; genesis config ships in `tests/vectors/signing_v2/` |
| mainnet | — | — | v0.4.0 or later | not yet launched |

**Read the genesis hash at runtime; do not hard-code it.** Devnet's changes at every relaunch, and testnet's will be whatever the relaunch fixes it at. A firmware table of hashes exists only to put a name on the screen (§3.2), so an entry the firmware does not recognise shows "UNKNOWN CHAIN" and signing is unaffected.

# 10 Test vectors (chain-validated)

Signed for **reference chain R**, a single v0.4.0 node whose genesis config ships beside the vector set in `tests/vectors/signing_v2/`, so anyone with the node's `zbc-genesis` tool can rebuild block 0 and reproduce the hash. The generator takes any genesis hash, so the same seven cases can be regenerated for whichever chain you are testing against (§10, last paragraph).

```
genesis_hash  090ab3c7e0ce0630f9b6a3d9694dc4d2d5e1ccd018352ef27de6cd8de0a61878
tag           "ZBC-TX" = 5a42432d5458
preimage      tag ‖ genesis_hash ‖ unsigned_bytes          (6 + 32 + n bytes)
digest        SHA3-256(preimage)
signature     Ed25519.sign(sk, digest)
tx_hash       SHA3-256(unsigned_bytes ‖ signature)
```

Mnemonic (BIP-39 test phrase, holds nothing): `abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon about`, no passphrase.

Account 0 (`m/44'/883'/0'`):

```
seed     51bae95d58f32304a5c9d894819989a4fb04da6115d9a43612a026e7a5dd5d96
pubkey   5e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152
account  000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152
address  ZBC_L2HLFDOM_VKKKTEXX_C2P2M6LG_EB6ZNKSV_356SJUWW_5QPVHDE7_EFJA3PEX
```

Account 1 (`m/44'/883'/1'`):

```
seed     2c0f8a5722d5701a7a4e7d61219c5235a9b74ba46fdbaff9f7978ec7bc14f85d
pubkey   fd4472bbdec43b919aa37b9b3366a3318f2902a753a15f5fd6d43c44b5121b67
address  ZBC_7VCHFO66_YQ5ZDGVD_PONTGZVD_GGHSSAVH_KOQV6X6W_2Q6EJNIS_DNTVIFT4
```

Each vector lists the unsigned bytes, the digest, the signature and the tx hash. The preimage is always `5a42432d5458 ‖ 090ab3c7…a61878 ‖ unsigned`, so it is not repeated. The **legacy** line is the v0.3.2 signature for the same bytes, printed only so an implementer can confirm they are *not* producing it; it is rejected by every v0.4.0 chain.

| # | CASE | TYPE | TS | FEE | BODY | RECIPIENT | EXTRA | ACCT |
|---|---|---|---|---|---|---|---|---|
| 0 | send, no escrow, no message | 1 | 1700000000 | 5000000 | `00e1f50500000000` | self | — | 0 |
| 1 | send with message "hello zoobc" | 1 | 1700000001 | 5000000 | `00e1f50500000000` | self | message | 0 |
| 2 | send with escrow (approver self, commission 1000, timeout 720, "release on delivery") | 1 | 1700000002 | 5000000 | `00e1f50500000000` | self | escrow | 0 |
| 3 | dataset, empty recipient | 3 | 1700000003 | 1000000 | `deadbeef` | none (`02000000`) | — | 0 |
| 4 | account index 1 | 1 | 1700000004 | 5000000 | `00e1f50500000000` | self (acct 1) | — | 1 |
| 5 | SettleApp (type 39), empty recipient | 39 | 1700000005 | 5000000 | `0102030405060708` | none | — | 0 |
| 6 | fee above 2^53 (9007199254740995) | 1 | 1700000006 | 9007199254740995 | `00e1f50500000000` | self | — | 0 |

### Vector 0 — send, no escrow, no message

```
unsigned
010000000100f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215200000000
5e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152404b4c0000000000800000000e1f5050000000002
00000000000000
digest     f37e53402bd62fd5b7f2fe39eea7129eaedb929c0038a132aeefeaad774644fc
signature
9a3b044dca7c2f351e9d2b37dd6924e4260536c496207de87f0e1f1a31dd9f01cd6359e202e13964580a561a89e26688749fe9cdbf
12056ed05eb1fbcd03bb07
tx_hash    f00a6692eccb249e0e1cf70f1215502dcc82df6b4d47799efe3dc13d53ce78b0
legacy
cd6b6723d610833615b13b2db318cdbafc0e1d1fc97aaff39f9f9e985532167d2535292e1b5ca8a791b538a53a4d195fc8023694e6
06b78aa0d6e697aaa20d0a
live       HTTP 202 admitted; the node returned transaction_hash equal to tx_hash above
```

### Vector 1 — send with message "hello zoobc"

```
unsigned
010000000101f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215200000000
5e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152404b4c0000000000800000000e1f5050000000002
0000000b00000068656c6c6f207a6f6f6263
digest     c6b21e267818b3b026e8a7690324aa8c292042d7f569937487396d2512fcf4b1
signature
8d90ceacf8318cbbb6f544915ee084232af9e3222ac07ebd7c4db124df2969bba3ac9d0414e242b6a33962135ee1544a300485e407
44dd0436af1c37e8a33e01
tx_hash    511d27b8940291f6072a2009734f833655997c0eb9c2d365828d4c2908efcaf4
legacy
e6f1b4279432b9874df7ec895a5b84fe8b4c3655d90579db79a47acb9281636f14a31217c5d297fd084c64fd3080318de3b2e6ef0a
4b652afafa6f806194d309
live       HTTP 202 admitted; the node returned transaction_hash equal to tx_hash above
```

### Vector 2 — send with escrow

Pins the escrow block including its trailing `multi_party` byte, under the new digest.

```
unsigned
010000000102f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215200000000
5e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152404b4c0000000000800000000e1f5050000000000
0000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f2152e803000000000000d0020000000000001300
000072656c65617365206f6e2064656c69766572790000000000
digest     ac6577cbe68644d4bf482800746f954908826ada1186ef78db2ba9e851f7a588
signature
c6f926a1c558ca3e81d5e847a0d6df521ef39257fdcc2c4ad5e26ec44fdfacba92ad597a1482f83efe046f75bded03ba12c69ae247
2721eeb11f29b934c94302
tx_hash    3691d78e5d45c746d82d2327c43fe39762c143b2d3724ed4105d709f9e95b668
legacy
e74cce3b9cbb4541a16f5d02704b3762a2ada4739b9461036fa64ca0d993f257b147c1e4719f53399885719c1c40ac8233a7d37012
9583f3b6d20602192ce30b
live       signature verified, then refused: escrow timeout 720 is in the past. Byte-level vector only.
```

### Vector 3 — dataset, empty recipient

```
unsigned
030000000103f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215202000000
40420f0000000000004000000deadbeef0200000000000000
digest     06d45034ea2d9bd1cb0e9dc985e46c2ec7e137c34f3ccf11d6a7f4ec3dff0627
signature
6b5cce174ece3fb9b49f83ebc165455385c19cfdd12289d23f628a691488d3c2b30abc2ca5fc8e22897f67e7a3a08976fc12488009
15ec9c5052b688ae6a870d
tx_hash    06a8d9034c51fa6dcdfd4510e2e0df8d06412648b2186d9f269e8cc2e422f9fe
legacy
fbc74a54bc31fcc45a5bf1cef88e079eb3f3662386670d3aee9923948a0163018eec62198e78d94224e40bfc3f20931e4baf0c1f7c
33f72fe3fbe3f3bda47302
live       byte-level vector only (the body `deadbeef` is not a valid dataset payload)
```

### Vector 4 — account index 1

```
unsigned
010000000104f153650000000000000000fd4472bbdec43b919aa37b9b3366a3318f2902a753a15f5fd6d43c44b5121b6700000000
fd4472bbdec43b919aa37b9b3366a3318f2902a753a15f5fd6d43c44b5121b67404b4c0000000000800000000e1f5050000000002
00000000000000
digest     0b133ec600ab9825ef36b597dffdac90e05eeb524627653dc9da87734f9957dd
signature
5ec0df16d350faf7151e22d147e9600f10d836bf9d6b9911965991dc43f1ce86bfdc8be81cbb0a86f8e17160d1f420cf7130ec8349
b48100c0711adb4b75150c
tx_hash    35545ce6d6132ca7aa651de3af801574e3f6e89eaa01a90e860ea958ebc16cad
legacy
483a738f6a85413c6c6b6c7b18e4cabfbde3d37e5057bc9042ef3542d5533ce58940f0367d2014651289062ada446db91a097150c0
01d8cf6da76b3e4b9a5601
live       signature verified, then `Sender account does not exist` (account 1 is unfunded on R)
```

### Vector 5 — SettleApp (type 39), empty recipient

```
unsigned
270000000105f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215202000000
404b4c0000000000080000000102030405060708020000000000000
digest     15b9229f037e26041854c6a6a1542704573d6ba51985f25540bf85bc20ab3eef
signature
ae6e890cacd4833e4c017b2aba7c57e7de6e4f2c7a3a36f0a508628a10e5c7f0d8fb504a9ae4b56c3d5e8c9602f7f40dbec462fd6f
2814efdf21e521481a440b
tx_hash    9aef0f065455016d391b2c7673dbfba483001a56099541091ae5e3f94e364341
legacy
d8a262a1cb2ab5eb26d33110754cf9056dd4ab0b057c3769a67a09eb3efe8b8a2a049140e73567118e71cf978883ea2e6c4665eb31
54592fd4f1058353cce007
live       byte-level vector only
```

### Vector 6 — fee above 2^53

Pins 64-bit fee handling; a JSON parser that rounds through a double will fail here.

```
unsigned
010000000106f153650000000000000005e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f215200000000
5e8eb28dccaa94a992f7169fa67966207d96aa55df7d24d2d6ec1f538c9f21520300000000000200008000000000e1f5050000000002
00000000000000
digest     ef909cb518fe4aac388a15c9f4715c163df7555de22014b9f5d8ea4da1d66434
signature
83328dcc9bcfc34c82e463c5f031f998e66da6b242329b06f3f6c1cde78c57710d161ff7a44981605f8a7418d3f3b680a22c6b1034
2722b2f44143b0d76c380c
tx_hash    c080d7c5ad9586003a701d1c74c51eef0dd10ec82850f7a79fe3fc260b3f4b12
legacy
4c610340d4149f2cc055b0794e0dc10e885ce6456b3a27d3ee2fd8daa8e13854889d76d9f6bfc1a4ae0da8037f0c07bf379df811fd
2300aca4d71f9dc8138700
live       byte-level vector only
```

### Negative vectors

Both use vector 0's unsigned bytes, and both return HTTP 400 with the identical generic error (§9.3):

- vector 0 carrying the **legacy** v0.3.2 signature;
- vector 0 signed for a **different genesis** (`5a` × 32).

The construction is pinned three ways: the node's own test (`tests/chain_bound_signature_test.cpp`, genesis `5a`×32, digest `55f3ed31e5896bb127dfa7db19f3eb1d6e35a8e49e29fcbdca24acd84c9ab955`), the signer extension's `npm test` over the same values, and the live reference-chain run above. The set and its generator (`generate.py <genesis_hash>`) are in the repo under `tests/vectors/signing_v2/`, so the same seven cases can be regenerated for any chain by passing its genesis hash.

# 11 Reference implementation (Python, standard library + cryptography)

Reproduces vector 0 end to end; every `match` line prints `True`. This is the shortest complete statement of the algorithm and doubles as a firmware test oracle.

```python
import hashlib, hmac, struct, unicodedata, base64
from cryptography.hazmat.primitives.asymmetric import ed25519
from cryptography.hazmat.primitives import serialization

MNEMONIC = "abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon abandon about"
TAG      = b"ZBC-TX"                                          # 5a 42 43 2d 54 58, no terminator
GENESIS  = bytes.fromhex("090ab3c7e0ce0630f9b6a3d9694dc4d2d5e1ccd018352ef27de6cd8de0a61878")

def slip10_zbc(mnemonic, account, passphrase=""):
    seed = hashlib.pbkdf2_hmac("sha512", unicodedata.normalize("NFKD", mnemonic).encode(),
                               unicodedata.normalize("NFKD", "mnemonic" + passphrase).encode(), 2048, 64)
    I = hmac.new(b"ed25519 seed", seed, hashlib.sha512).digest(); k, c = I[:32], I[32:]
    for idx in (44, 883, account):                              # all hardened
        I = hmac.new(c, b"\x00" + k + struct.pack(">I", idx + 0x80000000), hashlib.sha512).digest()
        k, c = I[:32], I[32:]
    return k                                                    # 32-byte Ed25519 seed

def zbc_address(pub, prefix=b"ZBC"):
    buf = bytearray(pub + prefix); buf[32:35] = hashlib.sha3_256(bytes(buf)).digest()[:3]
    s = base64.b32encode(bytes(buf)).decode().rstrip("=")           # 56 chars
    return prefix.decode() + "".join("_" + s[i*8:(i+1)*8] for i in range(7))

def unsigned_tx(tx_type, ts, sender, recipient, fee, body, escrow=None, message=b""):
    out = struct.pack("<I", tx_type) + b"\x01" + struct.pack("<Q", ts) + sender + recipient
    out += struct.pack("<Q", fee) + struct.pack("<I", len(body)) + body
    if escrow:
        instr = escrow["instruction"].encode()
        out += escrow["approver"] + struct.pack("<QQ", escrow["commission"], escrow["timeout"])
        out += struct.pack("<I", len(instr)) + instr + b"\x00"       # multi_party = 0
    else:
        out += struct.pack("<i", 2)                                  # no-escrow marker
    return out + struct.pack("<I", len(message)) + message

def signing_digest(unsigned, genesis):
    return hashlib.sha3_256(TAG + genesis + unsigned).digest()       # the whole change, in one line

k  = slip10_zbc(MNEMONIC, 0)
sk = ed25519.Ed25519PrivateKey.from_private_bytes(k)
pub = sk.public_key().public_bytes(serialization.Encoding.Raw, serialization.PublicFormat.Raw)
acct = b"\x00\x00\x00\x00" + pub
print("address :", zbc_address(pub))
tx  = unsigned_tx(1, 1700000000, acct, acct, 5_000_000, struct.pack("<Q", 100_000_000))
dig = signing_digest(tx, GENESIS)
sig = sk.sign(dig)                                                   # digest IS the message
print("digest  :", dig.hex())
print("sig     :", sig.hex())
print("tx_hash :", hashlib.sha3_256(tx + sig).hexdigest())           # genesis is NOT in tx_hash
print("match   :", sig.hex() == "9a3b044dca7c2f351e9d2b37dd6924e4260536c496207de87f0e1f1a31dd9f01"
                            + "cd6359e202e13964580a561a89e26688749fe9cdbf12056ed05eb1fbcd03bb07")
```

A second reference, in JavaScript and runnable in Node ≥ 18 with no build step, is `zoobc-signer/extension/lib/zoobc-crypto.js` (functions `deriveZbcSeed`, `zbcAddress`, `buildTx`, `accountFromSeed`, `broadcastPayload`); `npm test` there checks all seven vectors.

# 12 Source references (ZooBC node repo, zoobc-main at 7ad161a4, v0.4.0)

## 12.1 Chain binding

- `include/zoobc/util/transaction_util.h:123` — the `"ZBC-TX"` tag constant.
- `src/util/transaction_util.cpp:471-486` — `SigningDigest(bytes, genesis)`, preimage assembly and SHA3; `:488-498` — the process-wide overload, which refuses at `:490-495` rather than falling back when the chain identity is unknown.
- `include/zoobc/common/chain_identity.h`, `src/common/chain_identity.cpp` — chain identity; set at start-up in `src/main.cpp:712-717`.
- `src/transaction/transaction_validator.cpp:1854-1873` — verification: `GetTransactionBytes` at `:1856`, `SigningDigest` at `:1865`, `VerifySignature` at `:1873`.
- `src/crypto/signature.cpp:416` — algorithm dispatch by sender account type; `:68` — the Ed25519 path.
- `src/transaction/transaction_staging_pool.cpp:147` — the signature is the first check on arrival.
- `src/transaction/multisignature_service.cpp:617-622` — multisig participant verification; `src/transaction/transaction_executor.cpp:5683` — the collection key (unsigned); `:4257` — the escrow co-signer.
- `tests/chain_bound_signature_test.cpp`, `tests/vectors/signing_v2/` — the node's own tests and the vector set.

## 12.2 Everything else

- `include/zoobc/common/types.h:19-90` — transaction type enum; `:199-235` — account types.
- `include/zoobc/util/transaction_util.h:38-58` — account type constants 0-16; `:233-237` — the ApprovalEscrow body.
- `src/util/transaction_util.cpp:144-270` — `GetTransactionBytes`; `:436-441`, `:446-458`, `:461-467` — hash and id; `:466-515` — payload lengths; `:519-548` — signature lengths; `:571-587` — SendZBC body; `:826-860` — ApprovalEscrow body (36 bytes, refuses any other length at `:845-860`); `:961-1000` — escrow bytes; `:1189-1250` — dataset body.
- `src/transaction/transaction_executor.cpp:964-973` — TransferToken body; `:5522-5533` — the escrow hash check on execution.
- `src/transaction/transaction_validator.cpp:60-120` — version/timestamp/address checks, version at `:77-79`; `:170-182` — escrow timeout; `:936-951` — the escrow hash check on admission; `:1692-1853` — foreign carrier paths, outside the digest rule; `:2120` — raw POOWN verification with the node key; `:2428-2515` — minimum fee.
- `src/crypto/slip10.cpp:585-705` — address encode/decode.
- `src/core/block_validator.cpp:456` — 3600 s inclusion window; `include/zoobc/transaction/transaction_pool.h:274-280` — mempool expiry.
- `src/api/api_server.cpp:589-706` — fee endpoints; `:764` — `POST /transactions`, fields from `:856`; `:1155-1220` — status; `:1429-1500` — account; `:2761` — escrows handler, `transaction_hash` near `:2812`; `:4663-4666` — node info genesis hash.
- `src/api/json_serializer.cpp:295` — genesis hash; `:296-300` — `signing_version` and `signing_tag`.
- `archival/api_server.cpp:213-220`, `:1864-1870` — the archival pass-through of the signing fields.
- `include/zoobc/util/input_validator.h:23-25` — message and body caps.
- Reference signer: `zoobc-signer/extension/lib/{zoobc-crypto,txrequest,decode,network,networks}.js`, `test/vectors.json`.