

# Custody Protocol — Transaction Execution & Data Store Specification

This document defines the deterministic transaction execution pipeline and RocksDB-backed state storage model for the Custody Protocol.

It maps directly onto the previously defined type system (Workspace, Policy, Intent, Asset, Attestation).

## 1. Execution Pipeline Overview

The protocol separates transaction handling into three stages:

1. `check_tx` — signature + structural validation (no writes)
2. `deliver_tx` — deterministic state transition (writes allowed)
3. `query` — read-only state access

All writes occur through a RocksDB WriteBatch to preserve atomicity and determinism.

```
Result deliver_tx(const TxEnvelopeV1& tx,
                  StoreTxn& st,
                  const TxContext& ctx) {

    enforce_and_bump_nonce(tx.signer, tx.nonce, st);

    return std::visit([&](auto&& payload) {
        return handle(payload, tx.signer, st, ctx);
    }, tx.payload);
}
```

## 2. Deterministic Storage Model

All state is SCALE-encoded bytes stored in RocksDB.

Each transaction operates over a snapshot + WriteBatch.

Reads are explicit. Writes are batched and committed atomically.

```
template<class T>
std::optional<T> get_t(KV& kv, const Bytes& key);

template<class T>
void put_t(KV& kv, const Bytes& key, const T& value);
```

## 3. Read/Write Sets per Transaction

Each transaction type has an explicit read/write set to preserve determinism.

ProposeIntent:

Reads: ActivePolicyPointer, PolicySet, AssetState, DestinationState

Writes: IntentState

ApproveIntent:

Reads: IntentState, Policy roles

Writes: ApprovalState, updated IntentState

ExecuteIntent:

Reads: IntentState, ApprovalState (prefix scan), AssetState, DestinationState, AttestationRecords

Writes: updated IntentState (status=EXECUTED)

## 4. Attestation Lookup Strategy

Attestations are keyed as:

AT|workspace|subject|claim|issuer

Execution scans the prefix:

AT|workspace|subject|claim|

to validate required compliance evidence.

## 5. Error Codes (Deterministic Failures)

Define canonical error codes:

ERR\_BAD\_SIGNATURE

ERR\_NONCE\_MISMATCH

ERR\_NOT\_AUTHORIZED

ERR\_POLICY\_MISSING

ERR\_THRESHOLD\_NOT\_MET

ERR\_TIMELOCK\_NOT\_SATISFIED

ERR\_ASSET\_DISABLED

ERR\_DESTINATION\_DISABLED

ERR\_ATTESTATION\_MISSING

## 6. Cryptographic Library Requirements

You should integrate well-established cryptographic libraries rather than implementing primitives yourself.

Recommended libraries:

Ed25519:

- libsodium (stable, audited, simple API)

Secp256k1:

- libsecp256k1 (Bitcoin Core implementation, constant-time, audited)

Use these libraries for:

- key generation
- signature creation
- signature verification

All signing should occur over:

`HASH("CUSTODY_TX_V1" || SCALE_ENCODE(envelope_without_signature))`

Private keys must never be stored inside the protocol state layer.

## 7. Recommended Implementation Order

1. RocksDB wrapper + typed get/put
2. Nonce enforcement
3. Workspace/Vault creation
4. Policy creation/activation
5. Asset registry
6. ProposeIntent
7. ApproveIntent
8. Attestation support
9. ExecuteIntent validation logic