SQRL: A New Settlement System for Money

Secure Quantum-Reliable Ledger

Tokenised deposits, instant cross-border payments and in-flight FX, built on Bitcoin and Lightning infrastructure

Dr Barry Childe
Co-founder of Arqit Quantum Inc.

Paul Dando
Senior Protocol Engineer, SQRL

Alex Holmes
Former Chairman and CEO of MoneyGram

Dominic Obojkovitz
Lead Protocol Engineer, SQRL

Ashif Asharaph
Senior Protocol Engineer, SQRL

Version 4.1 · 8 November 2026

Abstract

SQRL is a settlement system for tokenised money. Institutions issue deposit tokens and payment stablecoins as contracts on the network, transfer them instantly across borders over the Lightning Network with foreign exchange converted in flight, and settle net positions with finality on Bitcoin without closing the channels that carried the payments. Each institution can run its own subnet for confidential business, while a shared canonical ledger holds the registry of subnets and the public asset layer through which value routes between them; every call on either tier is committed to Bitcoin, so any party with the public inputs can re-derive the ledger's full state without trusting the operator that computed it. Execution and governance are permissioned and accountable to supervisors through observer nodes, issuer whitelists, and a complete audit trail, while settlement and ordering remain open on Bitcoin. The network is quantum-reliable by construction: ledger integrity rests on hash-based commitments that survive quantum cryptanalysis, and the endpoints the network controls run NIST-standardised post-quantum cryptography. Network usage is metered in the network token, XRB, and settled by burning it; the full economic model is specified in a companion tokenomics paper. Banks are the first users of this system, not the only ones: the same infrastructure serves any institution that issues or moves regulated digital money.

1 Introduction

1.1 The Problem

Moving money across borders still runs on correspondent banking: a chain of intermediary banks, each holding prefunded accounts in the other's currency, passing messages and reconciling balances between separate ledgers. The costs are structural. Value sits idle in prefunded nostro and vostro accounts along every corridor; payments clear only within each bank's business hours and stop at daily cut-off times; a cross-border transfer can take days to arrive and its status is opaque while in transit; and each institution keeps its own ledger, so settlement is a continuous exercise in reconciling records that were never shared. Tokenised money removes some of this friction, but only if the token can move instantly, settle with finality, convert between currencies without a settlement-risk window, and do all of this without exposing one bank's client flows to another. No existing network does all four at once.

1.2 The System

SQRL is a settlement system built for exactly that. An institution issues its deposit tokens and stablecoins as contracts and runs its own subnet, a private ledger holding its own state. A shared canonical ledger holds the registry of subnets and the public asset layer, and value routes between subnets through it. Payments move instantly over the Lightning Network, with foreign exchange converted in flight, and net positions settle with finality on Bitcoin without closing the channels that carried them. The infrastructure beneath the system, Bitcoin for settlement and ordering and Lightning for instant transfer, is specified in Section 3; what the institution gains is a single network on which it can issue, move, convert, and settle regulated digital money, with supervisors watching through observer nodes and the whole history reconstructible from the Bitcoin chain.

1.3 A Settlement Network for Institutions

The second shift this paper responds to is regulatory and commercial rather than technical. Banks are adopting tokenised money at production scale. Bank-issued deposit tokens and stablecoins now have explicit legal lanes in several major jurisdictions, while prudential standards assign punitive capital treatment to bank exposure on fully open execution environments. The networks banks actually join share a recognisable shape: a named, accountable validator set at the execution layer; closed sub-networks for confidential business; supervisory visibility by design; predictable, protocol-fixed operating costs; and, increasingly, a credible answer to the question of cryptographic longevity over a multi-decade asset horizon. Banks are the first institutions to need this shape, but they are not the only ones: the same requirements describe any regulated issuer of digital money.

SQRL's execution model is unusually well suited to this shape, because it never derived its security from an open validator set in the first place. Validation is deterministic replay of Bitcoin-anchored calls: any party with the public inputs reaches the same verdict, so restricting who operates the execution layer does not weaken what can be verified. This paper therefore specifies the network as it is being deployed: permissioned execution and governance above an open settlement layer. Named institutions operate the canonical ledger and their own closed subnets; Bitcoin orders, finalises, and audits everything; and anyone, member or not, can verify the network's full history from the Bitcoin chain alone.

The target workload is equally specific. The network is built for payments and cross-border settlement: issuance of regulated digital money (deposit tokens and stablecoins), interbank transfer of those instruments, foreign-exchange corridors between them, and instant distribution over the Lightning Network. Sections 3 and 8 describe this infrastructure; Section 9 sets out the compliance and supervision surface; Sections 4 through 7 specify the execution substrate beneath it; Section 11 specifies the post-quantum posture that the name SQRL commits the network to; and Section 14 summarises how network usage is metered in XRB, with the full economic model in a companion tokenomics paper.

2 System Overview

SQRL provides a deterministic execution model with a single, public source of truth: Bitcoin. Contracts are written in SCL and compiled into bytecode C, and a cryptographic commitment $h = \mathrm{H}(C)$ is immutably embedded into a Bitcoin UTXO. Every subsequent call is itself anchored: the call travels as a signed call envelope, and a commitment to that envelope is carried on Bitcoin in a protocol stamp inside the anchoring transaction (Section 6). State transitions are validated off-chain by replaying payloads against C and Bitcoin-derived context (for example, the UTXO set, block height, and signatures), and each node folds the resulting write-sets into a per-block state root. This anchoring ensures immutability, a canonical total ordering of all state transitions by Bitcoin block, and objective validation.

This model builds on client-side validation, first proposed by Todd in 2016 [2] and later refined through the work of Orlovsky and the RGB project [3]: rather than executing computation on a shared chain, each node validates state transitions by deterministically replaying Bitcoin-anchored calls. SQRL takes that verifiable core and builds on it the permissioning, privacy, routing, and cryptographic posture that a regulated settlement network requires.

The same execution substrate is deployed in two tiers. The canonical ledger is the flagship SQRL instance, operated by a named set of accredited validators, carrying the shared contract registry, the subnet registry, and the public asset layer. Subnets are independent SQRL instances run from the same codebase by a bank or consortium: their members hold full subnet state, outsiders see only Bitcoin-anchored commitments, and their membership in the wider network is established by registering periodic checkpoints on the canonical ledger (Section 3.2). Both tiers anchor to the same Bitcoin chain, so the audit trail of the entire network, canonical and closed, is enumerable from public data.

3 Network Architecture

The network applies permission only where institutions require it: at execution and governance. Settlement and code remain open. Figure 1 shows the four layers and the permission applied at each.

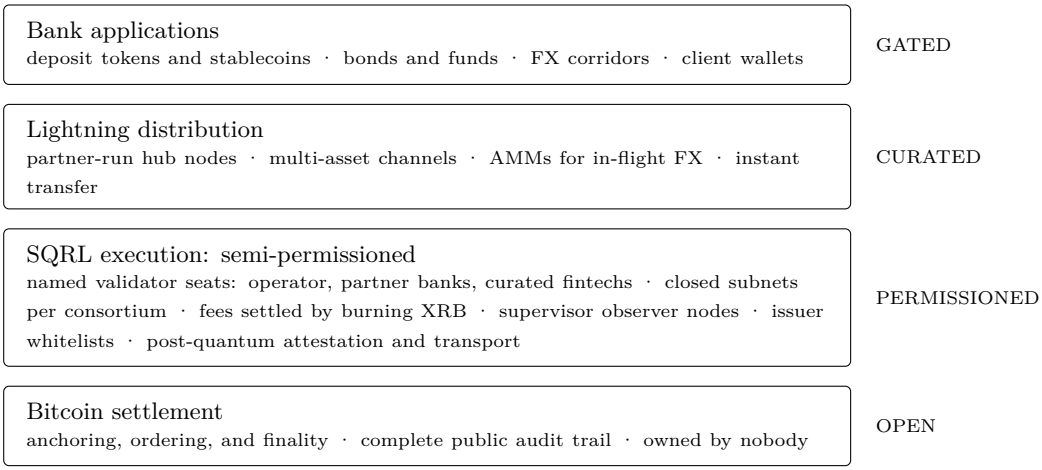


Figure 1: The four-layer stack. Permission decreases downward: the layers institutions touch are gated; the layer that settles is open.

3.1 The Canonical Ledger and the Validator Set

The canonical ledger is operated by a named set of institutions, initially five to nine seats: the network operator, partner banks, a custodian or market-infrastructure provider, and curated fintechs. Because SQRL consensus is deterministic replay of Bitcoin-anchored calls, a validator's job is attestation and state-root publication, not leader election: validators independently replay confirmed calls, publish per-block state roots, and co-sign network-level attestations under a t-of-n quorum (three-of-five minimum). There is no consensus ceremony and no slashing of bonded capital; misbehaviour is handled by governance removal and contract termination, which is how accredited validator sets operate in production financial networks. Crucially, the permissioning of seats does not reduce verifiability: any outside party can replay the same public inputs and detect a divergent root immediately (Section 6).

Validator seats carry three obligations: continuous replay and state-root publication; operation of network infrastructure (a Lightning hub, Section 8.3, or an observer-serving endpoint); and participation in the quorum that admits subnets, issuers above the free tier, and AMM operators. Seats are compensated from the validator emission pool, weighted by attested activity (the schedule is specified in the companion tokenomics paper), and the validator attestations themselves are the first endpoint the network secures with post-quantum signatures (Section 11).

3.2 Banking Subnets

A bank or consortium can run its own SQRL instance from the same codebase. A subnet is a complete SQRL deployment: its own contract registry, its own validator quorum drawn from its members, its own state. Three properties define the design:

- Members hold full state; outsiders see commitments. Subnet participants replay and validate everything, exactly as on the canonical ledger. Non-members see only the subnet's state-root commitments anchored on Bitcoin. Confidentiality is achieved by data distribution, not by new cryptography.
- Integrity comes from Bitcoin. Every subnet call is anchored with the same protocol stamps and anchor-outpoint binding as canonical traffic (Section 6). A subnet cannot rewrite its history without contradicting the Bitcoin chain, and a supervisor or auditor given the subnet's data can re-derive its entire state from public anchors.
- Membership comes from the canonical ledger. A subnet participates in the wider network by registering periodic checkpoints (its state root, height range, and validator attestations) with the subnet registry contract on the canonical ledger. Checkpoint registration is the protocol event that carries the subnet's metered anchoring rent, settled by burning XRB (Section 14). Registered checkpoints are the precondition for everything a subnet joins the network to obtain: cross-subnet value routing, access to the Lightning hub layer and its AMMs, and standing in the supervisor tooling. A subnet that stops registering checkpoints keeps its internal integrity but leaves the network.

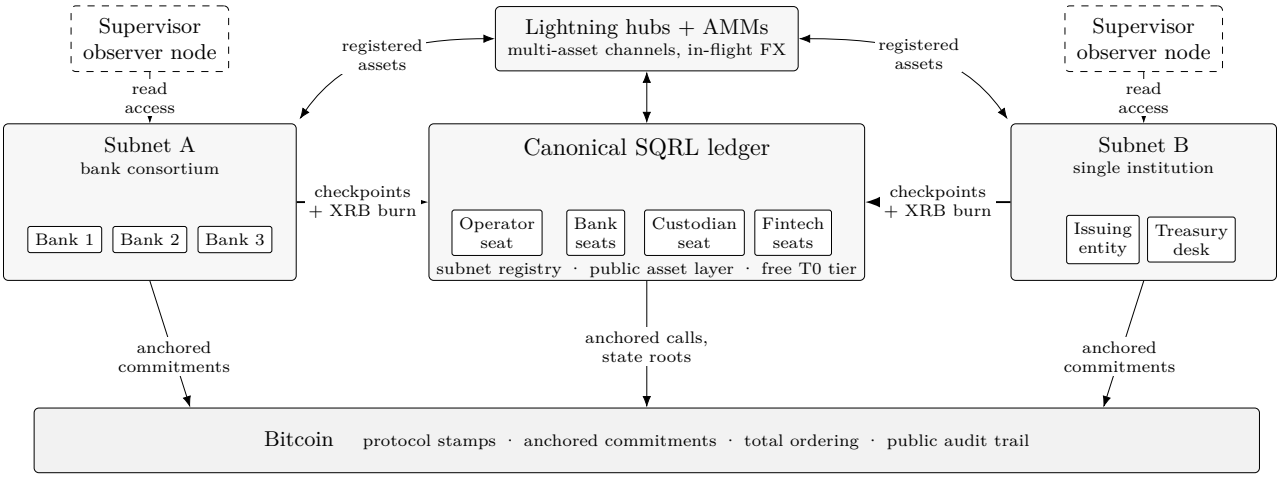


Figure 2: Network topology. Subnets anchor their own calls and state roots to Bitcoin for integrity, and register checkpoints (with burned XRB anchoring rent) on the canonical ledger for membership: cross-subnet routing, Lightning distribution, and supervisor standing all require registered checkpoints.

3.3 Cross-Subnet Routing Through the Canonical Ledger

A subnet in isolation is a private ledger. The network exists to let value move between subnets, and between a subnet and the public asset layer, and that movement is routed through the canonical ledger by design rather than through bilateral trust between banks. The canonical ledger is the only tier that holds the subnet registry and the shared asset layer, so it is the natural and the authoritative point through which a cross-subnet transfer passes.

Routing works on the checkpoint spine. When an institution in Subnet A pays an institution in Subnet B, the movement resolves against the registry state both subnets have already committed. The precondition is standing: the canonical ledger will route value out of or into a subnet only while that subnet holds a current registered checkpoint, because the checkpoint is what proves the subnet's state root to the rest of the network and what the counterparty tier verifies against. A debit recorded inside Subnet A appears in A's next checkpoint; a canonical settlement record anchors the transfer on Bitcoin and is attested by the validator quorum; the matching credit appears in Subnet B's next checkpoint. Two patterns carry the actual value:

- Anchored settlement for wholesale, non-urgent flows. The net position between two subnets, or between a subnet and the canonical asset layer, is cleared by a batch settlement call on the canonical ledger (Section 8.4), ordered and finalised by Bitcoin and metered in XRB.
- Lightning routing for instant flows. A registered asset moves over multi-asset channels between the two banks' hubs, converting currency in flight through an AMM hop if the corridor crosses a currency boundary (Section 8.3); the canonical ledger anchors only the net result when the channels splice or settle.

In both patterns the canonical ledger is where cross-subnet authority lives, and the security of the route reduces to two facts a counterparty can check without trusting the sending bank: the sending subnet holds a current checkpoint, and the settlement record is attested by staked validator seats and anchored to Bitcoin. Section 14 summarises how burned and staked XRB make both of those facts economically costly to fake, with the full model in the companion tokenomics paper.

3.4 What Stays Open, What Is Gated

Semi-permissioned is a precise claim, and the credibility of the design depends on stating it exactly.

Open: Bitcoin settlement and the complete audit trail; the protocol source (the reference node is MIT licensed, with full publication of the network stack targeted alongside mainnet economics activation); the free T0 execution envelope, under which simple operations such as token transfers remain free; and the XRB market itself, which remains freely traded. Anyone can verify the network from the Bitcoin chain alone, run a node from source, and replay the state.

Gated: validator seats and state-root attestation on the canonical ledger; deployment above the free tier; subnet membership and checkpoint registration; cross-subnet routing standing; the AMM operator set; and admission to the partner programmes. Permission is a property of the execution and governance layers only. The moment settlement or the token market is also gated, the network becomes a consortium chain; this boundary is therefore treated as a design invariant, not a policy.

4 Execution Model and the SCL Language

Contracts on SQRL are written in the Simple Contract Language (SCL). SCL is the language; SQRL is the ledger, node, and network that executes it. The two are designed together, and the language's restraint is what makes the ledger's guarantees hold.

4.1 Determinism and Boundedness

SCL deliberately forbids nondeterministic constructs such as time, randomness, or external oracles. This ensures that for any given payload $p \in \mathcal{P}$ and execution context $\Gamma \in \mathcal{C}$, the state transition Δ is uniquely defined by a total deterministic function:

$$\Delta = f_C(p, \Gamma), \quad f_C : \mathcal{P} \times \mathcal{C} \to \mathcal{S},$$

where $\mathcal{S}$ denotes the state space. Determinism implies that two independent evaluators, given identical inputs, will produce identical state deltas:

$$f_C(p, \Gamma) = f_C(p, \Gamma) \quad \forall\, p \in \mathcal{P}, \Gamma \in \mathcal{C}.$$

To mitigate denial-of-service and ensure computational predictability, SCL enforces hard ceilings on execution cost. Each execution is constrained by an instruction budget

$$N \leq 10^5,$$

stack depth

$$d \leq 1024,$$

and call depth

$$d_{\text{call}} \leq 64,$$

with persistent storage bounded at 64 KB per contract and payloads capped at 64 KB. The instruction budget is shared across nested cross-contract calls: however deep a call chain goes, the entire transaction runs under a single budget, so composition cannot amplify cost. Formally, for an execution trace $\tau = (s_0, s_1, \ldots, s_n)$, where each s_i is a machine state and $n = |\tau|$, SCL enforces the invariant

$$n \leq N \quad \wedge \quad \max_i \text{depth}(s_i) \leq d.$$

These invariants guarantee termination within polynomially bounded resource usage. Since each transition step can be evaluated in time $O(1)$ relative to the instruction budget, the total cost of evaluating a payload is bounded by

$$T(p, \Gamma) \in O(N).$$

Thus, every execution of SCL resides within **PTIME**, and more specifically within $\mathbf{O}(10^5)$ steps, regardless of the adversarial structure of p. This combination of determinism and bounded execution ensures that safety properties (for example, totality and termination) are provable, and denial-of-service vectors such as infinite loops, unbounded recursion, or uncontrolled memory allocation are mathematically excluded from the operational semantics. For a settlement network these are not abstract properties. Deterministic, bounded execution is what makes every balance and every transfer independently re-derivable by a counterparty bank, an auditor, or a supervisor, with no privileged party whose word must be taken.

4.2 Language Features

In SCL, each contract is authored in its own file and comprised of four essential components:

- Constants declared via const, used for embedding immutable values that remain fixed at compile time.
- Fields, which store the mutable state and are annotated with access-control via attributes like @writeable_by(init, bump). Such annotations are enforced at code generation time; if a function not authorised attempts to modify a protected field, the assignment is silently skipped without causing a runtime error.
- Events, declared at contract level via event and raised with emit, giving contracts a typed, queryable record of what happened.
- Functions marked by the fn keyword, which deterministically evolve the contract's state through state-modifying logic that respects write constraints and deterministic semantics.

A minimal illustrative contract from the documentation:

```
contract Minimal {
    // constants
    const STEP: u64 = 1;

    // fields
    @writeable_by(init, bump)
    counter: u64;

    // events
    event Bumped(value: u64);

    // functions
    fn init(x: u64) {
        counter = x;
    }

    fn bump() {
        counter = counter + STEP;
        emit Bumped(counter);
    }
}
```

Supported Types. SCL supports a restrained but expressive type system, allowing for rich yet well-typed contract logic:

- Scalar types: u64, u8, bool, Str, String.
- Collections: List⟨T⟩ for homogeneous sequences, HashMap⟨K, V⟩ for key-value mappings, and tuples written as (T1, T2, ...).
- Literals: numeric literals, strings, byte literals such as b"..." or hex-formatted, list literals like [1, 2], and map literals like {"k": 1} .

Notably, boolean values are compiled into integers at runtime: true becomes 1 and false becomes 0. Branching on booleans requires explicit comparison against 0 or 1.

Control Flow Constructs. SCL enables classic structured control flow, with syntax analogous to mainstream languages:

- Standard conditional branching using if, else if, and else. Example:

```
fn guard(n: u64) {
    if (n == 0) {
        PRINT("zero");
    } else {
        PRINT("nonzero");
    }
}
```

- Looping via for loops over concrete collections. Locals are immutable unless declared let mut. Example:

```
fn total(entries: List) {
    let mut s = 0;
    for e in entries {
        s = s + e;
    }
    PRINT("sum:", s);
}
```

There is no while loop and no unbounded iteration: every loop runs over a concrete collection, so every loop is bounded by construction. Recursion is syntactically possible but capped hard by the VM's 64-frame call-depth limit.

Events. Events are collected during execution, including those emitted by nested cross-contract calls, and are recorded when the call confirms on Bitcoin. Each stored event carries the Bitcoin transaction identifier of the emitting call and its confirmation height, and event schemas appear in the contract ABI, typed like function inputs. Nodes expose per-contract event queries filtered by event name and block range, so indexers and frontends can build provable activity feeds without contract-specific glue: state answers what is, events answer what happened, and both are provable against Bitcoin.

4.3 Built-ins and Context

The SCL runtime exposes a carefully curated set of built-in functions, designed to balance expressiveness with safety and determinism. These built-ins provide access to persistent state, debugging aids, blockchain context, and controlled inter-contract communication. By restricting developers to this finite and well-defined set of primitives, SCL ensures that all contract executions remain deterministic and resource-bounded.

Persistent Storage. Contracts are endowed with persistent key–value collections, exposed via map and set operations. These enable contracts to maintain state across invocations, while bounding storage operations within deterministic complexity classes:

$$\text{MAP_SET}(k, v) : (K \times V) \to \top, \quad \text{MAP_GET}(k) : K \to V \cup \{\bot\}, \quad \text{MAP_REMOVE}(k) : K \to \top,$$

$$\text{SET_ADD}(x) : T \to \top, \quad \text{SET_CONTAINS}(x) : T \to \{0, 1\}, \quad \text{SET_REMOVE}(x) : T \to \top.$$

All per-entry map and set operations execute in $O(1)$ expected time, ensuring contract performance is insensitive to the size of stored data. Whole-map reads (MAP_GET_ALL) and bounded list accessors (LIST_LEN, LIST_GET) round out the collection primitives; all remain subject to the shared instruction budget.

Debugging and Introspection. For development and verification, SCL provides:

$$\text{PRINT}(x) : T \to \top, \quad \text{TYPEOF}(x) : T \to \text{Str}.$$

These functions are side-effect free with respect to consensus-critical state; their effects are visible only during contract development or via log outputs.

Blockchain Context. Contracts can introspect Bitcoin-level execution state via dedicated context functions:

$$\text{SENDERS}() \to \text{List[PubKey]}, \quad \text{RECEIVERS}() \to \text{List[PubKey]}, \quad \text{SENDER_ADDRESS}() \to \text{PubKey},$$

$$\text{CONTRACT_ID}() \to \text{ContractID}, \quad \text{block_height}() \to \mathbb{N}.$$

Here, SENDERS() returns the set of public keys that authorised the transaction inputs (UTXO spenders), while RECEIVERS() exposes the public keys of transaction outputs; SENDER_ADDRESS() is the standard way to gate functions on the caller, and CONTRACT_ID() lets a contract refer to itself. This binding of contract execution to the Bitcoin UTXO model ensures that contract logic is always grounded in the consensus state of the underlying chain.

Cross-contract Communication. To enable modular design, SCL supports controlled invocation of other contracts:

$$\text{CALL}(c, p) : (\text{ContractID} \times \mathcal{P}) \to \Delta,$$

where c is the target contract identifier, p a payload, and Δ the resulting state delta. CALL propagates the caller's identity into the callee; its companion CALL_AS_CONTRACT invokes the callee with the calling contract as the sender, the standard pattern for escrow and staking contracts that hold balances in their own name. Nested calls share a single instruction budget and the global ceilings ($N \leq 10^5$, stack depth ≤ 1024, call depth ≤ 64), ensuring that cross-contract execution cannot be abused for denial-of-service or cost amplification. Events emitted by callees are collected alongside the caller's.

Transfers and Signature Verification.

$$\text{TRANSFER}(S, R, A) : (\text{List} \times \text{List} \times \text{List}) \to \{0, 1\}, \quad \text{CHECK_SIG}(\sigma, m, pk) : \text{Bytes}^3 \to \{0, 1\}.$$

TRANSFER performs a balance-checked multi-party transfer whose sender and receiver totals must reconcile, and CHECK_SIG deterministically verifies ed25519 and secp256k1 signatures, allowing contracts to validate off-chain authorisations in-band.

4.4 Turing Completeness Trade-offs

A central critique of many lightweight or alternative smart contract frameworks is the absence of Turing completeness, which theoretically allows arbitrary computation. However, practical Turing-complete blockchain environments have consistently shown high execution costs, inefficiency, and state bloat.

In designing SCL, we deliberately forgo strict Turing completeness to achieve determinism, efficiency, and verifiability. This trade-off is negligible in practice, and for the network's target workload it is not a trade-off at all: payments, issuance, whitelist management, escrow, and batch settlement are exactly the class of conditional transfers, state commitments, and proofs of authorisation that bounded execution expresses naturally. What a payments network needs is predictable execution, low cost, and scalable verification, not unrestricted computation.

Restricting SCL to deterministic and bounded operations ensures that execution is efficient and reliable across all nodes, mitigating the risk of runaway computation or adversarial exploits. While not Turing complete in theory, the language remains sufficiently expressive for the overwhelming majority of practical applications.

5 Compiler, Virtual Machine, and Node Architecture

This section describes the concrete implementation architecture underlying SQRL:

- the SCL compiler, which transforms human-readable contracts into bytecode;
- the SQRL virtual machine (VM), which executes that bytecode deterministically; and
- the SQRL node, which embeds the VM into a networked validator and settlement stack.

In principle, the design is chain-agnostic. The language and execution model do not assume any specific Layer 1 implementation beyond a UTxO-style transaction model, while the current reference node implementation targets Bitcoin as its settlement layer.[1] The same node binary runs the canonical ledger and every subnet; a subnet is a configuration of the stack described here, not a fork of it.

5.1 SQRL Stack Overview

At a high level, SCL contracts flow through the stack as follows:

1. A developer writes a contract in SCL.
2. The SCL compiler parses, validates, and compiles this source into a compact bytecode C together with associated metadata (constant pool, function offsets, variable layout).
3. The SQRL VM executes bytecode C in a sandboxed, deterministic environment, using:
   - a local, contract-scoped state store,
   - a fixed set of built-ins, and
   - Bitcoin-derived context (UTXOs, block height, transaction senders/receivers).
4. A SQRL node orchestrates compilation artifacts, VM execution, networking, and persistence, and anchors contract commitments on Bitcoin.

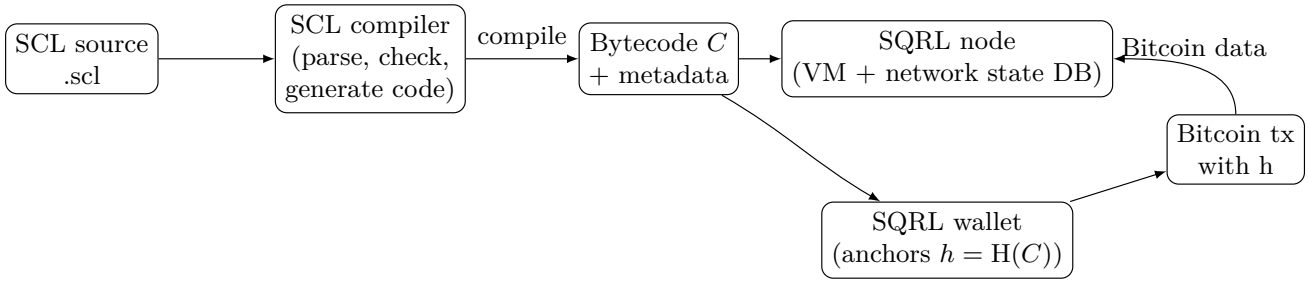


Figure 3: End-to-end path from SCL source code to compiled bytecode and execution. The SQRL wallet typically computes the hash $h = \mathrm{H}(C)$ and anchors it in a Bitcoin transaction. SQRL nodes then observe the Bitcoin chain, use the anchored commitment and relevant UTXOs as context, and execute the bytecode in the VM over the contract state to validate and update ledger state.

5.2 Compiler Design and Tooling

The SCL compiler is implemented as a modular pipeline with clearly separated concerns. Its role is to map a contract written in SCL to a canonical bytecode representation C and associated metadata used by the VM and by other tools in the SQRL ecosystem (wallets, deployment scripts, and the like).

Front-end: Parsing and Static Analysis. The front-end performs:

- Lexical analysis, transforming the source text into a token stream.
- Parsing, constructing an abstract syntax tree (AST) for contracts, fields, functions, expressions, and statements.
- Symbol table construction, tracking declared constants, fields, locals, built-ins, and their scopes.
- Semantic checks that enforce core language constraints:
  - rejection of syntactically unsupported control-flow constructs (only bounded for-style loops are admitted by the grammar);
  - enforcement of field access rules (for example, @writeable_by annotations are honoured during code generation so that unauthorised writes are not emitted);
  - validation of builtin usage and argument counts/types against a fixed registry of supported built-ins.

Contracts that violate these constraints fail to compile. Together with the VM's instruction, stack, and call-depth limits, this ensures that only contracts satisfying the determinism and boundedness guarantees of Section 4 can be deployed.

[1] At the time of writing, the production implementation focuses on Bitcoin Layer 1; support for additional UTxO-based layers and chains is not assumed by this paper.

Intermediate Representation and Code Generation. After successful analysis, the compiler lowers the AST into a small instruction set tailored to the SQRL stack-based VM. This intermediate representation is then assembled into:

- a constant pool, storing literals and static data;
- a bytecode stream C encoding VM instructions and jump targets;
- a function offset table, mapping function identifiers to byte offsets in C;
- per-function local layout metadata, such as the number of local variables and their allocation in the VM stack frame.

The compiler exposes this structured metadata that tooling can use, together with the original source, as well as an ABI of a contract's public interface. The same compilation pipeline is exposed both:

- as a command-line tool for local development and continuous integration; and
- by the hosted SCL IDE and online compiler.

This ensures that contracts compiled in-browser and contracts compiled offline share identical semantics.

Commitment and Canonicalisation. The canonical bytecode C from the compiled artifact is hashed and embedded into Bitcoin. Any change to the source that affects semantics yields a different C and thus a different h as detailed in Section 13. This binding allows independent nodes to verify that the bytecode they execute corresponds to the on-chain commitment without needing access to the original source.

5.3 Virtual Machine Execution Model

The SQRL VM is a deterministic, stack-based interpreter specialised for the bytecode generated by the SCL compiler. It is designed to be:

- sandboxed: no direct access to system time, randomness, or the network;
- resource bounded: each execution is constrained by instruction count, stack depth, call depth, and storage limits; and
- chain-aware: Bitcoin and UTxO information is supplied as read-only context.

Each VM instance is parameterized by:

- the compiled bytecode C and constant pool;
- an array of local variables and contract fields;
- a stack for intermediate values;
- an optional handle to a persistent key–value store that backs contract fields and collections;
- a context map of named values (for example, block height, transaction identifiers, and the public keys of senders and recipients);
- configurable limits (maximum instructions, maximum stack size, maximum call depth, maximum storage per contract).

Given a payload p and context Γ, the node constructs the initial VM state and runs the bytecode until one of the following occurs:

1. the program terminates successfully (normal return);
2. the VM reaches a resource limit (instruction budget, stack depth, call depth, or storage bound); or
3. a deterministic error condition is encountered (for example, an invalid opcode or a type mismatch in a built-in).

In all cases, the outcome is fully determined by (C, p, Γ) and the prior contract state, with no hidden sources of nondeterminism.

Persistent Storage and Context. The VM exposes the storage and context facilities described in Section 4 as built-ins:

- map and set operations backed by a contract-scoped key–value store; and
- read-only accessors for Bitcoin-derived data (senders, receivers, block height, and so on).

In the reference implementation, persistent storage is powered by an embedded key–value database, and contract state is logically namespaced by a unique contract identifier. In principle, the VM itself is agnostic to the concrete storage engine; it requires only a deterministic key–value interface.

Cross-contract Calls. SCL supports bounded cross-contract calls via the CALL built-in and a global contract registry maintained by the node. Each registered contract is associated with its bytecode, constants, function offset metadata, and access to the shared persistent storage backend (namespaced by contract identifier). When a contract invokes another via CALL:

1. the VM resolves the target contract and function in the registry;
2. it instantiates a fresh VM interpreter for the callee contract, reusing the same underlying storage handle but isolating execution state (stack, locals, instruction pointer);
3. it executes the callee function under the caller's remaining resource budget: nested calls share one instruction budget and the global stack and call-depth ceilings, so composition cannot amplify cost; and
4. it returns the callee's explicit return value (a typed VM value) to the caller. If the call fails or the callee returns no value, the reference implementation returns 0 as an error sentinel.

This design allows modular composition while preserving deterministic replay, with failures yielding a deterministic result.

5.4 Node Integration and Network Role

The SQRL node embeds the compiler and VM into a full networked application that:

- ingests SQRL-related payloads;
- derives Bitcoin context from a configured backend;
- executes contract logic in the VM; and
- persists contract state, payload history, and auxiliary indexes for explorers and wallets.

Conceptually, the node is composed of the following subsystems:

API layer. Exposes REST and gRPC endpoints to deploy contracts, submit signed call envelopes, poll call status, read contract state and per-block state roots, and query per-contract events by name and block range.

Consensus and execution layer. Normalises incoming payloads into a canonical internal format, extracts the referenced contract identifier and function, and invokes the VM. Domain-specific logic (for example, list/bid/transfer flows) is implemented by SCL contracts; the node coordinates their evaluation but does not introduce an independent consensus protocol.

Storage layer. Maintains:

- queues of pending and confirmed payloads,
- the set of known contracts and their compiled artifacts,
- cached Bitcoin transaction information and UTXO state, and
- auxiliary data structures (for example, indexes and validator presence records).

Network and synchronisation. Handles peer discovery and synchronisation of contract state and payload history. To minimise bandwidth, the node can exchange compact summaries (for example, Merkle roots and diffs) rather than full histories. On a subnet, synchronisation is restricted to the member set; only anchored commitments leave the membership boundary. Inter-node transport is the second endpoint the network secures with post-quantum key establishment (Section 11).

Validator coordination. For networks that reward validation, a coordinator/validator mode uses cryptographic challenges and signed presence claims to measure validator participation over time. On the canonical ledger these attestations feed the activity weighting of the validator emission pool (specified in the companion tokenomics paper). Importantly, this mechanism does not alter the fact that disputes about contract execution are objectively resolvable by replaying bytecode against Bitcoin-derived context.

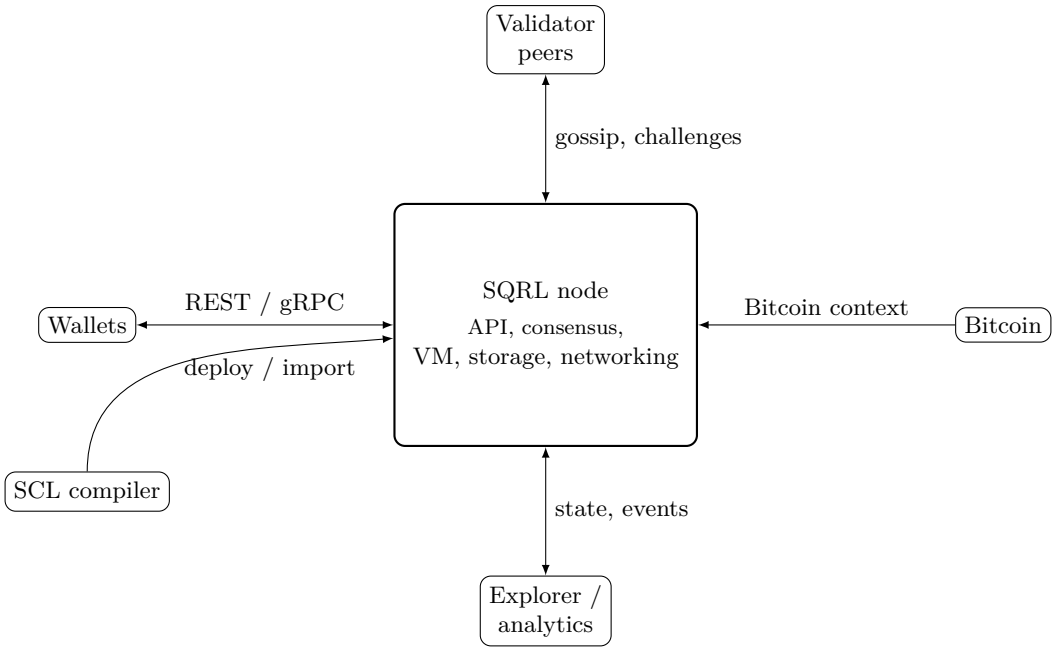


Figure 4: The SQRL node within the broader stack: the SCL compiler, wallets, explorer, validator network, and Bitcoin.

5.5 Developer Workflow: From Source Code to Execution

From a developer's perspective, the path from SCL source code to live execution proceeds through the following steps:

1. Authoring. The developer writes an SCL contract file, defining constants, fields, and functions according to the language model in Section 4. This can be done in a local editor or in the hosted SCL IDE.

2. Compilation. The contract is compiled using either:

   - the command-line compiler; or
   - the web-based compiler backed by the same compilation pipeline.

   The compiler outputs bytecode C, metadata (constant pool, function offsets, field layout), and an ABI description.

3. Anchoring on Bitcoin. A deployment tool or wallet constructs a Bitcoin transaction that embeds $h = \mathrm{H}(C)$ (for example, in an OP_RETURN output or other commitment-compatible pattern). This transaction is broadcast and settled on Bitcoin, providing an immutable anchor for the contract's semantics. Every subsequent call to the contract is anchored the same way, via a protocol stamp committing to its call envelope (Section 6).

4. Registration in the Node. Upon receiving a deployment (or invocation) request, the node persists the submitted artifact and request data keyed by the associated Bitcoin transaction identifier (txid) and marks it as pending. The node monitors the Bitcoin chain and, once the txid reaches a configurable confirmation threshold, promotes the entry to active and registers it in the in-memory contract registry (using the same txid as the contract identifier in the reference implementation).

5. Invocation and Validation. To invoke a contract function, a client submits a signed call envelope that carries:

   - the network and contract identifiers,
   - the function name and serialised arguments,
   - the caller's public key, a unique call identifier, and the caller's signature, and
   - the Bitcoin outpoint that the call's anchoring transaction must spend.

   The node:

   (a) validates the envelope's anchor binding at intake and queues the pending call;

(b) once the anchoring transaction confirms, re-validates the binding against the on-chain protocol stamp, resolves the commitment h, and fetches relevant Bitcoin context (UTXOs, block data);

(c) constructs the VM context Γ and executes $f_C(p, \Gamma)$ in the SQRL VM;

(d) persists the resulting state delta with undo data, folds it into the block's state root; and

(e) returns a structured result (for example, success/failure, logs, emitted events, and an updated state summary) to the client.

This workflow is designed so that each step is independently verifiable. Any third party with access to the same inputs (contract bytecode C, payload p, context Γ derived from Bitcoin, and prior state) can reconstruct and validate the node's verdict. It is expected that new nodes joining the network typically traverse this process, replaying contract history from the same public inputs, to independently validate the current contract state and satisfy themselves as to its correctness.

6 Per-Call Anchoring and Settlement

Anchoring in SQRL is not limited to deployment: every contract call is committed on Bitcoin. This section describes the call container, the on-chain commitment, and the two-phase validation pipeline that together make replay, re-anchoring, and front-running structurally impossible.

6.1 Call Envelopes and Protocol Stamps

A call travels as a signed call envelope: the network identifier, contract identifier, function name, serialised arguments, caller public key, a unique call identifier, and the caller's signature. Crucially, the envelope also names the exact Bitcoin outpoint that its anchoring transaction must spend.

The anchoring transaction carries a protocol stamp in an OP_RETURN output: a compact commitment to the envelope hash and the contract identifier. When the transaction confirms, the call is final and totally ordered by its position in the Bitcoin chain.

The anchor-outpoint binding closes replay and front-running holes structurally rather than heuristically:

- an envelope cannot be re-anchored in a different transaction, because the anchor outpoint would not match;
- a third party cannot front-run a call with their own transaction, because they cannot spend the caller's outpoint; and
- the confirmed stamp binds the routed function name to the committed envelope, so a forged wrapper cannot redirect a call to a different function than the one committed on-chain.

6.2 Two-Phase Validation

Nodes validate every call twice. At intake (from a client or gossip), the node parses the envelope, validates the anchor binding (the anchoring transaction's first input must spend the outpoint the envelope names), and queues the pending call. At confirmation (block by block), the node scans each Bitcoin block for protocol stamps and, for each one found: re-validates the anchor binding, a consensus-critical check under which a mismatched or unparseable envelope is invalid on every honest node with no execution; derives context Γ from Bitcoin (block height, sender address, and the anchor transaction's inputs and outputs); deterministically executes the call against the anchored bytecode; and commits the resulting write-set together with undo data. Invalid payloads fail locally; no global voting is required.

6.3 State Roots and Replay Consistency

At every Bitcoin block, each node computes a Merkle state root over all contract state, folding in each confirmed call's write-set. Two nodes with the same root at the same height provably hold the same state, and anyone can re-derive the root from the confirmed call log. Divergence is therefore detectable at a glance, and audits reduce to deterministic replay against public inputs. On the canonical ledger, validator seats additionally co-sign published roots; on a subnet, the member quorum does. In both cases the signature set is an accountability record layered over a verdict any observer can reproduce independently, and those signatures are the post-quantum attestations of Section 11.

6.4 Reorg Handling

SQRL settlement follows the Bitcoin chain. Nodes retain undo logs and a block journal covering the last 144 blocks (approximately one day): a shallow reorganisation rolls state back automatically and replays the new branch, while deeper divergence triggers a full re-derivation from the confirmed call log. Because every input is public and deterministic, both paths converge byte-identically on every honest node.

6.5 Finality Characteristics

Stated plainly for risk assessment: Bitcoin finality is probabilistic, with conventional confirmation depths reached in 30 to 60 minutes, and one-block reorganisations occur several times a year (the block journal replays them automatically). Base-layer throughput is low, which is why the network batches (a single settlement call clears up to roughly 9,000 transfers, Section 8.4) and why instant movement happens on Lightning with Bitcoin-paced finality behind it (Section 8.3). The two timescales are reconciled by splicing. A payment is instant and irrevocable to its recipient the moment its Lightning HTLC settles; the net position that payment contributes to is settled with final, on-chain settlement on Bitcoin; and splicing settles that position, and adjusts channel liquidity, without closing the channel that carries the payments (Section 8.2). Instant settlement on Lightning and final settlement on Bitcoin are therefore not a choice between two systems but two stages of one: the payment is final to its recipient immediately, and its on-chain settlement follows at Bitcoin's pace underneath, with no interruption to the channel. None of these limits is disqualifying for wholesale settlement, where final on-chain settlement within an hour compares favourably with T+2 correspondent chains, but each belongs in partner due-diligence materials, and this paper states them before a risk committee has to find them.

7 Data Availability and Partitioned State

An envelope call commits a hash of the signed payload on Bitcoin; the payload itself travels off-chain through the node network. This keeps Bitcoin usage minimal, but it creates a gap: the chain can prove that a call happened without every node holding the data required to execute it. If a payload is withheld, execution of the affected state must stall until the data arrives or the call expires. The design question that matters is the blast radius of that stall: if the stall unit were the whole contract, one withheld payload could freeze every holder of a token for the price of a single transaction fee.

7.1 Function Classification

SCL functions are classified as global or partition-scoped. For a partition-scoped function, the state a call can touch is derivable from its anchoring Bitcoin transaction alone: the token-bound input slots the transaction consumes, plus the output slots the transaction itself creates. This changes the stall unit from the whole contract to the withholder's own transaction footprint. If one user withholds the payload for their transfer, only the slots their transaction consumed and created freeze; every other user's calls read and write disjoint slots and execute normally.

Address-keyed balance maps are deliberately refused partition classification. A recipient's balance key can be touched by anyone who sends to them, so under address-keyed state a withheld transfer would freeze the recipient's balance merely for having been named as recipient, a griefing vector costing one transaction fee. Partition scoping is therefore tied to UTXO-bound state, where Bitcoin already guarantees that each outpoint is consumed exactly once.

7.2 Declarations Are Checked, Not Trusted

A classification is a performance promise, never a safety assumption. When a withheld payload finally arrives, the VM captures the call's actual read and write set and checks it against the partition that was frozen. A call that escaped its declared partition degrades to the standard rollback-and-replay path, which converges byte-identically on every node regardless. A wrong or malicious declaration can cost performance; it cannot corrupt state or cause divergence.

7.3 Time-Boxed Stalls

Partition stalls cannot last forever. Envelope transfers carry a void window (default 144 blocks, configurable per contract at deployment): the sender may void an unrevealed transfer within the window, and at expiry an unrevealed, unvoided transfer resolves automatically by the interface's deterministic fallback, carrying the tokens to a sender-designated output. Block distance is the only clock involved, so every node resolves it identically.

7.4 Interfaces and Compact Operations

Contracts declare conformance to versioned standard interfaces through source-level annotations: at contract level, @interface(utxo_token_v1), and on functions, the @partition_scoped, @global (the default), and @compact_only classifications. Conformance is enforced twice, deliberately: at compile time by the compiler, and again at registration by the node against the emitted ABI and deploy metadata, so a hand-crafted artifact cannot claim conformance it does not have. Interface identifiers are versioned by suffix; a changed requirement is always a new identifier, never a mutation of an existing one.

The first standard interface, utxo_token_v1, binds token ownership to Bitcoin outpoints rather than address-keyed balances. Recipients are addressed by vout index: a transfer names an output of its own anchoring transaction, the tokens bind to whatever outpoint that output becomes, and ownership of the outpoint is ownership of the tokens. This inherits Bitcoin's double-spend safety and composes naturally with Lightning channels and HTLCs, which are outpoint-level constructs (Section 8.3).

For interface-defined operations, conforming contracts may additionally use compact operations: the selector and arguments (at most 43 bytes; a token transfer is roughly 10) ride directly in the anchoring transaction's OP_RETURN, in place of the payload hash. The consequences are significant: there is no envelope and therefore nothing to withhold, so the stall surface is zero; authorisation is the Bitcoin spend itself, so replay is structurally impossible; and the operation is final at Bitcoin depth, with no void window. The trade-off is that arguments become chain-visible, so compact operations are opt-in per contract. A contract deployed @compact_only accepts nothing else and, as a result, can never stall.

8 Payments and Cross-Border Settlement

The network's target workload is the movement of regulated digital money: issuance, interbank transfer, foreign exchange, and open distribution over Lightning. This section describes that infrastructure end to end.

8.1 Asset Issuance: Deposit Tokens and Stablecoins

Banks issue instruments as SCL contracts conforming to the standard token interfaces. Two instrument shapes dominate, and the contract layer supports both natively:

- Deposit tokens: transfer-restricted bank liabilities whose holders must be customers of the issuing bank or members of the issuing consortium, the instrument banks increasingly prefer for interbank value.
- Payment stablecoins: fully reserved claims transferable to any whitelisted holder, issued under the applicable statutory lane in the issuer's jurisdiction.

Both are expressed with the same primitives. The issuer controls mint and redeem functions, gated by @writeable_by annotations and SENDER_ADDRESS() checks; a contract-level whitelist (a persistent set of approved holder keys) governs who may receive the instrument, with transfer logic refusing movement to non-members; and observer roles (Section 9.2) give the relevant supervisor read access to holdings and history. Because whitelists are ordinary contract state, every addition and removal is itself an anchored, auditable event. Issuance on a closed subnet and issuance on the canonical ledger use identical contracts; the choice is a confidentiality decision, not a technical one.

8.2 Assets over the Lightning Network

The UTXO-bound token model of Section 7 composes directly with the Bitcoin Lightning Network, because channels and HTLCs are themselves outpoint-level constructs. SQRL assets are bound to a Lightning channel at open time: the channel funding transaction is built from the opener's token-bound UTXOs, and the binding is registered against the channel's funding outpoint. From that moment, a per-channel asset allocation, sequence-numbered and co-signed by both parties, is the authoritative record of ownership inside the channel, maintaining the invariant that the two sides always sum to the bound total.

Asset transfers then ride ordinary Lightning payments. The recipient registers asset terms (contract identifier and amount) against a standard BOLT11 invoice's payment hash; intermediate hops route the payment normally, with no knowledge of the asset layer; and when the payment settles, the allocation update is applied atomically with the HTLC preimage release, keyed to the same payment hash, so the asset transfer settles exactly when the payment does. Balances therefore move instantly and off-chain, and the channel does not have to close for that value to reach Bitcoin.

Splicing: final settlement without closing. A channel settles to Bitcoin by splicing, not by closing. A splice modifies the channel's funding transaction in place: the two parties co-sign a spend of the current funding outpoint into a new one that pays out an agreed net position, or that adds or removes liquidity, and the channel continues on the new funding outpoint without interrupting the payments it carries. For a SQRL asset channel this means a hub can realise a net position to Bitcoin with final, on-chain settlement, or top up a drained direction, while the channel stays open and keeps clearing payments instantly. Payments remain instant on Lightning and their settlement is final on Bitcoin, with no trade-off between the two: splicing settles the net position underneath without the channel ever closing. Channel close, below, is therefore the exceptional path, taken to exit or to resolve a dispute, rather than the normal route to settlement.

Settlement back to Bitcoin at channel close, cooperative or forced, is automated and crash-safe: detected closes are persisted as durable settlement intents before any settlement is forwarded, intents are acted upon only after a confirmation threshold and re-verification (so reorganisations roll intents back), the final split maps each party's co-signed allocation onto the closing outputs that pay them, and the settlement call on the SQRL node is idempotent. Custody remains client-side throughout: a WebAssembly signer derives on-chain and per-channel keys from one seed and signs locally in the user's browser, while an always-online hub performs Lightning connectivity and channel management but can only request signatures, never produce them.

8.3 Partner Hubs, Multi-Asset Channels, and In-Flight FX

In the bank network, the Lightning layer is operated by the institutions themselves. Partner banks and curated fintechs run hub nodes: well-capitalized, always-online Lightning nodes holding channels with their own clients, with other hubs, and with the network's AMM operators. Channels are multi-asset: a single channel's allocation records balances in multiple SQRL instruments simultaneously (for example, a bank-issued local-currency stablecoin alongside a USD stablecoin), each tracked and co-signed under the same sequence-numbered allocation.

Foreign exchange happens in flight. An AMM is an SCL contract holding two-sided inventory in a currency pair, operated by an admitted market maker and quoting a deterministic curve. When a payment crosses a currency boundary, the route passes through a hub hosting the relevant AMM: the inbound HTLC delivers the source-currency asset, the AMM contract prices the conversion, and the outbound HTLC carries the destination-currency asset, all keyed to the same payment hash, so the conversion settles atomically with the payment or not at all. The sender knows the delivered amount before committing; there is no settlement-risk window between legs, which is the property on which cross-currency settlement systems are built.

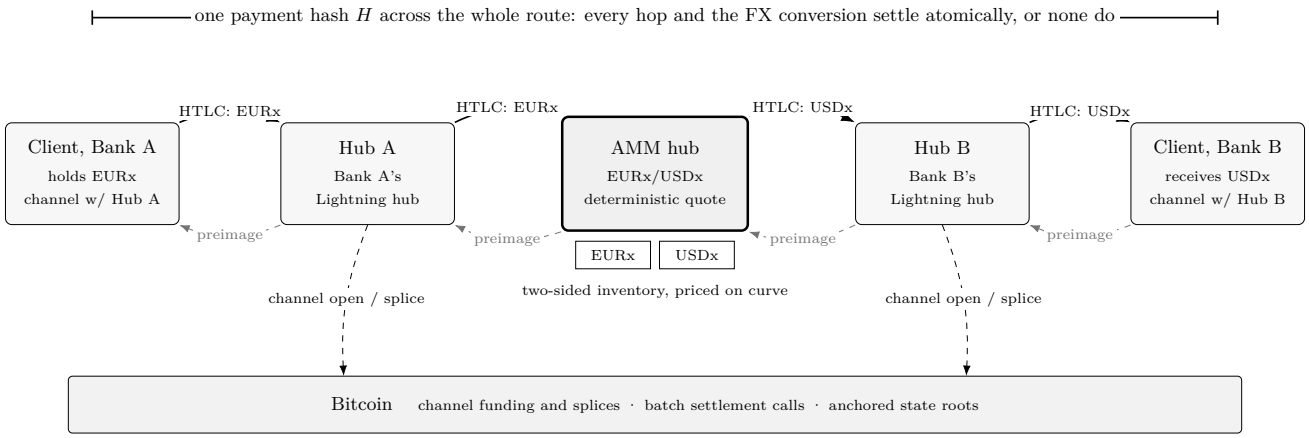


Figure 5: A cross-border payment with in-flight FX. The sender's source-currency stablecoin (EURx) is routed through the hub hosting the EURx/USDx AMM, converted at the contract's deterministic quote, and delivered as USDx, atomically under one payment hash. Bitcoin is touched only at channel opens, splices, and batch settlement; the channels stay open through settlement.

A corridor is therefore assembled from parts the network already defines: an issuer on each side (Section 8.1), a hub on each side, an AMM on the pair, and ordinary Lightning routing between them. Adding a new corridor is a liquidity and licensing exercise, not an engineering one. No comparable institutional network has this distribution layer: interbank ledgers settle positions between institutions but have no open payment network over which a bank-issued asset can actually move to a counterparty that can receive it.

8.4 Batch Settlement

Wholesale flows between institutions do not need per-payment anchoring. The network provides batch settlement calls: a single anchored SCL call that clears up to roughly 9,000 transfers in one Bitcoin transaction, with the batch's write-set folded into the state root like any other call. Interbank obligations accumulated across a business day (or across a Lightning corridor's net position) settle in one anchored, auditable operation whose finality is Bitcoin's. Batch settlement is also the unit on which value-based settlement fees are levied (Section 14): the fee follows the economic value settled rather than the instruction count, reflecting that a bank network performs fewer, larger transactions than an open network.

8.5 Corridor Lifecycle

End to end, a corridor operates in three repeating phases: (i) distribution, in which clients hold instruments in Lightning channels with their bank's hub and pay instantly, including across currencies via AMM hops; (ii) netting, in which hubs accumulate inter-hub and inter-bank positions inside open channels, touching Bitcoin not at all; and (iii) settlement, in which net positions are settled to Bitcoin by splicing the channels that carry the flow and by batch calls for wholesale obligations, with final on-chain settlement and without closing the channels, so distribution capacity is restored in place rather than torn down and rebuilt. Because settlement no longer requires a channel to close, the three phases run continuously and overlap: payments stay instant on Lightning while settlement proceeds on Bitcoin underneath. Supervisors observe all three phases through observer nodes; auditors can replay all three from public anchors plus member-provided data.

9 Compliance and Supervision

Regulated instruments bring obligations, and the network treats them as first-class rather than as controls bolted on after the fact. Four mechanisms carry the compliance surface: issuer whitelists and approvals at the contract level, observer nodes for supervisors, a complete audit trail on Bitcoin, and identified onboarding for AML and travel-rule obligations at the participant layer.

9.1 Whitelists and Issuer Approvals

Every issuer contract carries a whitelist: a persistent set of approved holder keys that transfer logic consults before moving the instrument, refusing movement to any key not on it. The issuer controls mint, redeem, and whitelist-management functions, gated by @writeable_by annotations and SENDER_ADDRESS() checks (Section 8.1). Because a whitelist is ordinary contract state, every addition and removal is itself an anchored, auditable event, so who was permitted to hold an instrument, and when, is answerable from the record rather than from an issuer's assertion. The network-level approvals, admission above the free tier, subnet membership, and the AMM operator set, are governed the same way, by the validator quorum (Section 3.1).

9.2 Observer Nodes

A supervisor receives an observer node on the subnets relevant to its mandate: a full replaying node with read access to the instruments it regulates and no write or governance capability. Because validation is deterministic replay, an observer node does not trust the subnet's members; it independently re-derives state from the subnet's data and the public Bitcoin anchors, giving the supervisor continuous, provable visibility rather than periodic reporting. At the contract level, issuer contracts expose observer roles: designated keys with read access to holder lists, balances, and event history for the instruments under supervision. The observer link is itself a network endpoint and is secured with the post-quantum transport profile of Section 11, so a supervisor's feed cannot be harvested today for decryption later.

9.3 Audit Trail

Every call on the network, on the canonical ledger and inside every subnet, is committed to Bitcoin (Section 6), so the audit trail is not a report the operator produces but a property of the chain. An auditor or supervisor given a subnet's data can re-derive its entire state from public anchors and detect any divergence from the committed state roots. Whitelist changes, issuance, redemption, and settlement are all anchored events carrying Bitcoin transaction identifiers and confirmation heights, queryable by contract and block range.

9.4 AML and Onboarding

Travel-rule and AML obligations are met the way every permissioned network meets them: identified onboarding at the participant layer, off protocol. Participants are subject to the admitting institution's KYC before they are added to any whitelist, and the binding between a real-world identity and a holder key lives in that institution's records, not on-chain. The protocol's role is to make the on-chain consequences of those decisions, who holds what and who was permitted to hold it, provable after the fact.

10 Privacy for Banking Subnets

The canonical ledger is fully transparent: call payloads and state roots are public. Banks will not put client flows on that, and the network does not ask them to. Privacy is delivered in three phases, each built on techniques already proven in production financial networks, and deliberately avoiding approaches the industry has abandoned.

Phase 1: closed subnets (ships with the first bank deployments). Confidentiality by data distribution. Subnet members hold full state; outsiders see only Bitcoin-anchored state-root commitments; supervisors receive observer nodes; issuers enforce whitelists at contract level. This requires no new cryptography: it is the need-to-know structure banks have already shipped on other permissioned networks, with Bitcoin in place of a foundation-run synchroniser. Its limit is the membership boundary: within a subnet, members see each other's flows, which is acceptable inside a single institution or a tightly scoped consortium.

Phase 2: confidential amounts. For shared subnets where competing banks coexist, amounts and asset types are blinded with Pedersen commitments: totals remain verifiable (commitments are additively homomorphic, so conservation checks still hold) while individual values are hidden from other members. Auditor and supervisor view keys preserve full visibility for oversight. Confidential transactions of this construction have operated continuously on a production Bitcoin sidechain for a decade without a cryptographic failure.

Phase 3: zero-knowledge compliance proofs. As tooling matures, ZK proofs allow an institution to prove compliance facts (whitelist membership, solvency, sanctions-screening status) without revealing counterparties. This phase is explicitly staged last: the primitives are maturing quickly but are not yet at the operational standard a bank deployment requires.

Two approaches are ruled out by the historical record rather than by taste. Trusted execution environments

as the root of trust are excluded because repeated enclave key extractions have shown that a single hardware break can retroactively decrypt a chain's entire privacy history; enclaves may be used as hardening, never as the trust root. Channel-partitioning schemes of the kind built into earlier enterprise ledgers are excluded because the industry that built them has formally sunset them. Separately, the network's asset model avoids the one component that has actually failed in Bitcoin-adjacent institutional systems: there is no pooled custody federation and no peg. SQRL assets bind to Bitcoin UTXOs directly, and state replays deterministically from the chain.

11 Post-Quantum Security

The institutions SQRL is built for hold assets on multi-decade horizons, and their diligence questions about cryptographic longevity are increasingly specific. The name of the network states the commitment the design makes to them: a Secure Quantum Reliable Ledger. This section sets out exactly what that commitment is and, as importantly, what it is not, because the credible position on post-quantum security is a precise one, and a vague one would be worse than silence.

11.1 What "Quantum Reliable" Means Here

SQRL's integrity does not depend on any signature scheme remaining unbroken. The ledger's authority rests on hash-based commitments: the contract commitment $h = \mathrm{H}(C)$, the per-block Merkle state root, the protocol stamp, and the Bitcoin block chain that orders them. Cryptographic hash functions are the primitive least affected by quantum computing. The best known quantum attack, Grover's algorithm, delivers only a quadratic speedup, which a 256-bit hash absorbs with a wide margin, whereas Shor's algorithm breaks the discrete-logarithm and factoring problems underlying today's signatures outright. The consequence is structural rather than aspirational: even under a cryptographically relevant quantum computer, a party who recorded the Bitcoin chain and the network's data can still re-derive and verify the entire history of the ledger, because that verification is replay against hash commitments, not signature checking. Integrity survives the quantum transition by construction.

The exposed surface is therefore narrow and specific: the digital signatures that authorise new actions. SQRL treats this surface in two parts, drawn along the line it actually controls. The Bitcoin settlement layer is governed by Bitcoin consensus, which SQRL does not and should not attempt to change; there the network relies on the key-hygiene mitigation of Section 11.4 and the extensibility path that Bitcoin itself is expected to adopt. The endpoints SQRL does control, namely the attestations, transports, and signing wallet that are the network's own software, are built to run NIST-standardised post-quantum cryptography, as Section 11.3 describes.

11.2 The NIST Standards Adopted

SQRL's post-quantum profile is built exclusively on the algorithms standardised by the United States National Institute of Standards and Technology in 2024, so that institutional reviewers evaluate named, published standards rather than bespoke cryptography:

- ML-KEM (Module-Lattice Key-Encapsulation Mechanism, FIPS 203 [4]), for establishing the symmetric keys that protect network transport.
- ML-DSA (Module-Lattice Digital Signature Algorithm, FIPS 204 [5]), for the network's own attestation and checkpoint signatures.
- SLH-DSA (Stateless Hash-Based Digital Signature Algorithm, FIPS 205 [6]), as a conservative, hash-based option for long-lived root and issuer keys whose security assumption is the same hash primitive the ledger already relies on.

Where post-quantum schemes protect transport or wrap an existing classical signature, they are deployed in hybrid mode, combining a classical primitive (for example X25519 or Ed25519) with its post-quantum counterpart so that the channel is secure if either component holds. Hybrid deployment is the conservative institutional default during the transition: it cannot be weaker than the classical scheme alone, and it defends against the harvest-now-decrypt-later adversary who records traffic today to decrypt once a quantum computer exists.

11.3 Securing the Endpoints SQRL Controls

The network's own software defines several endpoints that carry long-lived or sensitive material, and each is in scope for the post-quantum profile:

- Validator attestations and checkpoint signatures. The co-signed state roots of Section 6 and the subnet checkpoints of Section 3.2 are SQRL-protocol signatures, independent of Bitcoin consensus. Carrying them with ML-DSA means the network's accountability record remains unforgeable against a quantum adversary without any change to Bitcoin. Because the underlying verdict is always reproducible by replay, a signature scheme can even be rotated without rewriting history.

- Inter-node and observer transport. Gossip between nodes, inter-validator RPC, and the supervisor's observer-node feed are protected with hybrid ML-KEM key establishment, closing the harvest-now-decrypt-later channel on a supervisor's or a subnet's data in transit.
- The signing wallet. Custody is client-side: a signer derives keys from one seed and signs locally, and the hub can only request signatures (Section 8.2). The wallet is the natural place to offer post-quantum authorisation to users who require it, over the Bitcoin path described next.

Consistent with a measured rollout, these are profile capabilities the network is built to run rather than flag-day protocol mandates: operators adopt the post-quantum profile on the endpoints above as part of standard onboarding, while classical verification remains valid during the hybrid transition. This staging mirrors how regulated infrastructure actually migrates cryptography, and it avoids the two failure modes that damage credibility, namely claiming a flag-day cutover that has not happened and claiming a proprietary scheme in place of a published standard.

11.4 The Bitcoin Authorisation Path

Authorisation that must be verified by Bitcoin consensus cannot be unilaterally upgraded by SQRL, so the network takes the same prudent posture a Bitcoin-native system should.

Single-use keys as the primary mitigation. On Bitcoin, a public key is not revealed on-chain until its output is spent. Wallets employ hierarchical-deterministic derivation (BIP-32/44-style) and generate a fresh key for each authorisation event, so that each key is exposed exactly once and only briefly before its output is spent. Under Shor's algorithm the risk arises after a public key is revealed; a key used once and never again gives a quantum adversary no reusable target, which is why address reuse, not key revelation alone, is the amplifier that matters. This mitigation is available today and requires no protocol change.

A post-quantum authorisation path via Taproot. SQRL-compatible wallets may construct Taproot outputs that commit to a script path requiring a post-quantum signature (for example an ML-DSA or SLH-DSA signature) over a defined authorisation message. The post-quantum public key, or a commitment to it, is bound to the output at creation time, and the script and signature are revealed only if that path is exercised, so the larger witness is paid for only when it is used. All verification remains deterministic under Bitcoin's existing consensus rules, with no trusted intermediary and no protocol modification, preserving the trustless model. As Bitcoin's own post-quantum roadmap matures, SQRL's execution model does not preclude surfacing post-quantum-verified authorisation as a first-class predicate inside SCL contracts, letting a contract require or branch on quantum-resistant authorisation directly.

The honest summary is that SQRL's ledger integrity is quantum-reliable by construction today, its own endpoints are built to run NIST-standardised post-quantum cryptography, and its Bitcoin-layer authorisation is protected by single-use keys now and by a ready post-quantum script path as the base layer adopts one. No part of this claim rests on cryptography that is not either published by NIST or already standard in Bitcoin.

12 Security Model and Guarantees

Assumptions: the system assumes Bitcoin remains secure, since security derives from Bitcoin's consensus, finality, and availability.

Guarantees:

- Deterministic replay: identical inputs yield identical outputs.
- Objective disputes: correctness is provable from h, the protocol stamp, execution traces, and the block's state root.
- Automatic rejection: invalid transitions are discarded without forks or coordination.
- Replay and front-run resistance: anchor-outpoint binding ties every call envelope to exactly one Bitcoin transaction (Section 6).
- Divergence detection: per-block state roots make any disagreement between nodes detectable and auditable from the confirmed call log.
- Bounded stall exposure: partition-scoped state confines data-withholding to the withholder's own transaction footprint, time-boxed by the void window (Section 7).

- Validator accountability without validator trust: permissioned seats attest to state they cannot falsify undetected, since any observer can replay the same public inputs; a dishonest quorum can disrupt liveness of attestation, never the correctness of state (Section 3.1).
- Subnet integrity without subnet transparency: a closed subnet's history is bound to Bitcoin anchors it cannot rewrite, while its data remains inside the membership boundary (Section 3.2).
- Quantum-reliable integrity: verification of the entire ledger is replay against hash-based commitments, which survive quantum cryptanalysis, while the network's own attestation and transport endpoints run NIST post-quantum cryptography (Section 11).

12.1 Determinism and Reproducibility Testing

SQRL aims to ensure that independent evaluators reach identical results given the same inputs (bytecode and constants, payload, Bitcoin-derived context, and prior state). To support this goal, the reference implementation is expected to maintain:

- Golden test vectors: a suite of canonical contracts and payload sequences with expected outputs/state roots.
- Cross-platform determinism checks: continuous integration runs the same vectors across supported OS/architectures and compares outputs bit-for-bit.
- Fuzzing/property tests: randomised payload generation and contract-state mutation to detect panics, non-deterministic behaviour, or divergence across runs.

12.2 Resource Exhaustion and Denial-of-Service Surfaces

Because execution occurs off-chain, the primary availability risks are resource exhaustion and unbounded growth in locally maintained data. The reference node mitigates these risks through explicit limits and operational controls:

- Execution ceilings: maximum instruction count, stack depth, and call depth per invocation (see Section 4).
- Payload and request limits: maximum accepted payload size and maximum argument serialisation size per API request.
- Rate limiting: per-IP or per-API-key request throttling and backpressure on expensive endpoints.
- State growth controls: configurable retention/indexing policies for payload history, logs, and derived indexes to prevent unbounded disk growth.

These controls do not replace correctness guarantees; they restrict the worst-case cost of evaluation and storage for adversarial workloads.

12.3 Node Trust Boundaries and Verification Responsibilities

For independent verification, it must be clear which components are trusted and which are verified:

- Bitcoin data source: nodes obtain transaction and UTXO information from a configured backend. Regardless of the source, the nodes should verify the critical fields used for execution (for example, the tx outputs containing commitments, the relevant UTXOs, and block confirmation depth).
- Caching vs verification: cached Bitcoin responses and derived indexes are performance optimisations; correctness-critical decisions must remain reproducible from canonical inputs.
- Reorg handling: nodes treat recently seen transactions as pending until a configurable confirmation threshold is reached, and additionally retain undo logs plus a 144-block journal so that shallow reorganisations roll back and replay automatically (Section 6).

12.4 Key Management and Wallet Operational Considerations

The SQRL wallet (and any deployment tooling) may control Bitcoin private keys used to authorise anchoring transactions. Operational security therefore depends on standard key-management practices:

- secure key storage (hardware wallets, OS keychains, or HSMs where applicable);
- protection against malware and clipboard/transaction-substitution attacks; and
- user confirmation of transaction outputs and OP_RETURN contents before broadcast.

For institutional participants, anchoring keys and issuer keys are expected to live in the bank's existing HSM and key-ceremony infrastructure; the protocol imposes no custody model beyond standard Bitcoin key ownership.

12.5 Logging, Monitoring, and Incident Response

Operators should run nodes with:

- structured logs (execution outcome, limit violations, validation failures);
- metrics on CPU, memory, disk growth, and request rates; and
- alerting for abnormal failure rates or sustained resource pressure.

Where logs may contain sensitive data (addresses, payload content), nodes should support configurable redaction and retention policies. The SQRL software provides interfaces that enable the functionality above but does not impose a fixed policy.

13 Cryptographic Guarantees

SQRL leverages strong cryptographic primitives:

- Commitments. $h = \mathrm{H}(C)$ binds contract semantics immutably. Any modification to C changes h with overwhelming probability.
- Collision resistance. Given hash function H (for example, SHA-256), the probability of $C \neq C'$ but $\mathrm{H}(C) = \mathrm{H}(C')$ is negligible ($< 2^{-128}$).
- Authorisation. Payloads are signed by private keys associated with Bitcoin UTXOs. Verification ensures only rightful owners trigger state changes.
- Replay protection. Each call envelope names the unique Bitcoin outpoint its anchoring transaction must spend, binding the envelope to exactly one transaction; together with context binding (Γ includes block height and UTXO state), signatures cannot be replayed, re-anchored, or front-run.

These guarantees yield cryptographic soundness: contracts are immutable, verifiable, and objectively resolvable without coordination. The post-quantum posture that extends these primitives against a future quantum adversary is specified in Section 11.

14 Network Fees

Network usage is metered in XRB, the network token, and every chargeable action is settled by burning it on a fee schedule fixed in the protocol's consensus constants. Four charge classes exist: execution fees on computation above a free envelope that keeps simple token transfers free; anchoring rent, a metered charge paid in XRB each time a subnet registers a checkpoint on the canonical ledger, which is what binds a subnet's confidential activity to the public network and buys its standing to route; value-based settlement fees, levied on batch settlement in basis points of settled value rather than instruction count; and capacity licences, a staked fuel position that sets a deployer's execution capacity above the free tier. The same burn-settled rent that buys routing standing also secures it: a subnet can route across the network only while it holds a current, rent-paid checkpoint, and the validators that attest a cross-subnet settlement hold staked, forfeitable XRB positions, so both the right to move value and the integrity of the record that it moved reduce to XRB that has been burned or staked (Section 3.3). Supply mechanics, the validator emission schedule, the full fee tables, and the demand model are specified in the companion SQRL tokenomics paper rather than here.

15 Performance Considerations

Because inputs and context are fixed and public, histories can be replayed offline and batched. Near-fully batched payloads can yield orders-of-magnitude throughput gains when amortised over a single Bitcoin transaction (for example, an illustrative 7 $\rightarrow$ 70,000 TPS under 100% batching). These represent theoretical upper bounds and depend on workload characteristics. For the bank workload specifically, the relevant figures are the batch settlement call (up to roughly 9,000 transfers per anchored call, Section 8.4) and Lightning's instant off-chain transfer rate, which is bounded by hub capacity rather than by Bitcoin.

15.1 Commitment Storage

SQRL stores contract commitments as 32-byte hashes embedded in Bitcoin transactions via OP_RETURN outputs. This minimal footprint ensures immutable anchoring while respecting network constraints.

The choice of OP_RETURN has historically been debated, particularly regarding the data_carrier_limit, which caps the maximum size of arbitrary data per output. By restricting commitments to 32 bytes, SQRL minimises resource usage while maintaining robust cryptographic guarantees. This approach aligns with Satoshi Nakamoto's original design principles and imposes negligible burden on the Bitcoin network, demonstrating that compact commitments can coexist safely with global consensus operations. Interface-defined compact operations (Section 7) use the same footprint budget to carry the operation itself, at most 43 bytes, in place of a hash.

16 Conclusion

SQRL demonstrates that deterministic, resource-bounded smart contracts can be securely anchored to Bitcoin without introducing the inefficiencies of general-purpose on-chain virtual machines. By restricting the SCL language to statelessly verifiable primitives, bounded control flow, and explicitly defined resource ceilings, the ledger guarantees termination, determinism, and replayability. Anchoring is end-to-end: contract deployments and every subsequent call are committed on Bitcoin, with anchor-outpoint binding closing replay and front-running structurally, per-block state roots making divergence detectable at a glance, and partition-scoped state confining data-withholding to the withholder's own transaction footprint.

On this substrate, the paper has specified a settlement network shaped for the institutions actually adopting tokenised money: permissioned execution and governance above open settlement. Named validator seats make the execution layer accountable; closed subnets give banks confidentiality by data distribution while Bitcoin anchoring makes their histories unforgeable; cross-subnet value routes through the canonical ledger, where standing is bought with burned XRB and settlement is attested by bonded seats; supervisors get continuous, replayable visibility through observer nodes; and the Lightning layer, with multi-asset channels and in-flight FX through AMM hubs, gives bank-issued instruments an open distribution network over Lightning that no interbank ledger offers. Payments stay instant on Lightning while net positions settle with finality on Bitcoin through splicing, without the channels closing. The network is quantum-reliable by construction, with ledger integrity resting on hash-based commitments and its own endpoints built to run NIST-standardised post-quantum cryptography. Network usage is metered in XRB and settled by burning it, with the full economic model specified in the companion tokenomics paper.

The design yields a deliberate pair of trade-offs. SQRL forgoes Turing completeness in favour of safety, efficiency, and predictable execution, which is the correct trade for a payments workload. And the network forgoes open execution in favour of accountable execution, while refusing to gate the two layers that matter most: settlement on Bitcoin, and the token market. Permissioned execution on open settlement is the whole design in one line.

Future work will focus on extending the catalogue of standard interfaces beyond utxo_token_v1, the phased privacy roadmap of Section 10, the staged rollout of the post-quantum profile of Section 11, activating the published fuel schedule and fee parameters described in the companion tokenomics paper, formalising a proof-carrying execution model, and deepening Lightning-native asset and FX infrastructure.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] P. Todd, "Closed Seal Sets and Truth Lists for Better Privacy and Censorship Resistance," 2016. Available at: https://petertodd.org/2016/closed-seal-sets-and-truth-lists-for-privacy.

[3] M. Orlovsky, "Scalable Consensus for Client-side Validated Smart Contracts," 2025. Available at: https://github.com/RGB-WG/yellowpaper/blob/4c18a6ac346a4dd379766f0bf2fcc02a3ca7e722/rgb-yellowpaper.pdf.

[4] National Institute of Standards and Technology, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard," 2024. Available at: https://doi.org/10.6028/NIST.FIPS.203.

[5] National Institute of Standards and Technology, "FIPS 204: Module-Lattice-Based Digital Signature Standard," 2024. Available at: https://doi.org/10.6028/NIST.FIPS.204.

[6] National Institute of Standards and Technology, "FIPS 205: Stateless Hash-Based Digital Signature Standard," 2024. Available at: https://doi.org/10.6028/NIST.FIPS.205.