Teleport Universal Bridge Architecture: Frozen Wire Format, Pluggable Chain Adapters, and ~~Multi-Tenant Deployment~~

Zach Kelling

Lux Industries

2025

Abstract

We describe the engineering architecture of Lux Teleport, a universal cross-chain bridge serving 270+ chains through a single protocol binary. The design separates concerns into four layers: (1) a *frozen wire protocol* comprising four canonical proof tags—`DEPOSIT`, `YIELD`, `BACKING`, `RELEASE`—encoded as `bytes32`-tagged `abi.encode` payloads with MPC-signed proofs, per-`srcChainId` replay protection, and monotonic backing attestations with configurable staleness windows; (2) a *chain-agnostic Teleporter contract* that implements the wire protocol against an abstract chain adapter interface, enabling a single audited contract to serve all EVM deployments; (3) a *pluggable chain adapter* layer supporting four support classes—Lux-native (Warp, no MPC hop), external EVM (`TeleportVault` template), Bitcoin-family script chains (watcher-first: OP_NET, Litecoin, Dogecoin), and non-EVM account-model chains (Solana Ed25519, TON Ed25519, XRP secp256k1); and (4) a *multi-tenant architecture* where hostname-based routing, per-tenant manifests, and shared MPC infrastructure enable white-label bridge deployments from a single binary. We specify the threshold signing interface unifying CGGMP21 (ECDSA/secp256k1 for EVM), FROST (Schnorr for Bitcoin Taproot), and BLS (Lux Warp internal) under `github.com/luxfi/crypto/threshold`. We formalize the security model including per-chain confirmation depths, backing attestation monotonicity, sequential withdraw nonces, peg oracle auto-pause thresholds, and hot wallet / MPC key separation. We describe the yield composition pipeline through which bridged assets earn layered returns via Babylon staking, D-Chain LP, and bridge fees through the xLUX instrument. Finally, we specify the 6-artifact conformance testing framework that gates chain onboarding through CI/CD.

Contents

1 Introduction

Cross-chain bridge protocols face a tension between generality and security. A bridge that supports n chains requires $O(n)$ contract implementations, each a distinct attack surface. Teleport resolves this by separating the protocol into a chain-independent wire format and a chain-dependent adapter layer, connected through a narrow typed interface.

Three prior Lux papers define components used here: Lux Teleport Protocol [1] established lock-and-mint semantics, Lux Bridge [2] formalized threshold MPC custody, and Lux Warp Messaging [3] specified intra-ecosystem BLS messaging. This paper unifies these into a single deployable architecture and adds four contributions:

1. A frozen wire format with exactly four proof tags, audited once and never modified (§2).
2. A chain adapter taxonomy with formal adapter interface contracts (§4).
3. A multi-tenant deployment model for white-label bridge instances (§7).
4. A conformance test framework that enforces correctness at onboarding time (§9).

2 Wire Protocol

The wire protocol defines the canonical format for all cross-chain proofs. It is frozen: no new tags, no field reordering, no encoding changes. Upgrades occur exclusively at the adapter layer.

2.1 Proof Tags

Every proof is a `bytes32`-tagged ABI-encoded payload signed by the MPC threshold committee. Four tags exist:

Definition 1 (Canonical Proof Tags)**.**

$$\mathsf{DEPOSIT} = \textit{keccak256("DEPOSIT")} \tag{1}$$
$$\mathsf{YIELD} = \textit{keccak256("YIELD")} \tag{2}$$
$$\mathsf{BACKING} = \textit{keccak256("BACKING")} \tag{3}$$
$$\mathsf{RELEASE} = \textit{keccak256("RELEASE")} \tag{4}$$

These four values are compile-time constants in the Teleporter contract. No mechanism exists to add, remove, or modify tags post-deployment.

2.2 Proof Encoding

Each proof π consists of a tag, a payload, and a threshold signature:

Definition 2 (Wire Proof Format)**.**

$$\pi = \big(\mathsf{tag},\ \mathsf{payload},\ \sigma\big) \tag{5}$$

where:

$$\mathsf{payload} = \textit{abi.encode}(\mathsf{tag}, C_s, \nu, T, r, a, \mathsf{extra}) \tag{6}$$
$$\sigma = \mathrm{ThresholdSign}_{t,n}\big(H(\mathsf{payload})\big) \tag{7}$$

C_s is the source chain identifier, ν is the operation nonce, T is the token address, r is the recipient, a is the amount, and extra *is a tag-specific extension field (empty for `DEPOSIT` and `RELEASE`).*

The hash function H is chain-determined: `keccak256` for EVM targets, SHA-256 for Bitcoin-family, and the native hash for other environments. The adapter layer handles this mapping (see §4).

2.3 Tag Semantics

DEPOSIT. Attests that a user locked a units of asset T on source chain C_s, destined for recipient r on the destination chain. The destination Teleporter verifies σ and mints wrapped tokens.

$$m_{\mathsf{dep}} = H\big(\mathsf{DEPOSIT} \,\|\, C_s \,\|\, \nu \,\|\, T \,\|\, r \,\|\, a\big) \tag{8}$$

YIELD. Attests that accrued yield δ has increased the share price of bridge token T. The extra field carries the new share price numerator p' and the epoch index e.

$$m_{\mathsf{yld}} = H\big(\mathsf{YIELD} \,\|\, C_s \,\|\, \nu \,\|\, T \,\|\, p' \,\|\, e\big) \tag{9}$$

BACKING. Attests that total backing B exists for token T on source chain C_s at timestamp τ. The Teleporter enforces that τ is strictly greater than any previously accepted τ (monotonicity) and that $\tau \geq \text{block.timestamp} - \Delta_{\mathsf{stale}}$ (staleness bound, default $\Delta_{\mathsf{stale}} = 7200\,\text{s}$).

$$m_{\mathsf{back}} = H\big(\mathsf{BACKING} \,\|\, T \,\|\, B \,\|\, \tau \,\|\, C_s\big) \tag{10}$$

Invariant 1 (Backing Monotonicity)**.** *For successive backing proofs π_k, π_{k+1} on a given token:*

$$\tau_{k+1} > \tau_k \quad \textit{and} \quad \tau_{k+1} \geq \text{block.timestamp} - \Delta_{\mathsf{stale}} \tag{11}$$

RELEASE. Attests that the MPC authorizes releasing a units of underlying asset T to recipient r' on source chain C_s. Issued after a user burns wrapped tokens on a destination chain.

$$m_{\mathsf{rel}} = H\big(\mathsf{RELEASE} \,\|\, C_d \,\|\, \nu' \,\|\, T \,\|\, r' \,\|\, a\big) \tag{12}$$

The distinct tag prefixes in Equations (8)–(12) guarantee domain separation: a signature valid for `DEPOSIT` cannot verify as `RELEASE`.

2.4 Replay Protection

Each Teleporter contract maintains a per-source-chain nonce bitmap:

$$\mathcal{N} : \mathbb{Z}_{\mathsf{chainId}} \times \mathbb{Z}_{\mathsf{nonce}} \to \{0, 1\} \tag{13}$$

The verify-and-mark operation executes atomically within a single transaction:

Algorithm 1 Atomic Nonce Check-and-Set

Require: Source chain C_s, nonce ν

1: **assert** $\mathcal{N}(C_s, \nu) = 0$ ▷ Not yet processed
2: $\mathcal{N}(C_s, \nu) \leftarrow 1$

Nonces are *not* required to be sequential for `DEPOSIT` operations (out-of-order relay is permitted). Withdraw nonces for `RELEASE` operations *are* sequential per destination chain, preventing front-running reordering of withdrawals.

Property 1 (Sequential Withdraw Nonces)**.** *For `RELEASE` operations on chain C, the Teleporter enforces:*

$$\nu'_{k+1} = \nu'_k + 1 \tag{14}$$

where ν'_k is the last processed release nonce for chain C.

3 Teleporter Contract

The Teleporter is a chain-agnostic contract that implements the wire protocol. One audited Solidity implementation is deployed across all EVM chains. Non-EVM chains receive functionally identical implementations in their native languages, each tested against the same conformance suite (§9).

3.1 Contract Interface

Listing 1: Core Teleporter interface (simplified)

```
interface ITeleporter {
    // User-facing
    function deposit(
        address token,
        uint256 amount,
        uint64  dstChainId,
        bytes32 recipient
    ) external;

    function withdraw(
        bytes32 tag,    // RELEASE only
        uint64  srcChainId,
        uint256 nonce,
        address token,
        address recipient,
        uint256 amount,
        bytes   calldata proof
    ) external;

    // MPC-submitted
    function submitBacking(
        address token,
        uint256 backing,
        uint256 timestamp,
        uint64  srcChainId,
        bytes   calldata proof
    ) external;

    function submitYield(
        address token,
        uint256 newSharePrice,
        uint256 epoch,
        uint64  srcChainId,
        bytes   calldata proof
    ) external;

    // Views
    function isNonceUsed(uint64 srcChainId, uint256 nonce)
        external view returns (bool);
    function backingRatio(address token)
        external view returns (uint256);
}
```

3.2 State Variables

The contract holds five categories of state:

1. **MPC public key** — set once at deployment, rotated via 7-day timelock.
2. **Nonce bitmaps** — `mapping(uint64 => mapping(uint256 => bool))`, one per source chain.
3. **Release counters** — `mapping(uint64 => uint256)`, enforcing Property 1.
4. **Backing state** — per-token latest backing amount, timestamp, and derived pause flag.
5. **Yield state** — per-token share price numerator and epoch index.

The contract is non-upgradeable. No proxy pattern, no `delegatecall`, no admin functions beyond timelocked MPC key rotation.

4 Chain Support Classes

Teleport classifies connected chains into four support classes. Each class defines an adapter interface that translates the wire protocol into chain-native operations.

4.1 Class A: Lux-Native (No MPC Hop)

Transfers between Lux appchains use Warp Messaging [3] with BLS12-381 aggregate signatures. No MPC threshold ceremony is required—the source appchain's validator set signs the Warp message directly.

Property 2 (Warp Bypass)**.** *For transfers where both C_s and C_d are Lux appchains, the Teleporter verifies BLS aggregate signatures via the* `0x0200000005` *precompile rather than MPC ECDSA signatures via* `ecrecover`*.*

Latency: sub-second (limited by appchain block time). Cost: no external gas, precompile verification only.

4.2 Class B: External EVM (TeleportVault Template)

External EVM chains (Ethereum, Arbitrum, Optimism, Base, BSC, Polygon, etc.) deploy a `TeleportVault` contract from a shared template. The template provides:

- ERC-4626 compatible vault for lock/release accounting
- `ecrecover`-based MPC signature verification (CGGMP21 / secp256k1)
- Per-token daily mint limits and global pause
- Yield strategy integration hooks (see §8)

Listing 2: TeleportVault lock function (simplified)

```
function lock(
    address token,
    uint256 amount,
    uint64  dstChainId,
    bytes32 recipient
) external nonReentrant whenNotPaused {
    IERC20(token).safeTransferFrom(
        msg.sender, address(this), amount
    );
    uint256 nonce = ++lockNonces[dstChainId];
    emit Lock(
        block.chainid, dstChainId,
        nonce, token, recipient, amount
    );
}
```

Confirmation depth before the MPC signs: configurable per chain, defaults listed in Table 3.

4.3 Class C: Bitcoin-Family / Script Chains (Watcher-First)

Bitcoin, OP_NET, Litecoin, and Dogecoin lack smart contract runtimes capable of verifying threshold signatures on-chain. The adapter uses a *watcher-first* architecture:

1. **Deposit path**: A watcher service monitors the chain for deposits to a FROST-derived Taproot address. Upon reaching confirmation depth (6 blocks for Bitcoin, 12 for Litecoin, 40 for Dogecoin), the watcher submits a deposit attestation to the MPC cluster.
2. **Release path**: The MPC cluster constructs and signs a Bitcoin transaction using FROST (Schnorr/BIP-340) threshold signing, producing a valid Taproot spend.

Property 3 (Script Chain Asymmetry)**.** *On Class C chains, deposits are permissionless (any user sends to the Taproot address) but releases require MPC coordination. There is no on-chain contract to enforce nonces; the MPC cluster maintains canonical nonce state.*

For OP_NET specifically, the watcher interfaces with the OP_NET runtime's AssemblyScript contracts, which can emit structured events but cannot verify external signatures. The adapter treats OP_NET as a script chain with event-emitting capability.

4.4 Class D: Non-EVM Account-Model Chains

Solana, TON, and XRP use account-model architectures with non-EVM signature schemes.

Chain	Curve	Threshold Protocol	Verify Method
Solana	Ed25519	FROST	`Ed25519SigVerify` precompile
TON	Ed25519	FROST	`check_signature` TVM opcode
XRP	secp256k1	CGGMP21	Native transaction signing
Sui	Ed25519	FROST	`ed25519::verify` Move
Aptos	Ed25519	FROST	`ed25519::verify` Move
Cosmos	secp256k1	CGGMP21	`VerifySignature` SDK

Table 1: Non-EVM chain adapter configurations

Each Class D adapter implements the same four-tag wire protocol. The hash function H and signature encoding vary per chain but the proof semantics remain identical.

5 Threshold Signing Architecture

The MPC cluster exposes a unified signing interface regardless of the underlying cryptographic protocol. The interface is defined at `github.com/luxfi/crypto/threshold`.

5.1 Protocol Selection

Definition 3 (Unified Threshold Interface)**.** *The threshold signing interface exposes three operations:*

- $\text{KeyGen}(t, n, \mathsf{curve}) \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^{n})$
- $\text{Sign}(m, S) \to \sigma$ *where* $|S| \geq t$
- $\text{Verify}(\mathsf{pk}, m, \sigma) \to \{0, 1\}$

The curve *parameter selects the concrete protocol:*

Protocol	Curve	Chains	Signature Size
CGGMP21	secp256k1	EVM, XRP, Cosmos	65 bytes (r,s,v)
FROST	Ed25519	Solana, TON, Sui, Aptos	64 bytes (R,s)
FROST	secp256k1 (BIP-340)	Bitcoin Taproot	64 bytes (Schnorr)
BLS	BLS12-381	Lux Warp internal	48 bytes (compressed)

Table 2: Threshold protocol selection by chain family

5.2 CGGMP21 for EVM Chains

EVM chains verify signatures via the `ecrecover` precompile, which expects ECDSA signatures on the secp256k1 curve. CGGMP21 [7] provides threshold ECDSA with identifiable abort: if a participant deviates from the protocol, the honest majority can identify the cheater and exclude them.

Key properties:

- t-of-n threshold with $t = 2, n = 3$ default (production configuration).
- 4-round signing protocol (can be preprocessed to 1 effective round).
- Signatures are indistinguishable from standard ECDSA: no on-chain changes required.
- Forgery probability: $2^{-139.6}$ for $t = 67$-of-$n = 100$ [2].

5.3 FROST for Bitcoin Taproot and EdDSA Chains

FROST [8] provides threshold Schnorr signatures. For Bitcoin Taproot (BIP-340), FROST produces 64-byte Schnorr signatures that spend from a Taproot address derived from the group public key. For Ed25519 chains, FROST produces standard EdDSA signatures.

Property 4 (FROST Two-Round Signing)**.** *FROST signing completes in exactly 2 rounds:*

1. *Commitment round: each signer broadcasts* (D_i, E_i).
2. *Signature round: each signer produces partial signature* z_i*; aggregator computes* $\sigma = (R, s)$.

5.4 BLS for Lux Warp

Intra-Lux transfers bypass the MPC cluster entirely. Appchain validators sign Warp messages using their BLS12-381 keys. The destination appchain verifies the aggregate signature via precompile, checking that the signing set represents $\geq 2/3$ of the source appchain's staked weight.

5.5 Implementation: `luxfi/crypto/threshold`

The Go implementation provides a single entry point:

Listing 3: Threshold signing entry point

```go
package threshold

type Signer interface {
    KeyGen(ctx context.Context, t, n int,
           curve Curve) (*KeyShare, error)
    Sign(ctx context.Context, msg []byte,
         participants []PartyID) (*Signature, error)
}

type Curve int
const (
    Secp256k1 Curve = iota // CGGMP21
    Ed25519                // FROST
    BIP340                 // FROST (Schnorr)
    BLS12381               // BLS aggregate
)
```

The caller specifies the curve; the library selects the protocol. No protocol-specific API is exposed to bridge logic.

6 Security Model

6.1 Confirmation Depths

The MPC cluster waits for chain-specific confirmation depths before signing a `DEPOSIT` proof. These depths are chosen to make chain reorganization economically infeasible.

Chain	Depth	Rationale
Bitcoin	6	∼60 min; 51% attack cost > $10B/hr
Litecoin	12	∼30 min; proportional to hashrate
Dogecoin	40	∼40 min; lower hashrate margin
Ethereum	64	∼13 min; 2 finalized epochs
Arbitrum	1	Sequencer + L1 finality inherited
Optimism	1	Sequencer + L1 finality inherited
Base	1	Sequencer + L1 finality inherited
BSC	15	∼45 sec; fast finality mode
Polygon PoS	128	∼4 min; probabilistic finality
Solana	32	∼13 sec; confirmed commitment
TON	1	Instant finality (masterchain)
Lux (Warp)	0	Sub-second finality; Warp proof suffices

Table 3: Confirmation depth configuration per chain

6.2 Backing Attestation

The MPC cluster periodically attests to total backing. The Teleporter enforces three invariants on backing proofs:

Invariant 2 (Monotonic Timestamp)**.** $\tau_{k+1} > \tau_k$ *for successive backing proofs on a given token.*

Invariant 3 (Staleness Bound)**.** $\tau \geq \text{block.timestamp} - \Delta_{\mathsf{stale}}$ *where* $\Delta_{\mathsf{stale}} = 7200\,s$ *(2 hours) by default.*

Invariant 4 (Auto-Pause Threshold)**.** *If the backing ratio* $B/\text{totalMinted}(T) < 0.985$*, the Teleporter sets* $\mathsf{paused} \leftarrow \mathsf{true}$*. No* `DEPOSIT` *or* `YIELD` *operations are processed while paused. Unpausing requires a separate MPC-signed proof with a fresh backing attestation showing ratio* ≥ 1.0*.*

6.3 Sequential Withdraw Nonces

Deposit nonces use a bitmap and may arrive out of order. Withdraw nonces are strictly sequential (Property 1). This prevents an attacker who observes multiple pending `RELEASE` proofs from selectively replaying a subset in a different order to manipulate vault accounting.

6.4 Peg Oracle

An independent peg oracle monitors the price ratio of each wrapped token against its underlying on external markets. If the peg deviates beyond a configurable threshold (default: 2% for stablecoins, 5% for volatile assets), the oracle submits a pause transaction.

The peg oracle is defense-in-depth: it triggers only if the on-chain backing attestation fails to catch an undercollateralization event. The oracle runs as a separate process with its own signing key, distinct from the MPC committee.

6.5 Key Separation

Definition 4 (Key Hierarchy)**.** *The system maintains three distinct key classes:*

1. ***MPC threshold key****: distributed across n signer nodes; used exclusively for wire protocol proofs. Never held by a single entity.*
2. ***Hot wallet keys****: per-chain operational keys for gas payment, relay submission, and watcher operations. Individually compromisable without bridge fund risk.*
3. ***Admin timelock key****: controls MPC public key rotation with a 7-day delay. Held by a separate multisig.*

Compromise of hot wallet keys cannot produce valid wire protocol proofs. Compromise of the admin key introduces a 7-day window for detection and response.

6.6 Security Properties

Theorem 1 (Wire Protocol Soundness)**.** *Under the ECDLP assumption on secp256k1, the DLP assumption on Ed25519, and the co-CDH assumption on BLS12-381, an adversary controlling fewer than t MPC signer nodes cannot produce a valid proof π for any tag.*

Proof sketch. Each tag's proof requires a valid threshold signature. By the unforgeability of CG-GMP21 [7] and FROST [8] under their respective hardness assumptions, an adversary with $< t$ shares cannot produce a valid signature. Domain separation via distinct `bytes32` tags ensures that a signature valid for one tag cannot verify under another. The per-`srcChainId` nonce bitmap prevents replay. See formal proofs in [6]. □

Theorem 2 (Conservation)**.** *For any token T and any execution trace, the total supply of wrapped tokens across all destination chains does not exceed the total backing on source chains.*

Proof sketch. Minting requires a valid `DEPOSIT` proof with unused nonce. Backing attestation with auto-pause ensures that if backing drops below 98.5% of minted supply, minting halts. Sequential release nonces prevent double-withdraw. See formal proof in [6]. □

7 Multi-Tenant Architecture

Teleport supports white-label deployment: multiple independent bridge brands operate from a single binary, sharing MPC infrastructure while maintaining distinct identities, fee structures, and chain sets.

7.1 Routing

Incoming requests are routed by hostname:

Listing 4: Hostname-based tenant routing

```
func (s *Server) resolveTenant(
    r *http.Request,
) (*TenantConfig, error) {
    host := r.Host
    cfg, ok := s.tenants[host]
    if !ok {
        return nil, ErrUnknownTenant
    }
    return cfg, nil
}
```

Each tenant hostname (e.g., `bridge.lux.network`, `bridge.example.com`) maps to a `TenantConfig` specifying allowed chains, fee parameters, branding assets, and yield strategy set.

7.2 Tenant Manifest

Listing 5: Tenant manifest example

```
{
  "id": "tenant-lux",
  "hostnames": ["bridge.lux.network"],
  "chains": [1, 56, 137, 42161, 10, 8453, 96369],
  "fee_bps": 30,
  "stakeholder_bps": 5000,
  "yield_strategies": ["babylon", "dchain-lp"],
  "shariah_mode": false,
  "mpc_group": "default"
}
```

The `mpc_group` field selects which MPC key set to use. Multiple tenants can share the same MPC group (shared security) or use dedicated groups (isolated key material).

7.3 Isolation Guarantees

Property 5 (Tenant Isolation)**.** *Tenants share MPC signing infrastructure but are isolated in:*

1. *Nonce namespaces: each tenant's nonce bitmaps are independent.*
2. *Fee accounting: bridge fees accrue to tenant-specific addresses.*
3. *Chain access: a tenant can only bridge to chains listed in its manifest.*
4. *Rate limits: per-tenant daily volume caps.*

The on-chain Teleporter contract does not encode tenant identity. Tenant isolation is enforced at the API and MPC relay layer. The contract's security properties (soundness, conservation) hold regardless of tenant configuration.

8 Yield Composition

Bridged assets are not idle. Locked assets on source chains are deployed to yield strategies, and the returns are reflected in bridge token share prices on destination chains via the `YIELD` wire tag.

8.1 Yield Layers

Teleport composes yield from three sources:

1. **Babylon staking**: BTC locked on Bitcoin earns yield through Babylon's Bitcoin staking protocol. The MPC cluster delegates staked BTC to Babylon validators and collects rewards.
2. **D-Chain LP**: Assets bridged to Lux D-Chain (the native DEX chain) are deployed to automated market maker pools, earning trading fees.
3. **Bridge fees**: A portion of bridge fees (configurable per tenant) is distributed pro-rata to bridge token holders.

8.2 Share Price Mechanism

Bridge tokens on destination chains use a share-price model:

$$\text{value}(s) = s \times \frac{p'}{10^{18}} \tag{15}$$

where s is the number of bridge token shares held and p' is the current share price numerator. The share price only increases—the protocol never performs negative rebases.

When yield accrues, the MPC attests to the new share price:

Algorithm 2 Yield Attestation

Require: Token T, new backing B', epoch e

1: $p' \leftarrow B' \times 10^{18}/\text{totalShares}(T)$
2: **assert** $p' \geq p_{\text{prev}}$ ▷ Monotonic share price
3: $\sigma \leftarrow \text{ThresholdSign}(H(\mathsf{YIELD} \| C_s \| \nu \| T \| p' \| e))$
4: Submit $(\mathsf{YIELD}, C_s, \nu, T, p', e, \sigma)$ to destination Teleporters

8.3 xLUX

The xLUX instrument represents a claim on all three yield layers for LUX-denominated bridge positions. xLUX is minted when a user bridges LUX and opts into yield; the xLUX share price reflects accumulated Babylon rewards, D-Chain LP fees, and bridge fee distributions.

$$\text{APY}_{\mathsf{xLUX}} = r_{\mathsf{babylon}} + r_{\mathsf{dchain}} + r_{\mathsf{fees}} - r_{\mathsf{slashing}} \tag{16}$$

where r_{slashing} accounts for potential Babylon slashing events. The backing attestation mechanism (§6) ensures that any slashing event is reflected in the share price within Δ_{stale} seconds.

9 Conformance Testing

Adding a new chain to Teleport requires producing 6 artifacts that are validated by CI/CD before the chain is activated in production.

9.1 Onboarding Kit

#	Artifact
1	**Chain adapter implementation**: source code implementing the adapter interface for the target chain.
2	**Contract deployment script**: idempotent deployment of the Teleporter (or TeleportVault) contract with constructor arguments matching the wire protocol.
3	**Conformance test results**: passing output from the 47-case conformance test suite covering all four tags, nonce replay, backing monotonicity, staleness rejection, and auto-pause.
4	**Confirmation depth justification**: written analysis of the chain's finality model with recommended confirmation depth and attack cost estimates.
5	**Watcher configuration** (Class C/D only): configuration for chain-specific event monitoring, including block polling interval, event signatures, and confirmation logic.
6	**Gas cost profile**: measured gas costs for verify, mint, burn, and release operations on the target chain, with comparison to the reference EVM implementation.

Table 4: Required onboarding artifacts per chain

9.2 Conformance Test Suite

The test suite exercises all wire protocol operations against a chain-specific adapter:

1. **Tag verification** (4 tests): each of the four tags verifies correctly with a valid MPC signature.
2. **Tag rejection** (12 tests): each tag rejects signatures with wrong tag, wrong chain, wrong nonce, expired staleness, invalid signature, and insufficient backing.
3. **Nonce replay** (4 tests): resubmitting a processed nonce reverts.
4. **Sequential release nonces** (4 tests): out-of-order release nonces revert.
5. **Backing monotonicity** (4 tests): non-monotonic timestamps revert.
6. **Auto-pause** (6 tests): undercollateralization triggers pause; recovery requires fresh backing proof.
7. **Yield monotonicity** (4 tests): share price decrease reverts.
8. **Domain separation** (9 tests): cross-tag signature reuse fails for all tag pairs.

Total: 47 test cases. All must pass before a chain adapter is merged.

9.3 CI/CD Gating

The conformance suite runs in CI on every pull request that modifies a chain adapter. A chain adapter cannot be deployed to production unless:

1. All 47 conformance tests pass against a local testnet or devnet instance of the target chain.
2. Gas cost profile shows no operation exceeding 2× the reference EVM cost (normalized by chain gas pricing).

3. Confirmation depth justification is reviewed and approved by at least two engineers.

10 Related Work

LayerZero. Ultra Light Node architecture with configurable oracles and relayers [9]. Teleport differs in using frozen wire format (no per-deployment configuration of proof structure) and threshold MPC rather than oracle-relayer separation.

Wormhole. Guardian network with 19-of-19 multisig [10]. Teleport uses t-of-n threshold cryptography with $t < n$, providing liveness under $n - t$ node failures.

Across. Optimistic bridge with intent-based settlement [11]. Complementary approach: fast settlement but 7-day fraud proof windows for worst case. Teleport provides MPC-attested proofs with no fraud proof delay.

Axelar. Proof-of-stake validation with threshold ECDSA [12]. Similar threshold approach but Axelar runs its own consensus chain. Teleport leverages existing chain finality.

11 Conclusion

Teleport's architecture separates a frozen wire protocol from pluggable chain adapters, enabling support for 270+ chains without per-chain protocol changes. The four-tag proof format (`DEPOSIT`, `YIELD`, `BACKING`, `RELEASE`) is audited once and deployed everywhere. The unified threshold signing interface abstracts over CGGMP21, FROST, and BLS, selecting the correct protocol per chain family. Multi-tenant deployment enables white-label bridge instances from a single binary. The conformance test framework enforces protocol correctness at chain onboarding time.

This paper specifies the engineering architecture. Formal security proofs are provided in [6]. The Teleport Omnichain paper [4] covers sovereign governance and Shariah compliance. The Lux Universal Threshold Signatures paper [5] provides cryptographic analysis of the underlying threshold protocols.

References

[1] Z. Kelling and V. Seesahai. Teleport Protocol: Trustless Cross-Chain Asset Transfer via Light Client Verification. Lux Industries, Inc., 2022.

[2] Z. Kelling and V. Seesahai. Lux Bridge: Threshold-Secured Cross-Chain Communication with MPC Custody, Post-Quantum Verification, and Formally Verified Contract Invariants. Lux Industries, Inc., 2023.

[3] Z. Kelling. Lux Warp Messaging: Cross-Appchain Communication with BLS Aggregate Signatures. Lux Industries, Inc., 2023.

[4] Z. Kelling and V. Seesahai. Lux Teleport: Sovereign Omnichain Liquidity via Threshold Cryptography and Per-Chain Governance. Lux Industries, Inc., 2022 (Revised December 2025).

[5] Z. Kelling and V. Seesahai. Universal Threshold Signatures: Multi-Chain Cryptographic Infrastructure with Post-Quantum Security. Lux Industries, Inc., 2021.

[6] Z. Kelling. Formal Verification: Teleport Bridge Protocol. Lux Industries, Inc., 2024. Lean 4 proofs.

[7] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *ACM CCS*, 2020.

[8] C. Komlo and I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *SAC*, 2020.

[9] R. Zarick, B. Pellegrino, and C. Bhartia. LayerZero: An Omnichain Interoperability Protocol. LayerZero Labs, 2021.

[10] Wormhole Foundation. Wormhole: Generic Message Passing Protocol. 2022.

[11] UMA Protocol. Across: Intent-Based Cross-Chain Bridge. 2023.

[12] S. Lau et al. Axelar: Connecting Web3. Axelar Network, 2021.