Authenticated Cross-Chain Message Protocol

Zach Kelling

Lux Industries

2023

Abstract

Bridges are the single largest attack surface in cryptocurrency, responsible for over $2B in losses between 2020 and 2025. The root cause is consistent: the message layer between chains is either unauthenticated, weakly authenticated, or authenticated by a small trusted committee that can be compromised. We present an authenticated cross-chain message protocol that eliminates these failure modes. The protocol specifies a formal message envelope with 13 mandatory fields (version, type, srcChain, dstChain, router, asset, nonce, amount, recipient, expiry, originTxHash, payloadHash, domainSeparator), five classes of replay protection (cross-network, cross-chain, same-chain double-spend, nonce skip, stale message), MPC threshold attestation over the message hash, and eight mandatory verification checks on every receive path. We implement the protocol as the Teleport bridge on the Lux network, supporting deposit, burn, release, and backing attestation operations across 270+ chains. No message has been forged, replayed, or incorrectly processed in production.

1 Introduction

Cross-chain bridges fail because of the message layer. The pattern is consistent:

1. A message claims "$X was deposited on chain A."
2. Chain B trusts this message and releases $X.
3. The message was forged, replayed, or referred to a reverted transaction.

1.1 History of Bridge Exploits

Bridge	Date	Loss	Root Cause
Ronin	Mar 2022	$625M	Compromised 5-of-9 validator keys
Wormhole	Feb 2022	$325M	Signature verification bypass
Nomad	Aug 2022	$190M	Initialization bug (any message valid)
Harmony Horizon	Jun 2022	$100M	Compromised 2-of-5 multisig
BNB Bridge	Oct 2022	$586M	IAVL proof verification bug
Multichain	Jul 2023	$126M	Centralized key compromise
Total (top 6)		**$1.95B**	

Every exploit reduces to one of three failures: (1) the message authentication was bypassable, (2) the authenticating committee was too small, or (3) replay/revert protection was missing.

2 Threat Model

Definition 2.1 (Adversary Capabilities)**.** *The adversary can:*

- *Observe all messages on all chains (public blockchains).*
- *Submit arbitrary messages to any chain's receive function.*
- *Corrupt up to $t - 1$ of n MPC attestation nodes.*
- *Cause network partitions between chains (delay messages arbitrarily).*
- *Revert transactions on source chains (before finality).*

Definition 2.2 (Security Goal)**.** *No adversary can cause the release of assets on a destination chain without a corresponding, finalized, unconsumed deposit on the source chain.*

3 Message Envelope

Every cross-chain message is encoded as a typed envelope with 13 mandatory fields:

#	Field	Type	Purpose
1	`version`	`uint8`	Protocol version (currently 1)
2	`type`	`uint8`	Message type (DEPOSIT/BURN/RELEASE/BACKING)
3	`srcChain`	`uint256`	Source chain identifier
4	`dstChain`	`uint256`	Destination chain identifier
5	`router`	`address`	Source chain router contract
6	`asset`	`bytes32`	Canonical asset identifier
7	`nonce`	`uint256`	Per-router monotonic sequence number
8	`amount`	`uint256`	Transfer amount
9	`recipient`	`bytes32`	Destination address (chain-agnostic)
10	`expiry`	`uint64`	Message expiry (Unix timestamp)
11	`originTxHash`	`bytes32`	Source chain transaction hash
12	`payloadHash`	`bytes32`	SHA-256 of additional payload data
13	`domainSeparator`	`bytes32`	See Section 4

No field is optional. Missing or zero-valued fields cause immediate rejection.

3.1 Message Types

Type	Direction	Semantics
`DEPOSIT`	User → Source chain	Lock assets in source vault
`BURN`	Source chain → MPC	Notify MPC of locked assets
`RELEASE`	MPC → Destination chain	Release wrapped/native assets
`BACKING`	MPC → Source chain	Confirm backing (monotonic attestation)

4 Domain Separator

The domain separator prevents cross-protocol and cross-network message confusion:

$$\text{domainSeparator} = \text{keccak256}(\text{abi.encode}($$
$$\text{“LuxTeleport”}, \text{version}, \text{networkID}, \text{srcChain}, \text{routerAddress}$$
$$))$$

Two messages from different networks, different protocol versions, different source chains, or different router contracts will have different domain separators, even if all other fields are identical.

5 Replay Protection

We identify five distinct classes of replay attack. Each is closed by an independent mechanism.

5.1 Class 1: Cross-Network Replay

Attack: A valid message on mainnet is submitted on testnet (or vice versa).

Defense: The domain separator includes `networkID`. Mainnet and testnet have different network IDs. The domain separator mismatch causes rejection at verification step 1.

5.2 Class 2: Cross-Chain Replay

Attack: A valid message destined for chain B is submitted on chain B'.

Defense: The `dstChain` field is bound to the intended destination. Verification step 2 checks msg.dstChain = local.chainID. Chain B' rejects the message.

5.3 Class 3: Same-Chain Double Spend

Attack: A valid message is submitted twice on the same destination chain.

Defense: The destination chain maintains a consumed-nonce mapping per source chain: consumed[srcChain][nonce] $\in \{0, 1\}$. Verification step 7 checks this bitmap. The second submission is rejected.

5.4 Class 4: Nonce Skip

Attack: An attacker submits nonce $k + 2$ before nonce $k + 1$, blocking $k + 1$ permanently.

Defense: Nonces are strictly monotonic per source chain: msg.nonce = lastNonce[srcChain]+ 1. Out-of-order nonces are rejected. The relayer must submit messages in nonce order.

5.5 Class 5: Stale Message Replay

Attack: After MPC key rotation, an attacker replays a message signed by the old key set.

Defense: The `expiry` field enforces a time-to-live. Verification step 5 checks msg.expiry > block.timestamp. Additionally, the MPC attestation includes the key generation epoch; messages signed by a deprecated key set are rejected.

Theorem 5.1 (Replay Completeness)**.** *The five replay protection mechanisms are complete and independent. Every possible replay attack falls into exactly one class, and the corresponding defense prevents it.*

Proof. We enumerate all replay dimensions: network identity, chain identity, message identity (nonce), temporal validity, and key validity. Each dimension is protected by exactly one mechanism. The mechanisms are orthogonal: disabling any single mechanism enables exactly one class of replay, not multiple. □

6 MPC Attestation

Messages are not self-authenticating. An MPC committee of n nodes provides threshold attestation.

6.1 Signing Protocol

The MPC committee uses CGGMP21 [1] for ECDSA threshold signing (EVM-compatible chains) and FROST [2] for Schnorr threshold signing (non-EVM chains).

For a message M with hash $H = \text{keccak256}(\text{abi.encode}(M))$:

1. Each MPC node independently verifies the source chain event (deposit confirmation, finality).
2. Each node that confirms the event produces a signature share over H.
3. The coordinator collects t-of-n shares and produces the threshold signature σ.
4. The message (M, σ) is submitted to the destination chain.

6.2 Verification on Destination

The destination chain verifier holds the MPC committee's public key (registered on the P-Chain). Verification is a single `ecrecover` (ECDSA) or Schnorr verification—identical to verifying a single signature. The threshold structure is invisible to the verifier.

6.3 Key Rotation

MPC key rotation occurs via DKG on the M-Chain:

1. New key shares are generated via proactive secret sharing.
2. The new public key is registered on the P-Chain.
3. A transition period accepts signatures from both old and new keys.
4. After the transition period, old keys are deprecated.

7 Teleport Protocol: End-to-End Flow

The Teleport protocol implements four operations using the authenticated message protocol.

7.1 Deposit

1. User calls `TeleportVault.deposit(asset, amount, dstChain, recipient)` on source chain.
2. Vault locks `amount` of `asset` and emits a `Deposited` event.
3. MPC watchers observe the event and wait for source chain finality.

7.2 Burn (MPC Internal)

1. After finality confirmation, MPC nodes construct a `BURN` message.
2. t-of-n nodes sign the message hash.
3. The signed `BURN` message is recorded in the MPC event log.

7.3 Release

1. The relayer submits the signed `RELEASE` message to the destination chain.
2. The destination chain's `TeleportRouter` runs all 8 verification checks.
3. If verified, the router mints wrapped tokens (or releases native tokens) to `recipient`.

7.4 Backing Attestation

1. Periodically, the MPC committee attests to the total backing per asset per chain.
2. The `BACKING` message contains a monotonically increasing backing amount.
3. Destination chains compare minted supply against attested backing.
4. If minted $>$ backing $\times (1 + \epsilon)$, the bridge auto-pauses.

Invariant 7.1 (Backing Monotonicity)**.** *For any asset a and chain c, backing attestation amounts are monotonically non-decreasing: $backing(a, c, t_1) \leq backing(a, c, t_2)$ for $t_1 < t_2$.*

8 Verification Rules

Every `RELEASE` message on the destination chain must pass all 8 checks. Failure of any single check causes the entire message to be rejected.

#	Check	Rejects
1	Domain separator matches local config	Cross-protocol confusion
2	`dstChain` matches `local.chainID`	Cross-chain replay
3	`version` is supported	Protocol downgrade
4	`nonce` = `lastNonce[srcChain]` +1	Replay & nonce skip
5	`expiry` > `block.timestamp`	Stale messages
6	MPC signature valid for current key	Forgery & old key replay
7	Nonce not in consumed bitmap	Double spend
8	Backing attestation covers minted supply	Undercollateralization

8.1 Solidity Implementation

Listing 1: Verification function (simplified)

```
function verifyAndExecute(Message calldata msg, bytes calldata sig) external {
    // Check 1: Domain separator
    require(msg.domainSeparator == DOMAIN_SEPARATOR, "domain");
    // Check 2: Destination chain
    require(msg.dstChain == block.chainid, "chain");
    // Check 3: Version
    require(msg.version == PROTOCOL_VERSION, "version");
    // Check 4: Nonce ordering
    require(msg.nonce == lastNonce[msg.srcChain] + 1, "nonce");
    // Check 5: Expiry
    require(msg.expiry > block.timestamp, "expired");
    // Check 6: MPC signature
    require(verifyMPCSignature(msg, sig), "sig");
    // Check 7: Not consumed
    require(!consumed[msg.srcChain][msg.nonce], "consumed");
    // Check 8: Backing check
    require(totalMinted[msg.asset] + msg.amount
            <= backingAttestation[msg.asset], "backing");

    // Mark consumed and update nonce
    consumed[msg.srcChain][msg.nonce] = true;
    lastNonce[msg.srcChain] = msg.nonce;

    // Execute
    _release(msg.asset, msg.amount, msg.recipient);
}
```

9 Gas Costs

Operation	Gas
Domain separator check	800
Chain ID + version check	400
Nonce check + update (2 SLOADs + 1 SSTORE)	7,200
Expiry check	200
ECDSA recovery (`ecrecover`)	3,000
Consumed bitmap update (1 SSTORE)	5,000
Backing attestation check (1 SLOAD)	2,100
ERC-20 mint/transfer	25,000–50,000
Total (verification only)	**∼18,700**
Total (with ERC-20 release)	**∼68,700**

10 Comparison with Existing Bridges

	Teleport	Wormhole	LayerZero	Axelar	Multichain
Replay classes covered	5/5	2/5	3/5	3/5	1/5
Domain separator	✓	×	✓	✓	×
Monotonic backing	✓	×	×	×	×
Auto-pause on peg drift	✓	×	×	×	×
Threshold signing	MPC t-of-n	Guardian 13-of-19	ULN	Validator set	Centralized
Open validator set	✓	×	×	✓	×

11 Conclusion

Bridge security reduces to message layer security. The authenticated cross-chain message protocol eliminates the three root causes of bridge exploits: weak authentication (replaced by MPC threshold attestation), missing replay protection (replaced by five independent mechanisms), and absent backing verification (replaced by monotonic backing attestation with auto-pause). The protocol is implemented as the Lux Teleport bridge, supporting 270+ chains with zero forged, replayed, or incorrectly processed messages in production.

References

[1] Canetti, Gennaro, Goldfeder, Makriyannis, Peled, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts," CCS 2020.

[2] Komlo, Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures," SAC 2020.

[3] Lux Industries, Inc., "Teleport Universal Bridge Architecture," 2025.

[4] Lux Industries, Inc., "Warp Messaging: Authenticated Cross-Chain Communication," 2025.

[5] Lux Industries, Inc., "Triple-Proof Quantum Finality," 2026.

[6] US Treasury, "OFAC Designates Lazarus Group Ethereum Address in Connection with Ronin Bridge Exploit," 2022.