Lux Warp Messaging: Cross-Appchain Communication with BLS Aggregate Signatures

Efficient, Trustless Message Relay Between Sovereign Chains via Multi-Signature Verification and Optimistic Delivery

Zach Kelling

Lux Industries

2023

Abstract

We present Lux Warp Messaging (LWM), a cross-appchain communication protocol that enables trustless message passing between sovereign blockchains within the Lux Network ecosystem. LWM leverages BLS aggregate signatures over the BLS12-381 curve to achieve compact, efficiently verifiable proofs of cross-chain messages without requiring a trusted relay chain or committee. A message originating on a source appchain is signed by a quorum of the source appchain's validators; the aggregate signature, together with a bitmap identifying signers and their stake weights, constitutes a succinct proof verifiable on any destination appchain. We formalize the protocol's security properties, proving that message authenticity holds under a 2/3 honest-stake assumption and that liveness is preserved as long as at least 2/3 of the source appchain's weighted validators are online. We introduce an optimistic delivery mode that reduces end-to-end latency from $O(\Delta_{\text{finality}})$ to $O(\Delta_{\text{block}})$ by allowing relayers to submit messages before full finality, with a dispute window for fraud proofs. Empirical benchmarks demonstrate that LWM achieves sub-second cross-appchain message verification with aggregate signature sizes of 48 bytes regardless of the number of signers, and total message overhead below 500 bytes for appchains with up to 1,000 validators.

1 Introduction

Cross-chain communication is a fundamental requirement for multi-chain blockchain architectures. As the number of application-specific blockchains (appchains) grows, the ability to transfer assets, share state, and coordinate actions across chains becomes critical for composability and user experience [6, 5].

Existing cross-chain messaging solutions fall into three categories: (i) trusted bridge committees, which introduce centralization risks [7]; (ii) relay chains that verify source chain consensus on the destination chain, which are expensive in gas and latency [5]; and (iii) light client protocols that verify block headers, which require maintaining header chains on every connected chain [8].

Lux Warp Messaging (LWM) takes a fourth approach: leveraging the shared validator infrastructure of the Lux Network to produce BLS aggregate signatures that attest to cross-chain messages. Because appchains share validators with the primary network, the destination appchain can verify the source appchain's validator set without maintaining a separate light client. The BLS12-381 pairing-based signature scheme [1, 2] enables aggregation of arbitrarily many signatures into a single 48-byte proof, achieving constant-size verification regardless of validator set size.

1.1 Contributions

(i) We formalize the LWM protocol including message format, signing, aggregation, relay, and verification phases.

(ii) We prove security under 2/3 honest-stake with formal reduction to the co-CDH assumption on BLS12-381.

(iii) We introduce optimistic delivery with fraud proofs for latency-sensitive applications.

(iv) We provide concrete benchmarks showing sub-second verification and sub-500-byte message overhead.

(v) We analyze relay incentive design to ensure liveness without trusted infrastructure.

1.2 Paper Organization

Section 2 covers BLS signature preliminaries and the Lux appchain model. Section 3 specifies the full LWM protocol. Section 4 provides formal security analysis. Section 5 introduces the

optimistic delivery mode. Section 6 discusses relay network design and incentives. Section 7 addresses validator set tracking across appchains. Section **??** analyzes on-chain verification costs. Section 8 presents message batching for throughput scaling. Section 9 compares LWM with existing protocols. Section 10 reports empirical benchmarks. Section 14 concludes.

2 Preliminaries

2.1 Bilinear Pairings and BLS12-381

Let $\mathbb{G}_1, \mathbb{G}_2$ be additive groups and $\mathbb{G}_T$ a multiplicative group, all of prime order p. A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying:

1. **Bilinearity**: $e(aP, bQ) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, a, b \in \mathbb{Z}_p$.
2. **Non-degeneracy**: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$ for generators g_1, g_2.
3. **Efficiency**: e is computable in polynomial time.

BLS12-381 [4] is a pairing-friendly curve providing approximately 128 bits of security with:

- $\mathbb{G}_1$ elements: 48 bytes (compressed)
- $\mathbb{G}_2$ elements: 96 bytes (compressed)
- Scalar field order: $p \approx 2^{255}$

2.2 BLS Signature Scheme

Definition 1 (BLS Signatures [1]). *Let $H : \{0,1\}^* \to \mathbb{G}_1$ be a hash-to-curve function modeled as a random oracle.*

- $\mathsf{KeyGen}(1^\lambda)$*: Sample* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$*; output* $(\mathsf{sk}, \mathsf{pk} = \mathsf{sk} \cdot g_2)$.
- $\mathsf{Sign}(\mathsf{sk}, m)$*: Output* $\sigma = \mathsf{sk} \cdot H(m) \in \mathbb{G}_1$.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$*: Accept iff* $e(\sigma, g_2) = e(H(m), \mathsf{pk})$.

Definition 2 (Same-Message Aggregate Signature). *Given n signatures $\sigma_1, \ldots, \sigma_n$ on the same message m from public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$:*

$$\sigma_{agg} = \sum_{i=1}^{n} \sigma_i \in \mathbb{G}_1 \tag{1}$$

$$\mathsf{apk} = \sum_{i=1}^{n} \mathsf{pk}_i \in \mathbb{G}_2 \tag{2}$$

Verification: accept iff $e(\sigma_{agg}, g_2) = e(H(m), \mathsf{apk})$.

The aggregate signature is a single $\mathbb{G}_1$ element (48 bytes), regardless of n.

2.3 Co-CDH Assumption

Definition 3 (Computational co-Diffie–Hellman). *Given $(g_1, g_2, a \cdot g_1, a \cdot g_2, b \cdot g_1)$ for random $a, b \in \mathbb{Z}_p$, it is computationally hard to compute $ab \cdot g_1$.*

Theorem 1 ([2]). *BLS aggregate signatures are existentially unforgeable under chosen-message attack in the random oracle model, assuming co-CDH holds on BLS12-381 and all signers prove possession of their secret keys.*

2.4 Lux Appchain Model

Definition 4 (Appchain)**.** *A Lux appchain* $\mathcal{S} = (\mathcal{V}, \mathsf{VM}, \mathsf{State}, \mathsf{ChainID})$ *consists of:*

- *A weighted validator set* $\mathcal{V} = \{(v_i, w_i, \mathsf{pk}_i)\}_{i=1}^{n}$ *with total weight* $W = \sum_i w_i$.
- *A virtual machine* VM *defining state transition rules.*
- *A state* State *maintained by consensus.*
- *A unique 32-byte chain identifier* $\mathsf{ChainID}$.

Consensus requires agreement from validators with combined weight $\geq 2W/3$.

3 Protocol Specification

3.1 Message Format

Definition 5 (Warp Message)**.** *A warp message is a tuple:*

$$\mathcal{M} = (\mathsf{src}, \mathsf{dst}, \mathsf{nonce}, \mathsf{payload}, \mathsf{expiry}, \mathsf{gasLimit}) \tag{3}$$

where $\mathsf{src}, \mathsf{dst} \in \{0,1\}^{32}$ *are chain IDs,* $\mathsf{nonce} \in \mathbb{N}$ *is a per-pair sequence number,* $\mathsf{payload} \in \{0,1\}^*$ *is the application data,* $\mathsf{expiry} \in \mathbb{N}$ *is a block height deadline, and* $\mathsf{gasLimit} \in \mathbb{N}$ *bounds destination execution cost.*

The signing digest is domain-separated:

$$d(\mathcal{M}) = H_{\text{warp}}\big(\texttt{"LUX-WARP-V1"} \| \mathsf{src} \| \mathsf{dst} \| \mathsf{nonce} \| \mathsf{SHA256}(\mathsf{payload}) \| \mathsf{expiry} \| \mathsf{gasLimit}\big) \tag{4}$$

where $H_{\text{warp}} : \{0,1\}^* \to \mathbb{G}_1$ uses the hash-to-curve construction from RFC 9380.

3.2 Signing Phase

Algorithm 1 Warp Message Signing (Validator v_i)

Require: Finalized block B at height h containing log event $\mathsf{WarpEmit}(\mathcal{M})$

1: Verify B is accepted by appchain consensus (finalized)
2: Verify $\mathcal{M}.\mathsf{src}$ = local chain ID
3: Verify $\mathcal{M}.\mathsf{expiry} > h$
4: Compute $d \leftarrow d(\mathcal{M})$
5: $\sigma_i \leftarrow \mathsf{sk}_i \cdot d$ ▷ BLS sign in $\mathbb{G}_1$
6: Broadcast $(\mathcal{M}.\mathsf{nonce}, i, \sigma_i)$ to relayer gossip network

Key invariant: honest validators only sign messages that appear in finalized blocks. This is the foundation of message authenticity.

3.3 Aggregation Phase

Algorithm 2 Signature Aggregation (Relayer)

Require: Partial signatures $\{(i, \sigma_i)\}$ for message $\mathcal{M}$
Require: Source validator set $\mathcal{V}_{\text{src}}$
Ensure: Aggregate proof Π or $\perp$
$\mathcal{Q} \leftarrow \emptyset$; $W_q \leftarrow 0$
for each (i, σ_i) received **do**
  if $e(\sigma_i, g_2) = e(d(\mathcal{M}), \mathsf{pk}_i)$ **then** ▷ Verify individual sig
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{i\}$; $W_q \leftarrow W_q + w_i$
  end if
end for
if $W_q < 2W/3$ **then**
  return $\perp$ ▷ Insufficient quorum
end if
$\sigma_{\text{agg}} \leftarrow \sum_{i \in \mathcal{Q}} \sigma_i$ ▷ Point addition in $\mathbb{G}_1$
$\mathsf{bm} \leftarrow \texttt{BitVector}(n)$ with bit i set $\forall\, i \in \mathcal{Q}$
return $\Pi = (\sigma_{\text{agg}}, \mathsf{bm})$

3.4 Verification Phase

Algorithm 3 On-Chain Warp Verification (Destination Appchain)

Require: Message $\mathcal{M}$, proof $\Pi = (\sigma_{\text{agg}}, \mathsf{bm})$
Require: Source validator set $\mathcal{V}_{\text{src}}$ for current epoch
$\mathcal{Q} \leftarrow \{i : \mathsf{bm}[i] = 1\}$
$W_q \leftarrow \sum_{i \in \mathcal{Q}} w_i$
if $W_q < 2W_{\text{src}}/3$ **then**
  revert `"InsufficientQuorum"`
end if
if block.height $> \mathcal{M}$.expiry **then**
  revert `"MessageExpired"`
end if
if nonces$[\mathcal{M}.\text{src}] \neq \mathcal{M}$.nonce **then**
  revert `"InvalidNonce"`
end if
$\mathsf{apk} \leftarrow \sum_{i \in \mathcal{Q}} \mathsf{pk}_i$
$d \leftarrow d(\mathcal{M})$
if $e(\sigma_{\text{agg}}, g_2) \neq e(d, \mathsf{apk})$ **then**
  revert `"InvalidSignature"`
end if
nonces$[\mathcal{M}.\text{src}] \leftarrow \mathcal{M}$.nonce $+ 1$
Execute $\mathcal{M}$.payload with gas limit $\mathcal{M}$.gasLimit

3.5 Proof Size Analysis

Proposition 2 (Constant-Size Core Proof)**.** *The proof* Π *has size* $48 + \lceil n/8 \rceil$ *bytes, where* $n = |\mathcal{V}_{src}|$.

Proof. $\sigma_{\text{agg}} \in \mathbb{G}_1$ compresses to 48 bytes. The bitmap bm uses one bit per validator, totaling $\lceil n/8 \rceil$ bytes. These are the only components; the message $\mathcal{M}$ itself is required by the application and not part of the proof overhead. □

Table 1: Total warp message sizes (header + proof) for varying validator counts.

$\lvert\mathcal{V}\rvert$	**Signature (B)**	**Bitmap (B)**	**Total w/ Header (B)**
100	48	13	181
500	48	63	231
1,000	48	125	293
5,000	48	625	793

4 Security Analysis

4.1 Threat Model

We consider a Byzantine adversary $\mathcal{A}$ controlling validators with total stake $W_{\mathcal{A}} < W/3$ on the source appchain. The adversary may:

1. Produce arbitrary signatures with compromised keys.
2. Delay, reorder, or drop relay messages (asynchronous network).
3. Attempt rogue-key attacks during key registration.
4. Collude across appchains (cross-chain adversary).

4.2 Rogue Key Prevention

Definition 6 (Proof of Possession (PoP))**.** *At registration, validator v_i provides $\pi_i = \mathsf{sk}_i \cdot g_1 \in \mathbb{G}_1$. Verification: $e(\pi_i, g_2) = e(g_1, \mathsf{pk}_i)$.*

Lemma 3 ([3])**.** *With PoP, same-message BLS aggregate signatures resist rogue-key attacks: no adversary can choose $\mathsf{pk}_{\mathcal{A}}$ to cancel honest contributions in the aggregate.*

4.3 Message Authenticity

Theorem 4 (Authenticity)**.** *Under co-CDH on BLS12-381 and $W_{\mathcal{A}} < W/3$, no PPT adversary can produce a valid proof Π for a message $\mathcal{M}$ not finalized on the source appchain, except with probability $\mathsf{negl}(\lambda)$.*

Proof. Suppose $\Pi = (\sigma_{\text{agg}}, \mathsf{bm})$ verifies for $\mathcal{M}$ not in any finalized block. Let $\mathcal{Q}$ be the signers indicated by bm, with $W_q \geq 2W/3$. Since $W_{\mathcal{A}} < W/3$, there exists an honest validator $j \in \mathcal{Q}$ contributing weight $w_j > 0$.

By Algorithm 1, honest j never signed $\mathcal{M}$. For σ_{agg} to verify under $\mathsf{apk} = \sum_{i \in \mathcal{Q}} \mathsf{pk}_i$, define $\sigma_j^* = \sigma_{\text{agg}} - \sum_{i \in \mathcal{Q} \setminus \{j\}} \sigma_i$. Then σ_j^* is a valid BLS signature by j on $d(\mathcal{M})$, constituting an existential forgery. This contradicts co-CDH security of BLS. □

4.4 Liveness

Theorem 5 (Liveness)**.** *If honest validators with total weight $\geq 2W/3$ are online and the network delivers messages within bounded delay Δ, then every emitted warp message is relayed to the destination within $3\Delta + O(1)$ time.*

Proof. Honest validators sign within one consensus round after finalization ($\leq \Delta$). Relayer collects signatures within Δ of broadcast. Submission to destination occurs within Δ. Total: 3Δ plus constant computation time for aggregation and verification. □

4.5 Replay Protection

Per-pair nonces (Algorithm 3, line 8) prevent replay: each $(\mathsf{src}, \mathsf{dst})$ pair maintains a monotonic counter, and the destination rejects any message whose nonce does not match the expected next value.

Lemma 6 (No Replay)**.** *A message $\mathcal{M}$ with nonce k accepted on the destination at nonce counter k cannot be replayed, since the counter advances to $k+1$ upon acceptance.*

5 Optimistic Delivery

5.1 Protocol Description

For latency-sensitive applications, LWM supports optimistic delivery where messages are executed before source finality:

Algorithm 4 Optimistic Warp Delivery

Require: Message $\mathcal{M}$ in preferred (unfinalized) block B
Require: Relayer economic bond $\beta \geq 2 \cdot \mathcal{M}.\mathsf{gasLimit} \cdot \text{gasPrice}$
1: Relayer posts $(\mathcal{M}, H(B), \beta)$ to destination
2: Destination executes $\mathcal{M}.\mathsf{payload}$ provisionally
3: Open dispute window: $\Delta_d = 100$ blocks (≈ 20 min)
4: **if** valid fraud proof received within Δ_d **then**
5: Revert provisional execution
6: Slash β; distribute 50% to challenger, 50% burned
7: **else**
8: Finalize execution; refund β plus relay fee
9: **end if**

5.2 Fraud Proof Construction

Definition 7 (Fraud Proof)**.** *A fraud proof against optimistic message $\mathcal{M}$ claiming inclusion in block B at height h consists of:*

$$\mathsf{FP} = (B', h, \Pi_{B'}) \tag{5}$$

where $B' \neq B$ is a finalized block at height h on the source appchain and $\Pi_{B'}$ is a valid warp proof (BLS aggregate) of B''s finality.

Theorem 7 (Optimistic Safety)**.** *If honest stake $\geq 2W/3$ and Δ_d exceeds the network synchrony bound, then undetected fraud occurs with negligible probability.*

Proof. If B is not finalized, then a different block B' is finalized at the same height (by consensus safety, at most one block is finalized per height under 2/3 honesty). Honest validators can produce $\Pi_{B'}$ and submit the fraud proof within Δ_d under synchrony assumptions. □

5.3 Latency Comparison

Table 2: End-to-end latency: standard vs. optimistic delivery.

Phase	**Standard (ms)**	**Optimistic (ms)**
Source finality	1,000–3,000	0
Signature collection	200–500	0
Aggregation	5–20	5–20
Destination submission	100–500	100–500
On-chain verification	2–5	2–5
Total	**1,307–4,025**	**107–525**

6 Relay Network Design

6.1 Incentive Structure

Relayers earn fees for successful message delivery:

$$\text{RelayFee}(\mathcal{M}) = F_{\text{base}} + |\mathsf{payload}| \cdot F_{\text{byte}} + \mathcal{M}.\mathsf{gasLimit} \cdot P_{\text{gas}}^{\text{dst}} \tag{6}$$

where $F_{\text{base}} = 0.001$ LUX, $F_{\text{byte}} = 10^{-7}$ LUX/byte, and $P_{\text{gas}}^{\text{dst}}$ is the destination gas price. The fee is escrowed on the source chain at message emission and released to the first relayer that delivers a valid proof on the destination.

6.2 Relayer Competition

Multiple relayers may attempt to deliver the same message. The first valid submission succeeds; subsequent attempts revert with nonce mismatch. This first-come-first-served market incentivizes fast relay without coordination.

Algorithm 5 Relayer Selection via Competition

```
for each relayer r monitoring source appchain do
    Collect signatures until quorum ≥ 2W/3
    Π ← Aggregate({σ_i}_{i∈Q})
    Submit (M, Π) to destination
    if nonce matches (first delivery) then
        Collect relay fee F
    else
        Transaction reverts (already delivered)
    end if
end for
```

6.3 Relayer Staking and Reputation

To prevent spam, relayers stake a minimum bond of 100 LUX. Successful deliveries increase a relayer's reputation score $\rho_r \in [0, 1]$:

$$\rho_r^{(t+1)} = 0.99 \cdot \rho_r^{(t)} + 0.01 \cdot \mathbb{1}[\text{successful delivery at } t] \tag{7}$$

High-reputation relayers receive priority in the gossip network, reducing their signature collection latency.

7 Validator Set Tracking

7.1 Cross-Appchain Validator Awareness

Destination appchains must know the source appchain's validator set to verify proofs. LWM propagates validator set updates as special warp messages:

Definition 8 (Validator Set Update (VSU))**.**

$$\mathcal{M}_{VSU} = (\mathsf{src}, \mathsf{broadcast}, e{+}1, \Delta\mathcal{V}_{e\to e+1}, \infty, 0) \tag{8}$$

where $\Delta\mathcal{V}$ is the set difference (added/removed validators with keys and weights), signed by $\geq 2/3$ of epoch-e validators.

The genesis validator set for each appchain is embedded in the primary network's state, establishing the root of trust.

7.2 Efficient Representation via Merkle Commitments

For large validator sets, we commit to the set via a Merkle tree:

$$\mathsf{VRoot}_e = \mathsf{MerkleRoot}\big(\{(\mathsf{pk}_i, w_i)\}_{i\in\mathcal{V}_e}\big) \tag{9}$$

Warp proofs then include Merkle inclusion proofs for each signer. This trades proof size (additional $\log n \cdot 32$ bytes per signer) for destination storage ($O(1)$ per tracked appchain instead of $O(n)$).

Table 3: Destination storage: full set vs. Merkle commitment.

Method	Per-Appchain Storage	Proof Overhead
Full validator set	$128n$ bytes	0
Merkle commitment	40 bytes	$32\lceil\log_2 n\rceil \cdot \lvert\mathcal{Q}\rvert$ bytes

8 Message Batching

8.1 Merkle Batch Proofs

For high-throughput applications, multiple messages can be batched:

Algorithm 6 Batched Warp Delivery

Require: Messages $\mathcal{M}_1, \ldots, \mathcal{M}_k$ for same destination

1: $\mathsf{root} \leftarrow \mathsf{MerkleRoot}(d(\mathcal{M}_1), \ldots, d(\mathcal{M}_k))$
2: Validators sign root instead of individual digests
3: $\Pi_{\text{batch}} \leftarrow (\sigma_{\text{agg}}, \mathsf{bm})$ over root
4: Relayer submits $(\mathcal{M}_1, \ldots, \mathcal{M}_k, \Pi_{\text{batch}}, \text{Merkle proofs})$
5: Destination verifies aggregate signature on root
6: For each $\mathcal{M}_j$: verify Merkle proof, execute payload

Proposition 8 (Batch Efficiency)**.** *Batching k messages reduces total verification cost from $k \cdot C_{verify}$ to $C_{verify} + k \cdot C_{merkle}$, where $C_{merkle} \ll C_{verify}$.*

For $k = 100$ messages: individual verification costs $100 \times 50{,}000 = 5{,}000{,}000$ gas; batched costs $50{,}000 + 100 \times 2{,}000 = 250{,}000$ gas—a 20x reduction.

9 Comparison with Existing Protocols

Table 4: Cross-chain messaging protocol comparison.

Protocol	**Trust**	**Proof**	**Verify Cost**	**Latency**	**Liveness Dep.**
LWM	2/3 stake	48 B + bm	50K gas	1–4 s	2/3 online
IBC	Light client	∼1 KB	∼200K gas	6–20 s	Relayer + LC
XCMP	Relay chain	∼500 B	∼100K gas	12–60 s	Relay chain
LayerZero	Oracle+Relay	∼200 B	∼80K gas	10–30 min	Oracle
Hyperlane	ISM	∼400 B	∼120K gas	5–30 min	ISM config
Axelar	PoS committee	∼300 B	∼150K gas	30–60 s	Committee

Key differentiators. LWM achieves the lowest latency and verification cost by exploiting shared validator infrastructure. IBC achieves trustlessness but at higher cost and latency. Bridge protocols (LayerZero, Hyperlane, Axelar) introduce external trust assumptions absent in LWM.

10 Empirical Benchmarks

All benchmarks run on AMD EPYC 7763 (single core), using the `blst` library for BLS operations.

Table 5: Microbenchmarks for LWM operations.

Operation	$n = 100$	$n = 500$	$n = 1000$
Individual BLS sign	0.82 ms	0.82 ms	0.82 ms
Signature aggregation	1.18 ms	5.74 ms	11.42 ms
Public key aggregation	2.05 ms	10.21 ms	20.38 ms
Pairing verification	1.73 ms	1.73 ms	1.73 ms
Total verification	**3.78 ms**	**11.94 ms**	**22.11 ms**

Table 6: End-to-end throughput under sustained load.

Mode	**Messages/s**	**Avg Latency**	**p99 Latency**
Standard (individual)	500	2.1 s	4.3 s
Optimistic (individual)	2,000	310 ms	620 ms
Batched (100/batch)	10,000	2.5 s	5.1 s

11 Protocol Extensions

11.1 Conditional Message Execution

Warp messages can include execution conditions evaluated on the destination:

Definition 9 (Conditional Warp Message)**.** *A conditional message* $\mathcal{M}_c = (\mathcal{M}, \mathsf{condition})$ *where* condition *is a predicate over destination chain state. The message payload executes only if the condition evaluates to true at execution time.*

Use cases include price-conditional cross-chain swaps, time-locked governance votes, and state-dependent contract calls.

11.2 Message Ordering Guarantees

Definition 10 (FIFO Ordering)**.** *For messages from the same source to the same destination, LWM guarantees FIFO delivery: if* $\mathcal{M}_1$ *is emitted before* $\mathcal{M}_2$ *on the source, then* $\mathcal{M}_1$ *is processed before* $\mathcal{M}_2$ *on the destination.*

This is enforced by the nonce mechanism (Algorithm 3, line 8): the destination accepts nonce k only if nonce $k-1$ has been processed.

Proposition 9 (Causal Ordering)**.** *FIFO ordering per source-destination pair, combined with application-level dependency tracking, provides causal ordering for multi-party cross-chain protocols.*

11.3 Recursive Aggregation for Multi-Hop

For messages traversing multiple appchains (e.g., $A \to B \to C$), recursive BLS aggregation avoids linear proof growth:

Algorithm 7 Multi-Hop Warp Relay

Require: Message $\mathcal{M}$ from appchain A to appchain C via appchain B

1: Appchain A validators sign $\mathcal{M}$, produce Π_A
2: Relayer delivers $(\mathcal{M}, \Pi_A)$ to appchain B
3: Appchain B verifies Π_A, processes or forwards
4: Appchain B validators sign forwarded message, produce Π_B
5: $\Pi_{\text{recursive}} \leftarrow (\Pi_A, \Pi_B)$ ▷ Chain of proofs
6: Relayer delivers $(\mathcal{M}, \Pi_{\text{recursive}})$ to appchain C
7: Appchain C verifies Π_B (trusts B's verification of Π_A)

Each hop adds 48 bytes (aggregate signature) plus bitmap. For a 3-hop path with 100-validator appchains: total proof $\approx 3 \times (48 + 13) = 183$ bytes.

11.4 Backward Compatibility

LWM messages include a version byte in the domain separator, enabling protocol upgrades while maintaining backward compatibility:

$$d_v(\mathcal{M}) = H_{\text{warp}}\big(\texttt{"LUX-WARP-V"} \| v \| \mathsf{src} \| \cdots\big) \tag{10}$$

Destination contracts can support multiple versions simultaneously, with deprecated versions disabled via governance after a transition period.

Warp 2.0 (LP-021v2 / LP-6022). The current shipping version is **Lux Warp 2.0**, a Prism-bound hybrid envelope that carries the Warp 1.x message intact (the *Beam* lane: BLS aggregate + signer bitmap, this paper) alongside two post-quantum lanes: a per-validator *ML-DSA cert set* (FIPS 204) and a *Pulse* produced by the source chain's Pulsar threshold kernel (Lux's variant with DKG2 + Pulsar-SHA3, see LP-073). All three lanes are bound to a common source-chain transcript, so a destination chain can verify under any combination of classical /

PQ assumptions, and Warp 1.x bytes parse forward into a Warp 2.0 envelope with PQ lanes empty. Reference implementation: `github.com/luxfi/warp` (`envelope.go`, `pulsar/`). Warp 2.0 is the envelope path used by Lux's Quasar consensus integration (LP-110); Lux Teleport (LP-6332) is the asset-transfer protocol layered on top.

11.5 Rate Limiting and Flow Control

To prevent spam, each source-destination pair has a configurable message rate limit:

Definition 11 (Rate Limit)**.** *Each appchain pair (s, d) has a maximum message rate $\mu_{s,d}$ messages per epoch. Messages exceeding this rate are queued and delivered in subsequent epochs. The rate limit is set by the destination appchain's governance.*

Table 7: Default rate limits by message type.

Message Type	Per-Epoch Limit	Per-Block Limit
Token transfer	10,000	50
Contract call	5,000	25
Validator set update	1	1
Governance action	10	1

12 Formal Verification Status

The core LWM protocol has been formally verified using the Tamarin prover for the following properties:

1. **Authentication**: A valid proof Π implies the message was signed by a quorum of source validators.
2. **Non-repudiation**: A validator who signed cannot deny signing (signatures are publicly verifiable).
3. **Replay prevention**: Each message is delivered exactly once on the destination.
4. **Ordering**: FIFO ordering holds for same-pair messages.

The BLS signature security reduces to co-CDH, which has been extensively analyzed in the cryptographic literature. The pairing-based verification is implemented using the audited `blst` library with constant-time operations.

13 Related Work

Zamyatin et al. [7] systematize cross-chain bridge designs, identifying trust assumptions and attack surfaces. Kiayias et al. [8] develop non-interactive proofs of proof-of-work enabling SPV-level cross-chain verification. The IBC protocol [9] established the light-client paradigm for cross-chain communication in the Cosmos ecosystem.

BLS multi-signatures were formalized by Boneh et al. [2] with security proofs under the co-CDH assumption. Boldyreva [17] studied threshold BLS signatures. The `blst` library [4] provides production-grade BLS12-381 implementations used in Ethereum 2.0 and LWM.

Optimistic protocols appear in rollup designs [11, 12] and have been adapted for bridges by Optimism and Arbitrum. Our optimistic delivery mode applies similar ideas at the messaging layer rather than the execution layer.

14 Conclusion

Lux Warp Messaging achieves trustless, efficient cross-appchain communication through five design choices:

1. BLS aggregate signatures over BLS12-381 yield 48-byte constant-size proofs.
2. A 2/3 weighted-stake quorum inherits trust from appchain consensus with no external assumptions.
3. Domain-separated hash-to-curve and nonce-based replay protection ensure message integrity.
4. Optimistic delivery with economic bonds and fraud proofs reduces latency to sub-second for time-sensitive applications.
5. Native EVM precompiles reduce on-chain verification to 50,000 gas, enabling practical cross-chain composability.

14.1 Message Acknowledgment

For reliable delivery with confirmation, LWM supports acknowledgment messages:

Definition 12 (Acknowledged Message)**.** *An acknowledged warp message triggers a return message from the destination to the source confirming execution:*

$$\mathcal{M}_{ack} = (\mathsf{dst}, \mathsf{src}, \mathsf{ack\_nonce}, H(\mathsf{result}), \infty, 0) \tag{11}$$

The source contract can wait for $\mathcal{M}_{ack}$ to confirm successful execution before finalizing state transitions.

This enables two-phase commit protocols for atomic cross-appchain operations.

14.2 Message Size Limits

Table 8: Maximum warp message sizes by component.

Component	Maximum Size
Payload	64 KB
Header (fixed)	120 B
Proof (1000 validators)	173 B
Total message	65,573 B

For payloads exceeding 64 KB, applications must implement their own chunking protocol over LWM.

14.3 Replay Protection

Every Warp message includes a monotonically increasing nonce per source appchain and a chain identifier, preventing replay attacks across appchains or after chain reorganizations:

Definition 13 (Replay-Safe Message)**.** *A message m is replay-safe if and only if:*

1. $m.\mathit{nonce} > \mathit{lastSeen}[\mathit{sourceAppchain}]$ *at the destination.*

2. $m.\texttt{chainID}$ *matches the destination's expected source chain identifier.*

3. $m.\texttt{expiry} > \texttt{currentTimestamp}$ *(message has not expired).*

The destination chain maintains a per-source-appchain nonce watermark. Out-of-order delivery is permitted: the watermark advances to $\max(\texttt{lastSeen}, m.\texttt{nonce})$*, and a bitmap tracks received nonces within a sliding window of size* $W = 256$ *to reject duplicates without requiring strict ordering.*

Algorithm 8 Nonce Verification with Bitmap

Require: Message m, watermark w, bitmap $B[0..255]$

if $m.\texttt{nonce} \leq w - 256$ **then**
  return reject (too old)
end if
if $m.\texttt{nonce} \leq w$ **then**
  if $B[m.\texttt{nonce} \bmod 256] = 1$ **then**
    return reject (duplicate)
  end if
  $B[m.\texttt{nonce} \bmod 256] \leftarrow 1$
else
  Clear bits in B for nonces in $(w, m.\texttt{nonce} - 256]$
  $w \leftarrow m.\texttt{nonce}$
  $B[m.\texttt{nonce} \bmod 256] \leftarrow 1$
end if
return accept

14.4 Congestion Feedback

The protocol includes a congestion signaling mechanism. Each destination appchain maintains a message ingestion rate counter λ_{in} (messages per second). When λ_{in} exceeds the processing capacity $\lambda_{\max}$, the destination emits a backpressure signal via Warp to the source appchain:

$$\texttt{backpressure}(t) = \max\left(0, \frac{\lambda_{\text{in}}(t) - \lambda_{\max}}{\lambda_{\max}}\right) \tag{12}$$

Source appchains receiving a non-zero backpressure signal apply exponential backoff to their outbound message rate, reducing transmission by a factor of 2^{-k} after k consecutive congestion signals. Normal transmission resumes after receiving a zero-backpressure acknowledgment.

The protocol is live on Lux Network mainnet and supports the growing appchain ecosystem. Future work includes recursive BLS aggregation for multi-hop routing, threshold signature schemes for enhanced privacy, and integration with zero-knowledge proofs for compressed state assertions.

References

[1] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT*, 2001.

[2] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *ASIACRYPT*, 2018.

[3] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," in *EUROCRYPT*, 2007.

[4] S. Bowe, "BLS12-381: New zk-SNARK elliptic curve construction," *Zcash Blog*, 2017.

[5] J. Kwon and E. Buchman, "Cosmos: A network of distributed ledgers," 2016.

[6] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," 2016.

[7] A. Zamyatin et al., "SoK: Communication across distributed ledgers," in *FC*, 2021.

[8] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," in *FC*, 2020.

[9] C. Goes, "The interblockchain communication protocol," *arXiv:2006.15918*, 2020.

[10] P. Robinson and R. Hyland-Wood, "Atomic crosschain transactions for Ethereum private sidechains," *Blockchain: Research and Applications*, 2022.

[11] H. Kalodner et al., "Arbitrum: Scalable, private smart contracts," in *USENIX Security*, 2018.

[12] V. Buterin, "An incomplete guide to rollups," *Ethereum Blog*, 2021.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[14] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014.

[15] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *OSDI*, 1999.

[16] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *PODC*, 2019.

[17] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *PKC*, 2003.

[18] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," in *PKC*, 2006.

[19] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Master's thesis, 2016.

[20] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *JACM*, 1988.

[21] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *JACM*, 1985.

[22] S. D. Galbraith, "Mathematics of public key cryptography," Cambridge University Press, 2012.

[23] L. Gudgeon et al., "SoK: Layer-two blockchain protocols," in *FC*, 2020.

[24] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, 1991.

[25] B. Libert et al., "Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions," in *ASIACRYPT*, 2016.

[26] L. Baird, "The Swirlds hashgraph consensus algorithm," 2016.