

The Rand Bridge: Attested Lock-and-Mint into a Shielded Pool with Post-Quantum Co-Signatures

Rand Protocol
Technical report, draft 1

September 2026

Abstract

Rand is a confidential blockchain whose value lives as commitments in a shielded pool rather than as transparent balances. Its payments story needs a stable asset, and no issuer deploys one natively on a chain without volume. The Rand bridge imports USDT and USDC from Ethereum, BNB Smart Chain, Tron and Solana as a single shielded dollar, z USD, backed one-for-one by custody on those chains. This report specifies the bridge as built: the 184-byte transfer message and its two signature layers (a secp256k1 guardian quorum for every message, and a Dilithium2 co-signature quorum on every mint into Rand, both 6-of-8 today: guardian set 1 replaced the original 5-of-6 secp256k1 set on 2026-09-25, and the Dilithium2 set grew from six to eight keys at the chain-15 genesis on 2026-09-26), the deposit path that ends in a note the chain itself derives from the attested amount, the withdrawal path in which a burn is bound to its proof and released against per-chain custody counters, the governance surface (guardian rotation with a one-day grace window, explicit listing, a one-way pause that only a post-quantum quorum can lift), and the clock rules a bridged chain imposes on its own consensus. We state and prove the properties the construction gives—quorum unforgeability, the necessity of emitter binding, custody soundness, supply auditability, and the asymmetric post-quantum bound under which a Shor-capable adversary can forge releases but not mints—and we state, with equal care, what it does not give: the guardians are neither bonded nor slashable, and although six of the eight keys now run on hosts of their own, one hosting account and one operator key still reach all of them. Deployment addresses, message sizes, gas costs and the results of the live rounds are reported.

Contents

1	Introduction	2
2	System Model	3
2.1	Chains and their codes	3
2.2	Actors	3
2.3	Trust assumptions	4
3	Messages and Attestations	4
3.1	The body	4
3.2	Digest and the classical signature layer	5
3.3	The post-quantum co-signature layer	5
3.4	Verification order on Rand	6

4	The Deposit Path	6
4.1	Locking on a source chain	6
4.2	Why the source chain sees only a hash	6
4.3	From lock to attestation to <code>BridgeAttest</code>	7
4.4	The chain derives the note	7
4.5	Replay, listing and caps	7
4.6	One asset, seven backings	8
5	The Withdrawal Path	8
5.1	The burn is bound to its proof	8
5.2	Release on the destination	8
6	Governance	9
7	Time on a Bridged Chain	9
8	Security Analysis	10
9	Deployment and Measurements	11
9.1	Addresses	11
9.2	Sizes and costs	12
9.3	Live rounds	12
9.4	The operator’s front	12
10	Limitations and Future Work	12

1 Introduction

The Rand protocol paper [1] defines the shielded pool, the STARK-proved transfer relation, and the HotStuff consensus that Rand runs, and it gives, in its Section 9, the abstract design of a bridged stablecoin: an attested lock-and-mint under a guardian quorum, minting into the pool rather than into a transparent balance, with per-backing accounting and a proof that the shielded supply equals the sum of custodied balances. This report is the companion to that section. It describes the bridge that exists—the contracts, the daemons, the Rand-side rules and the operator’s web front—at the level of bytes, checks and numbers, and it re-derives the security properties against the implementation rather than the abstraction.

Three constraints from the host chain shape everything below [1, §9.1]. Execution on Rand is confidential, so a mint must land in the commitment tree, never in a balance. The source chains are transparent, so the privacy boundary is the bridge itself: what enters is public, what circulates is not. And Rand runs no light client of any source chain; verifying Ethereum or Solana consensus inside a STARK on every transfer costs more than the transfers it would secure. The bridge is therefore of the *attested* class surveyed in [2] and instantiated at scale by Wormhole [3]: a committee observes foreign chains and signs what it saw, and the destination verifies the committee’s signatures.

Contributions. Beyond the protocol paper’s design, this report contributes:

1. the concrete message and attestation format shared by four source chains and Rand (Section 3), with the digest, the signature envelope and the verification order fixed to the byte;
2. the post-quantum co-signature layer (Section 3.3): a second, index-aligned Dilithium2 committee that must co-sign every mint into Rand and every governance action on Rand,

and a proof of exactly what a quantum adversary gains and does not gain from it (Theorem 8.5);

3. the chain-derived mint (Section 4): the note a deposit creates is computed by the ledger from the attested amount and a blinding derived from the attestation digest, so that a relayer submitting the transaction can neither inflate nor redirect it, and the recipient’s address, which no source chain ever sees, is bound to the 32-byte hash the lock carried;
4. the burn-binding rule (Section 5): a withdrawal’s destination and relayer fee are inside the statement its zero-knowledge proof binds, closing a redirect attack that a copied burn would otherwise allow;
5. the clock rules a bridged chain imposes on consensus (Section 7), because guardian-set expiry and daily mint caps are decided by block time;
6. a security analysis that is honest about the residual trust (Section 8), and the measured sizes and costs of the deployment (Section 9).

Notation. We follow [1]: H is a collision-resistant hash, $\mathcal{G}^{(j)}$ is the guardian set at index j , $q(n) = \lfloor 2n/3 \rfloor + 1$ is the quorum, μ is the attested digest, $\mathcal{B}$ is the backing set of z USD and $\Lambda_{(a,c)}$ is the per-backing locked counter. Byte strings are written $x\|y$ for concatenation; all integers on the wire are big-endian.

2 System Model

2.1 Chains and their codes

Every message names chains by a 16-bit code. Table 1 gives the codes and the finality each chain is held to. The code is not the chain’s own identifier: Rand’s chain identifier, which enters post-quantum and governance messages so that signatures made for one Rand chain never verify on another, is 15 on the live chain (it was 14 until 2026-09-26), while inside an attestation Rand is always chain 1.

Chain	Code	Token standard	Finality a guardian waits for
Rand	1	shielded note	a committed block
Ethereum	2	ERC-20	the finalized tag
BNB Smart Chain	3	ERC-20 (BEP-20)	15 confirmations
Tron	4	TRC-20	19 confirmations (solidified)
Solana	5	SPL	the finalized commitment

Table 1: Chain codes and finality rules. The consistency level a lock carries (Section 3) records the rule the emitting contract was deployed with.

2.2 Actors

Users lock a stablecoin on a source chain, naming a 32-byte recipient hash, and later burn z USD on Rand, naming a destination chain, coin and address.

Guardians are $n = 8$ operators in the current set 1 ($n = 6$ in set 0, which served until 2026-09-25). Each holds a secp256k1 key, identified by its Ethereum-style address `keccak256(pk)[12:]`, and a Dilithium2 key at the same index. A guardian daemon watches every chain, waits for finality, reconstructs the message body from what it observed, and serves signatures over an HTTP API. It refuses, fatally, to sign a second body under

the same (chain, sequence) pair, and it never signs a guardian-set rotation; rotations are signed offline.

Relayers are permissionless. A relayer watches the chains, collects exactly a quorum of signatures (the lowest indices), verifies every signature itself, and submits: a **release** on a source chain, paying that chain’s gas, or a **BridgeAttest** on Rand, paying the transaction’s fee in RAND from its own shielded notes. A relayer also keeps a registry mapping recipient hashes to full Rand addresses, because a lock carries only the hash and the mint needs the address (Section 4).

Rand validators order and execute the two bridge actions like any other transaction. They verify no source-chain header or state proof; their contribution is the ledger rules of Sections 4–6 and the clock discipline of Section 7.

The **pause key** is one Dilithium2 key fixed at genesis, distinct from every guardian key, that can halt minting and nothing else.

Endpoint admins on each source chain are an ECDSA admin, who lists tokens and lifts a pause, and a pauser, who can only pause. On Tron both roles have been multi-signature accounts since 2026-09-26 (the admin needs 3 of 5 signers, the pauser 2 of 5), and the admin is being handed to a 48-hour timelock; the signers are still derived from one extended key, so the multi-signature is nominal until they are re-keyed on separate devices. The other endpoints keep single-key roles.

2.3 Trust assumptions

Assumption 2.1 (Cryptographic). *secp256k1 ECDSA is EUF-CMA secure against classical adversaries; Dilithium2 (round 3) is EUF-CMA secure against quantum adversaries; keccak256 and BLAKE3 are collision resistant; the STARK of [1] is sound.*

Assumption 2.2 (Committee). *Fewer than $q(n)$ of the n guardians’ secp256k1 keys are corrupted, and fewer than $q(n)$ of their Dilithium2 keys are corrupted; for the current set 1, $q(8) = 6$ of eight (for set 0, $q(6) = 5$ of six).*

Assumption 2.3 (Host chains). *Each source chain is safe at the finality depth of Table 1, and Rand’s consensus is safe with $f < n/3$ Byzantine stake.*

Assumption 2.2 is the one this report cannot strengthen and does not hide. The guardians are neither bonded nor slashable by Rand’s consensus. Under set 0 all six keys, the endpoint admin keys and the deployer keys were held on one machine, so its 5-of-6 was nominal. Set 1 improves on that without closing it: guardians 0–5 run on six separate hosts, each with its own Rand node so that no shared RPC can talk a guardian into signing, and guardians 6–7 on the operator’s laptop; but the six hosts sit in one cloud account reachable by one operator key, and the laptop also holds the six hosts’ Dilithium2 seeds from the previous chain. The 6-of-8 is therefore stronger than the old 5-of-6 against a compromised host, and still nominal against a compromised operator. Section 8 bounds what a violation costs; it does not prevent one.

3 Messages and Attestations

3.1 The body

Every event the bridge acts on—a lock on a source chain, a burn on Rand, a rotation—is a *message body*: a 51-byte header followed by a payload. A transfer payload is 133 bytes, so a transfer body is exactly 184 bytes. Table 2 gives the layout. There are no length prefixes; every field is fixed-width and big-endian.

Offset	Bytes	Field	Meaning
0	4	<code>timestamp</code>	source block time, seconds
4	4	<code>nonce</code>	caller-chosen on a lock; 0 for Rand and governance
8	2	<code>emitter_chain</code>	Table 1
10	32	<code>emitter_address</code>	the emitting contract, 20-byte addresses left-padded
42	8	<code>sequence</code>	per emitter, from 0
50	1	<code>consistency_level</code>	the finality rule the emitter enforces
51	1	<code>payload_id</code>	1 for a transfer, 2 for a rotation
52	32	<code>amount</code>	$u256$, at 8 decimals
84	32	<code>token_address</code>	the coin on its home chain
116	2	<code>token_chain</code>	the coin’s home chain
118	32	<code>to</code>	recipient: an address, or on Rand a recipient hash
150	2	<code>to_chain</code>	destination
152	32	<code>fee</code>	relayer fee, $u256$, $\text{fee} \leq \text{amount}$

Table 2: The transfer message body (184 bytes). The header is the first 51 bytes.

A rotation payload (id 2) is `new_index` ($u32$) $\parallel$ n ($u8$) $\parallel$ n twenty-byte addresses, and its header names the governance emitter (1, `keccak256`(“rand-bridge-governance”)).

3.2 Digest and the classical signature layer

Definition 3.1 (Attested digest). $\mu = \text{keccak256}(\text{keccak256}(\text{body}))$.

`keccak256` is the one hash every destination verifies natively—as a precompile on the EVM and TVM, as a syscall on Solana—which is why the attestation layer uses it and nothing else. A guardian signs μ raw, with no prefix, and the signature must be low- s ($s \leq n_{\text{curve}}/2$).

Definition 3.2 (Attestation envelope). *An attestation is*

$$\text{version}(1) = 1 \parallel \text{set_index}(u32) \parallel k(u8) \parallel \sigma_1 \parallel \dots \parallel \sigma_k \parallel \text{body},$$

where each σ_i is 66 bytes: guardian index (1) $\parallel$ r (32) $\parallel$ s (32) $\parallel$ v (1, in $\{0,1\}$). A quorum attestation under set 1 carries six signatures and is $402 + 184 = 586$ bytes (five signatures, 520 bytes, under set 0).

The guardian HTTP API returns (r, s, v) without an index; the relayer prepends the index for whichever set is current. A verifier holding $\mathcal{G}^{(j)}$ accepts the envelope when $k \geq q(n)$, the indices are strictly increasing and below n , and each signature recovers to the key at its index. Theorem 8.1 and Lemma 8.2 of Section 8 are the statements this buys.

3.3 The post-quantum co-signature layer

Guardian public keys are published on every chain, and `secp256k1` falls to Shor’s algorithm. The protocol paper resolves this by requiring a post-quantum signature on the mint path [1, Thm. 9.14]. As built, the resolution is a second committee, index-aligned with the first.

Definition 3.3 (PQ guardian set and co-signature). *The genesis of a bridged Rand chain fixes $\mathcal{P} = (pq_1, \dots, pq_n)$, one Dilithium2 public key (1312 bytes) per guardian index. A co-signature on a body is a Dilithium2 signature (2420 bytes) over*

$$M = \text{“rand-bridge-pq-cosign-1”} \parallel \text{chain_id}_{\text{Rand}}(u64) \parallel \mu,$$

63 bytes in all. A PQ quorum is a list of (index, signature) pairs such that $q(n) \leq \text{len} \leq n$, indices strictly increase and are below n , every signature is exactly 2420 bytes, and every signature verifies under pq_{index} .

Two design choices are load-bearing. The Rand chain identifier is inside M , so a co-signature made for a test chain never verifies on another. The guardian-set index is *not* inside M , so a co-signature survives a rotation of the classical set: the two committees rotate independently, and the PQ indices are counted independently of the ECDSA indices. Guardians co-sign only bodies whose `to_chain` is Rand; releases on source chains stay classical, for the reason given in Theorem 8.5. The scheme is Dilithium2 round 3 [4], not FIPS 204’s ML-DSA-44 [5]; the two differ in domain separation and the deployed library implements the former. A mint therefore carries about 14.5 KB of post-quantum signatures under set 1 (six co-signatures; 12.2 KB for five under set 0).

3.4 Verification order on Rand

The ledger’s admission check for a `BridgeAttest`, in the order it runs, is: (1) the PQ list’s structure; (2) envelope decode; (3) set lookup by index; (4) replay, against the spent-digest set; (5) payload decode; (6) the mint pause; (7) emitter binding; (8) `token_chain = emitter_chain`; (9) `to_chain = 1`; (10) `amount` fits `u128` and `fee ≤ amount`; (11) `amount ≠ 0` and fits `u64`; (12) the token is listed; (13) the mint caps; (14) the classical layer: set expiry, quorum, low- s , recovery; (15) the Dilithium2 verifications last. Cheap checks precede expensive ones so that a malformed submission costs the node a few comparisons rather than a quorum of lattice verifications.

4 The Deposit Path

4.1 Locking on a source chain

On the EVM and Tron endpoints the entry point is

```
function lock(address token, uint256 amount, bytes32 randRecipient,
             uint256 relayerFee, uint32 nonce) returns (uint64 sequence)
```

It requires that the contract is not paused, the token is enabled, `randRecipient` is non-zero and `relayerFee ≤ amount`. It normalises the amount to 8 decimals (Lemma 4.1), refuses an attested amount above $2^{64} - 1$, pulls the tokens while measuring the balance delta so that fee-on-transfer tokens are rejected, increments the token’s custody counter by what was actually received, and emits `MessagePublished(sequence, nonce, consistencyLevel, payload)` together with a `Locked` event. A lock pays no protocol fee. On Solana the equivalent instruction writes the body into a `PostedMessage` account at seeds [`“msg”`, `sequence`].

Lemma 4.1 (Normalisation conservation). *Let the wire carry amounts at $d_{\max} = 8$ decimals and let the source token use d decimals ($d = 6$ for USDT and USDC on Ethereum, Tron and Solana; $d = 18$ on BNB Smart Chain). If for $d > d_{\max}$ the custodian locks $\hat{v} = \lfloor v/10^{d-d_{\max}} \rfloor \cdot 10^{d-d_{\max}}$ and attests $\hat{v}/10^{d-d_{\max}}$, and for $d \leq d_{\max}$ attests $v \cdot 10^{d_{\max}-d}$, then every locked unit is covered by an attestation and no attested unit is uncovered.*

Proof. For $d > d_{\max}$, $\hat{v}$ is exactly representable at $d_{\max}$ decimals by construction, and the residue $v - \hat{v}$ is never taken from the user. For $d \leq d_{\max}$ the map is injective and exact. In both cases the attested value equals the locked value. $\square$

4.2 Why the source chain sees only a hash

A Rand shielded address is a 32-byte spending public key together with a 1184-byte ML-KEM-768 encapsulation key [6], 1216 bytes in all; the wire’s `to` field is 32 bytes. The lock therefore carries

$$\text{to} = \text{BLAKE3}(\text{“rand-shielded-recipient”} \parallel \text{pk} \parallel \text{kem_ek}),$$

and the full address travels only inside the Rand transaction that mints, where the ledger recomputes the hash and refuses a mismatch. This is what keeps the recipient’s address off every transparent chain. It has a cost: there is no checksum on the source side, so a lock naming a hash that no one can open strands the funds, and a deposit can be minted only by a relay that has been told the full address for that hash. The operator’s front (Section 9.4) registers the address with a relay before the user is shown the hash to lock against.

4.3 From lock to attestation to BridgeAttest

Each guardian daemon observes the lock at the finality of Table 1, rebuilds the 184-byte body from the event, applies its signing policy, and serves an ECDSA signature over μ and a Dilithium2 co-signature over M . A relay assembles the envelope from a quorum of signatures (six under set 1) and submits to Rand the action

BridgeAttest{attestation, recipient, r , time, asset, envelope, pq_signatures},

paying the bundle base fee (0.001 RAND) from its own notes. The transaction is about 1.43 MB. Almost all of that is the STARK proof of the fee bundle in the production profile; the attestation is 520 bytes, the recipient 1.2 KB, the note envelope 1380 bytes and the co-signatures 12.2 KB.

4.4 The chain derives the note

Definition 4.2 (Chain-derived mint). *On accepting an attestation with amount v for backing $(a, c) \in \mathcal{B}$, the ledger constructs the note*

$$note = (pk_{recipient}, from = 0, v, idx_{zUSD}, time, r), \quad r = BLAKE3("rand-deposit-r-1" \parallel \mu),$$

*computes its Poseidon2 commitment, appends the commitment to the tree, records the encrypted envelope the submitter supplied, inserts μ into the spent set, and increments $\Lambda_{(a,c)}$ by v . The **time** field must satisfy $time \leq height$ and $height - time \leq TIME_WINDOW$.*

Proposition 4.3 (The submitter controls nothing that matters). *Under Definition 4.2, no choice available to the submitter of a **BridgeAttest** changes the recipient, the amount or the asset of the note that is minted.*

Proof. The amount and backing are read from the attested body, whose digest μ is what the quorum signed. The recipient key is bound to the body’s **to** field by the hash of Section 4.2, checked by the ledger. The asset index is resolved by the ledger from the backing. The blinding r is a function of μ . The only submitter-chosen fields are **time**, constrained to a window, and the encrypted envelope, which affects whether the recipient can *find* the note, not what it is. $\square$

The deposit is gross: the body’s **fee** is recorded but paid to no one on the inbound path, so inbound relaying is altruistic and the relay’s only cost is the bundle base. On the outbound path the fee is real (Section 5).

4.5 Replay, listing and caps

A digest μ enters a persistent spent set on acceptance; a second attestation of the same body is refused as a replay, and the mempool refuses a second pending copy. The lock’s **nonce** is caller data and plays no part in replay protection. Tokens are *listed*, never created by first sighting: an attestation naming a (chain, token) pair that is not a backing of a registered asset mints nothing, so a token that merely calls itself USDC cannot acquire a bridged identity. Each backing has a mint cap per UTC day of the block’s timestamp—100,000 $zUSD$ per backing on the live chain, seven backings—and an attestation over the cap is refused as **MintCapExceeded**, a verdict that is not permanent.

4.6 One asset, seven backings

z USD is a single asset index in the multi-asset pool, at 8 decimals, with the backing set of [1, Def. 9.9]: USDT and USDC on chains 2, 3 and 5, and USDT alone on chain 4, since Circle stopped issuing USDC on Tron in 2024. Each backing keeps

(chain, token, decimals, locked, minted_today, mint_day),

and the invariant $\text{total_supply} = \sum_{\mathcal{B}} \text{locked}$ holds because only a mint and a burn move either side. This is Theorem 8.4 in Section 8.

5 The Withdrawal Path

5.1 The burn is bound to its proof

A holder exits with

`rand bridge-burn <ASSET> <AMOUNT> <TO_CHAIN> <TOKEN> <TO>`

which produces the action `BridgeBurn{asset, amount, relayer_fee, to_chain, token, to}` riding on one hidden-asset bundle: slots 0–1 burn `amount` of the asset, slots 2–3 pay the RAND fee of at least 0.01 RAND (ten bundle bases). The bundle’s proof is bound to the transaction’s binding value, which commits to the action’s fields.

Proposition 5.1 (No redirect). *An adversary who observes a valid `BridgeBurn` cannot produce a valid transaction that burns the same notes with a different `to` or `relayer_fee`.*

Proof. The proof’s public inputs include the binding value, which is a hash over the action; changing any field of the action changes the binding and invalidates the proof, and producing a new proof for the changed statement requires the spending key of the burned notes. $\square$

The ledger’s check on a burn requires: the asset is bridged; `to` $\neq 0$ and, for chains 2–4, its upper 12 bytes are zero; $0 < \text{relayer_fee} \leq \text{amount}$; the pair (`to_chain`, `token`) backs the asset; `amount` and `relayer_fee` are whole release units, that is multiples of 10^{8-d} for the destination’s decimals d ; and `amount` $\leq$ locked of that backing, else `InsufficientBacking`. A holder may redeem into *any* listed backing up to that backing’s locked balance, not only the coin that was deposited: the asset is shared, the custody is not, and naming the backing is what keeps the accounting exact.

On acceptance the ledger emits a body whose header carries `emitter_chain` = 1, the chain’s burn emitter as `emitter_address`, the block time, a zero nonce and a zero consistency level, whose payload names the chosen backing, and whose sequence is the ledger’s burn counter, which then increments. The record is stored by sequence outside the state root and served by `rand_getBridgeBurn(sequence)`, which is what the guardians poll. The nullifiers of the burned notes are published in the same block: the claim is destroyed before the custodian releases, never after.

5.2 Release on the destination

A relayer submits `release(attestation)` to the endpoint. The contract checks that the set index is current or within the rotation grace window; the quorum and every signature; that μ is not consumed; that the emitter is (1, `randEmitter`); that `to_chain` = `token_chain` = this chain; `fee` $\leq$ `amount`; the token is enabled; the recipient’s shape; and the custody, per-transfer and daily caps, the daily window being $\lfloor \text{block.timestamp} / 1 \text{ day} \rfloor$. Effects precede transfers: $\text{consumed}[\mu] \leftarrow \text{true}$ and $\text{custody} \mathrel{-=} \text{amount}$ before any token moves. The payout takes a protocol fee of 10 basis points (capped at 100 in code), pays the relayer $\min(\text{fee}, \text{amount} -$

protocolFee), and the recipient the rest. Solana records the consumed digest as a permanent ["spent", digest] account.

Theorem 5.2 (Custody soundness). *Let each endpoint keep, per token, a counter L incremented on lock by the amount received and decremented on release by the amount attested, and refuse any release with `amount` $> L$. Then the total released for a token on a chain never exceeds the total locked there, whatever the validity of any attestation.*

Proof. By induction on releases: the guard keeps $L \geq 0$ an invariant, and L equals total locked minus total released by construction. $\square$

Theorem 5.2 is independent of Assumption 2.2. It does not prevent theft by a corrupt quorum; it bounds the loss on each chain by the value actually held there rather than by whatever a forged payload claims, and the per-transfer and daily caps (100 and 1000 tokens respectively at launch, in native units) bound the rate.

6 Governance

Classical rotation. A payload-2 message, signed by the *current* set, installs set $j + 1$ with unique non-zero keys. The old set’s transfers remain acceptable for a grace window of 86 400 seconds after the rotation’s acceptance; rotations themselves get no grace. On Rand a rotation additionally needs a PQ quorum, and it does not change the PQ set. Rotations are signed offline with the governance CLI and are bearer instruments: whoever holds a signed rotation can submit it.

Listing. RegisterBridgedToken creates a bridged asset and costs the bundle base plus a registration fee of 1 RAND; ListBacking attaches a (chain, token, decimals) backing. Both require a PQ quorum over a fixed-byte message that includes a listing nonce, so a listing signed once cannot be replayed. The required order is: list on Rand first, then enable the token on the endpoint. On chain 14 zUSD was registered at block 256 and six listings took the nonce from 1 to 7; chain 15 carries the registry, its seven backings and the outstanding supply over at genesis (Section 9).

Pause and unpaue. The pause key signs

$$M_{\text{pause}} = \text{"rand-bridge-pause-1"} \parallel \text{chain_id} \parallel \text{pause_nonce}$$

with Dilithium2; the action is bundle-less and fee-less, and it only pauses. While paused every transfer attestation is refused; burns and rotations stay open. Lifting the pause needs a PQ quorum over M_{unpause} under the same nonce. The pause key must live on a different machine from the guardian keys, since a pause is the response to their compromise. Each endpoint has an independent classical pause under its admin.

Rules v2. A genesis-gated extension (audit v4) adds PQ-set and pause-key rotation under a PQ quorum with a shared rotation nonce and no grace window, rolling mint caps in 24 slots per backing plus one global cap, and refuses listing while paused. Chain 14 did not carry it; chain 15 does, with a global cap of 4,000 zUSD per 24 hours.

7 Time on a Bridged Chain

The shielded chain reads no clock for its own rules, but a bridged chain decides two things by block time: whether a rotated-out guardian set is still inside its grace window, and which

mint-cap day an attestation lands in. Block time therefore becomes consensus input, and three rules bound it:

1. a block may not carry a timestamp earlier than its parent's (no rewind);
2. a block may advance at most 60 000 ms past its parent (the step rule), so a leader cannot jump the clock to expire a set or open a new cap day, and after a stall the chain's time catches up one step per block;
3. a validator does not vote for a block whose timestamp is more than 15 000 ms ahead of its own clock (the drift rule), a vote rule rather than a validity rule, so a quorum whose clocks agree can still certify the block.

Two consequences are worth stating. Leaders holding two thirds of the stake can freeze the clock, which is the same power they have over everything else. And after a long halt the catch-up compresses wall time: at three-second blocks a mint-cap day passes in about 76 minutes of chain time, which is why caps are per backing and small rather than large and global.

8 Security Analysis

Theorem 8.1 (Quorum unforgeability). *Under Assumption 2.1, producing an envelope that a verifier holding $\mathcal{G}^{(j)}$ accepts, for a body fewer than $q(n)$ guardians signed, succeeds with negligible probability. An adversary must hold $q(n) = 5$ of the six secp256k1 keys.*

Proof. Acceptance needs $k \geq q(n)$ signatures each recovering to a distinct key of $\mathcal{G}^{(j)}$ (Lemma 8.2); a signature under an uncorrupted key is an EUF-CMA forgery. $\square$

Lemma 8.2 (Signature non-replay). *With indices strictly increasing and below n , each guardian contributes at most one signature and k lower-bounds the number of distinct signers.*

Proof. Monotonicity forbids a repeated index; the bound forbids an index outside the set. $\square$

Theorem 8.3 (Attestation is not authorization). *If the source-chain message layer is callable by any address and a verifier accepts on Theorem 8.1 alone, any party able to submit a source-chain transaction can mint an arbitrary quantity of z USD without corrupting a guardian.*

Proof. The adversary deploys a contract that publishes a transfer payload naming the genuine USDT address and a value of its choosing; honest guardians attest to publication, not to meaning [1, Thm. 9.6]. $\square$

The bridge closes this by *emitter binding*: the Rand ledger accepts a transfer only if `emitter_address` equals the endpoint registered for the attestation's own `emitter_chain` field (check 7 of Section 3.4), and each endpoint accepts a release only from (1,randEmitter). The registration is indexed by the chain field *inside the attested body*, never by a chain supplied alongside it.

Theorem 8.4 (Bridged supply auditability). *With $M_{(a,c)}$ and $B_{(a,c)}$ the cumulative mints and burns against backing (a,c) , $\Lambda_{(a,c)} = M_{(a,c)} - B_{(a,c)}$, and the shielded supply $S = \sum_{\mathcal{B}} \Lambda_{(a,c)}$ is publicly verifiable without opening any note. If every endpoint's custody equals its backing's Λ , then S equals the custodied dollars.*

Proof. In-pool transfers are value-conserving within an asset by the transfer relation of [1]; the only non-conserving events are the mint of Definition 4.2 and the burn of Section 5, each of which names one backing and moves its counter by exactly the value minted or burned. $\square$

Theorem 8.5 (The post-quantum asymmetry). *Let an adversary compute discrete logarithms in secp256k1 (a Shor-capable adversary) but not forge Dilithium2 signatures. Then:*

1. *it cannot cause a mint on Rand, nor a guardian rotation, listing, or unpause on Rand;*
2. *it can forge a release on any source endpoint, bounded by that endpoint’s custody counter, per-transfer cap and daily cap (Theorem 5.2), and it can forge endpoint admin actions, including a classical rotation on the endpoints;*
3. *a pause on Rand is not available to it (the pause key is Dilithium2), nor is lifting one.*

Proof. (1) Every mint and every governance action on Rand requires a PQ quorum (Definition 3.3, Section 6); the adversary holds no Dilithium2 key. (2) Endpoints verify only the classical layer, since Dilithium2 verification costs on the order of 10^6 – 10^7 gas; with a quorum of the secp256k1 keys the adversary signs any release body, and the endpoint’s counters and caps are the only remaining bound. Endpoint admin and pauser roles are classical keys. (3) The pause key and the unpause quorum are Dilithium2. $\square$

The asymmetry bounds the exposure by custody rather than by shielded supply, and it shrinks as the bridge is used for flow rather than storage. It also fixes the order in which a post-quantum break should be answered: pause on Rand first, since that key is safe, then drain custody through the classical pause on each endpoint before the adversary does.

Remark 8.6 (What a corrupt quorum can do today). *Under a violation of Assumption 2.2 with both committees corrupted, the bridge mints arbitrary value and authorises illegitimate rotations; nothing in the construction detects or resists that. The pause bounds a corrupt classical quorum; it is not a defence against theft of the PQ quorum. After an unpause the effective bound is up to 32 times one backing’s daily cap, since a PQ quorum may list up to 32 backings.*

Remark 8.7 (Privacy at the boundary). *The value of every deposit and withdrawal is public, as it must be for Theorem 8.4. The recipient of a deposit appears on the source chain only as a hash and on Rand only inside the mint transaction; the note it creates is indistinguishable in the pool from any other. A deposit of USDT on Tron and a redemption of USDC on Ethereum are unlinkable by asset as well as by note, which is what the single-asset design buys over per-token pools.*

9 Deployment and Measurements

9.1 Addresses

Chain	Endpoint	Deployed at
Ethereum	0xd6EBD21C3dF90c9175EBdc8d6b377a9361604892	block 26 010 199
BNB Smart Chain	0xd6EBD21C3dF90c9175EBdc8d6b377a9361604892	block 122 760 747
Tron	TAqq2i8KfYpACPUc9f5e2gAjdSgXqmPpkU	block 86 377 422
Solana	program FGA3kY3RjfdKjUszJESMYtYXAbsnkFhhoxM3Mb34vycu	—
Rand (chain 15)	burn emitter keccak256("rand-bridge-mainnet-burn-emitter")	genesis cc30e085...b6b8

Table 3: The live endpoints. The Ethereum and BNB Smart Chain contracts share an address by deterministic deployment.

9.2 Sizes and costs

Quantity	Value
Transfer body	184 bytes
Quorum attestation, set 1 (six signatures)	586 bytes
Dilithium2 public key / signature	1312 / 2420 bytes
Co-signatures per mint, set 1 (six)	≈ 14.5 KB
BridgeAttest transaction	≈ 1.43 MB (the fee bundle’s proof)
Burn: proving time on a laptop	≈ 100 s
Ethereum lock	108k–125k gas
Ethereum / BSC release	180k–276k gas
Tron release	300k–450k energy (≈ 30 –45 TRX)
Solana release	≈ 125 k compute units, plus 817 880 lamports rent
Bundle base fee on Rand	0.001 RAND
Burn fee floor on Rand	0.01 RAND
Registration fee	1 RAND
Mint cap per backing per day (chains 14, 15)	100 000 <i>z</i> USD
Global mint cap per 24 h (chain 15, rules v2)	4 000 <i>z</i> USD
Endpoint caps (launch)	100 tokens per transfer, 1000 per day
Release protocol fee	10 bps

Table 4: Measured sizes and costs. Gas figures are from the deployment transactions and the live rounds.

9.3 Live rounds

Round one moved 1 USDT on each chain through a full round trip; the release paid 0.999 USDT after the protocol fee. Round two left 36 *z*USD in the pool against 36 USDT in custody, the invariant of Theorem 8.4 holding by inspection of the per-backing counters served by **rand_getAssets**. On 2026-09-25 the guardian set was rotated from set 0 to set 1 on all five chains. Before Rand moved from chain 14 to chain 15 on 2026-09-26, 26 *z*USD were burned on chain 14 and released on Ethereum, BNB Smart Chain and Solana (sequences 4–6); the remaining 10 *z*USD were carried into chain 15’s genesis as one note, with the counters set to 9 (Tron USDT) and 1 (Solana USDT), so that $\text{total_supply} = \sum \Lambda = \text{custody}$ holds at block 0. Chain 15 starts at guardian set 1 and at burn sequence 7, and because the Dilithium2 co-signature names the Rand chain identifier, no chain-14 mint can be replayed on it.

9.4 The operator’s front

randbridge.org is a status service and a web app in front of the bridge. To deposit, a user pastes a Rand address; the service computes the recipient hash and registers (hash, address) with a relayer over the relayer’s private registration API, then shows the user the hash to lock against on the source chain. A transfer is then tracked by lock or burn transaction hash, by mint or release transaction hash, or by (chain, sequence), through the states *finalized*, *signing*, *relaying*, *complete*, with a derived *stalled* state that names its reason: an unregistered recipient, a pause, a quorum that did not form in time, or a cap. The service matches a mint by locating the 184-byte body inside the **BridgeAttest** transaction and never stores a full address.

10 Limitations and Future Work

The bridge is the weakest component of the system it serves, and this report has tried to say so precisely rather than softly. Its trust reduces to Assumption 2.2, an unbonded committee whose keys are today co-located; every other component of Rand reduces to cryptographic hardness or to a staked, slashable Byzantine bound. Three directions follow.

Decentralise the committee. Distinct operators, jurisdictions and key-management hardware for the six guardians, and the pause key on a machine none of them touch, are operational work that no theorem replaces.

Post-quantum releases. The release path stays classical until the source chains ship post-quantum precompiles. Until then the exposure of Theorem 8.5 is bounded by custody, and custody caps should be set with that in mind.

Replace attestation with proof. Rand’s own proving infrastructure is the natural successor: a STARK that verifies an Ethereum sync-committee proof or a Solana vote aggregate would reduce the bridge’s trust to that chain’s consensus plus STARK soundness, both already in the trusted base, and would retire the guardian committee entirely [1, Rem. 9.17].

References

- [1] D. Suhubdy and A. Mohammed. *Randprotocol: Confidential Arbitrary Computations*, Draft 3, August 2026, with reconciliation passes through September 2026. `randprotocol.tex` in this repository.
- [2] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, W. J. Knottenbelt. SoK: Communication Across Distributed Ledgers. *Financial Cryptography and Data Security*, 2021.
- [3] Wormhole Foundation. Wormhole: a generic message-passing protocol. <https://docs.wormhole.com>, accessed 2026.
- [4] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé. CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation, round 3, 2021.
- [5] National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard, 2024.
- [6] National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard, 2024.
- [7] J. O’Connor, J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn. BLAKE3: one function, fast everywhere, 2020.
- [8] L. Grassi, D. Khovratovich, M. Schofnegger. Poseidon2: A Faster Version of the Poseidon Hash Function. *AFRICACRYPT*, 2023.
- [9] Polygon Zero. Plonky3: a toolkit for polynomial IOPs. <https://github.com/Plonky3/Plonky3>, 2024.