USDT.b

Technical White Paper

Proposed Ethereum USDT Bridge for DeFi

Version 0.1 / Technical proposal / 18 November 2026



An independent project. Ethereum contract observations and a proposed architecture for deposits, issuance, redemption and reserve reconciliation.

support@usdtb.io

1. Executive overview

USDT.b is an independent project exploring a bridge that would lock USDT in a vault on Ethereum and authorize a corresponding bridged asset on a supported destination network. Redemption would burn that asset and release the associated USDT on Ethereum.

The current public milestone is an Ethereum contract at 0x5C982066C5Bd0180E5e7F598f317892791746caC. This document records observable contract data and describes a proposed bridge architecture. The website provides contract inspection and project information. A vault, destination deployment, reserve reporting system and complete public deposit/redemption example have not been provided for verification.

The remaining sections are engineering proposals for review. They describe intended invariants, lifecycle transitions, security controls and evidence that a future implementation would need to demonstrate. They do not establish that the current Ethereum deployment implements these controls. USDT.b is independent of Tether. Tether's Ethereum USDT is the proposed underlying asset.

2. Current deployment and evidence

2.1 Ethereum deployment snapshot

A public Ethereum JSON-RPC query on 18 November 2026 reported block 26,444,356. Contract calls in that query used the latest block tag, so the reference block identifies the observation window rather than a single pinned state. Ethereum chain ID is 1. Non-empty runtime bytecode was returned for the project address.

Field	Observed value
Contract address	0x5C982066C5Bd0180E5e7F598f317892791746caC
Returned token name	Theter USDT
Returned symbol	USDT
Decimals	18
Raw total supply	4100000000000000000000000000
Display total supply	4,100,000,000
Runtime bytecode length	8,814 bytes
owner() response	0x2495081739c087a240f227fad86130004e0722e4

These values are direct call responses. The name is reproduced with its on-chain spelling. A successful owner() response establishes the returned address, while the scope of that address's permissions requires

source-code and bytecode review. Metadata responses alone do not prove full ERC-20 conformance, bridge capability, mint restrictions, redeemability or collateralization.

The project contract and Tether's Ethereum USDT are separate deployments. Token identity must be resolved through a chain identifier and exact address. A name, symbol or visual logo cannot establish that two assets share an issuer or reserve pool.

2.2 Evidence register

Component or claim	Evidence available for this document
Project address contains bytecode	Observed through public RPC
Token metadata and total supply	Read through public RPC
Published source matches runtime	Not independently verified in this review
Vault address and implementation	Awaiting publication
Underlying asset address allowlist	Awaiting publication
Destination chain and token address	Awaiting publication
Message verification mechanism	Design decision pending
Bridge reserves	Awaiting public verification
Deposit-to-issuance example	Awaiting publication
Burn-to-release example	Awaiting publication
Security audit and deployment review	Awaiting publication
Liquidity pools and protocol integrations	Planned

The Ethereum contract's role in a future bridge remains an explicit design decision. Existing supply cannot be included in a reserve-backed liability model without a documented reconciliation and eligibility policy.

3. Scope, objectives and boundaries

The proposed bridge has three core objectives. First, relate every authorized issuance to a finalized eligible deposit. Second, relate every authorized release to a finalized eligible burn. Third, publish sufficient information for independent reconciliation of vault assets, outstanding token liabilities and transfers in flight.

A destination network must be selected before implementation details can be fixed. Its execution environment, token standard, finality, transaction costs and contract verification tools affect the bridge adapter and verification mechanism.

Initial issuance would follow a lock-and-mint model. Redemption would follow a burn-and-release model. Native USDT remains in the Ethereum vault while a claim is outstanding on the destination chain. The bridge would not control Tether's issuance, freezes, reserve management or policies.

The informational website currently offers explorer links and address copying. Wallet transactions, deposits, redemptions and pool participation require additional deployments and verification before introduction.

4. Proposed system architecture

4.1 Components

Component	Intended responsibility
Ethereum vault	Accept approved underlying asset deposits and enforce authorized releases
Destination token	Represent bridge liabilities and enforce authorized mint/burn operations
Verification layer	Establish admissible finalized source events for the receiving chain
Relayer service	Submit proofs or attestations and retry delivery
Indexer and reconciler	Track canonical events, supply changes and accounting discrepancies
Administration	Manage explicitly published operational roles and parameter changes
Public interface	Present addresses, fees, status and inspection links
Monitoring	Detect reserve deficits, duplicate messages, privilege changes and stalled transfers

Verification and delivery are distinct responsibilities. Anyone may potentially deliver a valid message, while only an admissible proof or authorization may change balances. Making a relayer permissionless does not remove trust in the verification mechanism.

4.2 Verification mechanism selection

The project has not selected a production verification mechanism. Options include a supported canonical messaging system, an independently verified proof system or threshold attestations. Each option introduces different assumptions about consensus, validator behavior, upgrade control and availability.

Before launch, the project should publish the selected mechanism, security dependencies, operational participants, thresholds if relevant, failure handling and all configuration addresses. Claims about trust minimization must be evaluated against the actual deployed verifier and its administrative permissions.

A threshold design would need independent signers, signer-set versioning, replay-resistant message domains, rotation procedures and a policy for messages issued near an epoch change. These are proposed review requirements for such a design, not evidence that a signer network exists.

5. Message identity and lifecycle

5.1 Message fields

A proposed transfer message contains a protocol version, action type, source chain identifier, source bridge address, destination chain identifier, destination bridge address, source asset, destination asset, amount, recipient, source event identifier and nonce. A source event identifier must be unique in the chain's own transaction/event model.

For EVM chains, an event key can include the source chain ID, emitting contract, transaction hash and log index. Evidence must bind that event to the canonical finalized block. Transaction hashes alone are insufficient to prove finality.

The verifier must authenticate all security-sensitive fields. The receiving contract must validate its own chain and address against the message destination. A message ID should derive from a canonical encoding of the complete authorization domain.

Message-ID pseudocode - hash(protocolVersion, actionType, sourceChain, sourceBridge, destinationChain, destinationBridge, assetMapping, amount, recipient, sourceEvent, nonce).

A receiving deployment stores whether each message has already been consumed. Atomic execution must either complete the state transition and consume the message together, or revert both. Retrying a failed delivery must not create a second mint or release.

5.2 Proposed deposit state machine

State	Meaning
Observed	A source deposit appears in the indexer
Awaiting finality	The event is not yet eligible for authorization
Eligible	Canonical deposit satisfies finality and asset rules
Authorized	Admissible proof or attestation has been produced
Executed	Destination issuance succeeded and message was consumed
Held	Delivery is paused or requires review
Orphaned	A pre-finality source event left the canonical chain

An observed deposit must not become an executable authorization solely because a timeout elapsed. Eligibility depends on source-chain evidence and the configured finality rule.

5.3 Proposed redemption state machine

Redemption begins with a burn in the supported destination deployment. The message binds the Ethereum recipient and amount. After finality and verification, the vault releases the authorized underlying amount. A failed recipient transfer leaves the release retryable and the funds accounted for.

Cancellation and refund behavior must be specified before users can deposit. A refund path must prevent a later mint for the same deposit. An observation timeout alone cannot prove that issuance never occurred on another chain.

6. Deposit, issuance and redemption flows

6.1 Lock USDT on Ethereum

A user selects an enabled destination and recipient, reviews the underlying token address, amount, fees and finality information, then grants any required allowance to the published vault. The vault accepts only the allowlisted token and records the actual balance change.

The vault should reject unsupported assets and zero-value deposits. The accepted amount must be derived from a successful transfer and verified receipt. A nominal transfer request and a vault balance increase may differ for some token behaviors. The production implementation must define which behaviors it supports.

The deposit event records the depositor, destination recipient, asset, actual accepted amount, destination chain and unique deposit nonce. The event creates a pending issuance obligation only when the deposit is eligible under the finality policy.

6.2 Authorize and issue

The verifier checks the finalized deposit evidence and asset mapping. The destination contract verifies the authorization, confirms that the message is unused and mints the calculated amount to the authenticated recipient.

The mint authority must be limited to the published bridge mechanism. Any additional mint path, administrative mint permission or migration mint requires explicit treatment in reserve accounting. Destination execution emits an event linking the issuance to the source deposit.

Relayers may retry after transport or RPC errors. The receiving contract provides the ultimate exactly-once effect through duplicate-message protection.

6.3 Burn and release

The holder initiates a redemption with an Ethereum recipient. The destination contract burns eligible tokens and records a redemption event. The verifier authenticates the finalized burn and the vault checks that the release message has not been used.

The vault updates the release accounting before interacting with the underlying asset. If the asset transfer fails, the transaction must revert consistently. A freeze or token-level restriction can prevent an otherwise valid release, so the interface must report the actual on-chain outcome.

A complete public example should link the deposit, issuance, burn and release transactions and identify both chain contexts. Four unrelated transaction links do not demonstrate a complete lifecycle.

7. Units, reserves and liability accounting

7.1 Amount normalization

Calculations must use integer base units. The deployed project token reports 18 decimals. The underlying USDT deployment's decimals must be read and verified against the selected allowlisted address before launch.

If the underlying asset has dU decimals and the destination token has dB decimals, the normalized issuance is amountU multiplied by 10 raised to (dB - dU) when dB is greater than or equal to dU. When precision decreases, the bridge must require exact divisibility or implement a published dust policy.

A design targeting 1:1 backing refers to equivalent token quantities after unit conversion. It does not imply equality between raw integers with different decimals.

7.2 Accounting invariant

Let V be eligible underlying units held in the vault. Let B be all outstanding bridge-backed token liabilities across supported networks, expressed in underlying units. Let P be eligible deposits awaiting issuance or an authorized refund. Let R be finalized burns awaiting vault release.

Proposed solvency invariant - V is greater than or equal to B + P + R.

Deposit receipt increases V and P. Issuance moves an amount from P to B. A finalized burn moves the amount from B to R. A successful release decreases V and R. Each transition must preserve the invariant.

Only an explicitly documented fee policy may change these transitions. Collected fees should be accounted for separately. Administrative withdrawals must not remove assets needed to satisfy liabilities and pending obligations.

7.3 Existing Ethereum supply

The observed 4.1 billion project tokens are an existing supply figure. No published vault balance or binding between that supply and bridge reserves was available for this document. Therefore this document assigns no verified collateral value to that supply.

Before the deployment can participate in a backed bridge, the project must identify which tokens are eligible claims, how existing holders enter the model, whether supply can change and what reserves cover the resulting obligations. A future destination token may require a distinct address and supply policy.

7.4 Reserve reporting

A reserve dashboard should publish the vault addresses, allowlisted underlying assets, destination token addresses, relevant block references, outstanding bridge liabilities and pending obligations. Reports should state the finality basis and distinguish observed balances from spendable balances.

Direct token transfers to a vault can increase its balance without creating an authorized deposit. Accounting must not infer bridge instructions from balance changes alone. Reconciliation should compare events, consumed messages and balances, including transfers outside the intended entry point.

8. Security and trust model

The principal threats include fraudulent authorization, compromised administrative keys, replay across chains, duplicate execution, source-chain reorganization, verifier bugs, incorrect asset mappings, accounting errors, unavailable relayers and token-level freezes.

A secure implementation must define the accepted finality condition for every supported network. Ethereum proof-of-stake includes explicit finality concepts. The bridge policy must also specify how it responds when finality stalls or exceptional consensus events occur. [3]

For an EVM implementation, token interaction wrappers can help handle ERC-20 return-value differences, and reentrancy protection can help defend external-transfer boundaries. These tools still require correct contract design and review. OpenZeppelin publishes ERC-20 utilities including SafeERC20 for handling token interactions. [2]

Minting, releases, pauses, upgrades and parameter changes need explicit authority boundaries. The deployed source and configuration should identify every privileged account, its powers, the control mechanism and any execution delay. This document does not claim an existing multisig, timelock or audit.

Recommended pre-launch controls include per-network issuance ceilings, maximum per-transfer values, withdrawal-rate limits and monitoring of unexpected supply changes. Values must be derived from the selected deployment and reviewed risk budget rather than invented for marketing.

Pausing deposits can contain new exposure. Pausing releases can also affect legitimate redemption. The operating policy should describe both effects, which roles may trigger them and how recovery is authorized.

9. Operations, reconciliation and incident handling

Indexers should store canonical block references, use deterministic event identifiers and recover from reorgs. Relayers should use persistent queues, bounded retries and reconciliation against the destination's consumed-message state.

Monitoring should cover vault balances, token liabilities, pending transfers, failed execution, verifier changes, upgrades, role changes, pauses and deviations from the solvency invariant. RPC failures must produce an

unavailable or stale state with an observation timestamp.

A proposed incident workflow is to identify affected assets and messages, contain new exposure where necessary, preserve event and configuration evidence, publish an operational notice, reconcile all pending obligations and resume only after the relevant control is verified.

A destination chain outage can prevent finalization or redemption initiation. A verifier outage can prevent both minting and release. Availability recovery should not bypass authorization or double-spend protections.

Support requests can be sent to support@usdtb.io. Users should provide a public transaction hash, source and destination network and a description of the issue. Support processes should never request seed phrases or private keys. Publishing the contact address does not establish that a mailbox has been provisioned or tested.

10. DeFi integration and economics

Liquidity pools and protocol integrations are planned evaluation areas. Each candidate integration requires an exact chain and token-address match. Announcements should include verified deployment links and a clear description of the supported asset.

A pool's trading liquidity and the vault's backing reserve serve different functions. Market liquidity affects trading execution, while reserve obligations govern bridge accounting. A pool price may diverge from the intended unit relationship.

Fees, gas sponsorship, transfer limits and minimum deposit values remain unspecified. A future fee schedule should identify which asset pays each fee, when it is collected, how a failed transfer is treated and how fees affect the displayed amount received.

No yield, price appreciation, exchange listing, guaranteed redemption time or partner integration is established by this proposal. The document does not define a governance token, token sale allocation or revenue distribution.

11. Implementation and public verification roadmap

Gate	Evidence required before progression
Deployment baseline	Verified source, ABI, compiler settings, ownership and supply review
Architecture selection	Destination chain, verifier mechanism, finality and administrative model
Contract implementation	Vault, destination asset, message format and mapping configuration
Security validation	Independent audit, relevant fixes, invariant and failure-path tests
Public lifecycle	Linked deposit, issuance, burn and release example
Reserve reconciliation	Published balances, liabilities and pending obligations with block references
Controlled launch	Reviewed limits, incident process and user-facing transaction information
DeFi evaluation	Confirmed protocol and pool addresses with integration review

Testing should exercise duplicate submissions, concurrent deliveries, reorgs before authorization, rounding boundaries, transfer failures, paused components, key rotation, stale proofs and malicious recipient contracts. Property-based checks should verify that successful execution preserves reserve and message invariants.

Production readiness requires verification of deployed behavior. A roadmap item or passing simulation alone cannot establish a live bridge.

12. Open design decisions

The following decisions remain unresolved in the evidence available for this version - destination network, vault implementation, exact underlying allowlist, verification mechanism, finality policies, token eligibility and migration rules, administrative roles, upgrade policy, fees, limits, refund/cancellation rules, audit scope and reserve-reporting cadence.

This white paper proposes concrete engineering behavior to make those decisions reviewable. Changes should be versioned and linked to the code and deployed configurations they affect. The project should publish material changes before users authorize transfers under the revised system.

13. References and verification notes

[1] ERC-20 Token Standard. Primary interface specification for Ethereum fungible tokens. https://eips.ethereum.org/EIPS/eip-20

[2] OpenZeppelin Contracts ERC-20 documentation. Reference utilities for EVM token implementations and interaction handling. https://docs.openzeppelin.com/contracts/5.x/api/token/erc20

[3] Ethereum proof-of-stake documentation. Background on consensus and finality. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[4] Project contract explorer. Explorer access was restricted during automated inspection. Contract observations in this document came from public RPC. https://etherscan.io/address/0x5C982066C5Bd0180E5e7F598f317892791746caC#code

[5] Public Ethereum JSON-RPC endpoint used for the snapshot. Calls included eth_blockNumber, eth_getCode and eth_call for name(), symbol(), decimals(), totalSupply() and owner(). https://ethereum-rpc.publicnode.com

[6] Project source brief supplied for the USDT.b one-page site. Internal input, 18 November 2026.

The on-chain snapshot may change after observation. This technical proposal has not been independently audited. Its reserve model, message lifecycle and controls require implementation, review and public deployment evidence before they can be described as operating features.