ORUSD Token — Technical White Paper & Audit Reference

Public Technical Documentation · Osnias Clearing · www.osnias-clearing.com

Drafted by / Rédacteur : Denis Bouzon · ORCID 0009-0007-8894-8902

Publication intent: this technical white paper is intended for deposit in a public scholarly/technical repository with assignment of a Digital Object Identifier (DOI). The DOI reference will be added after formal deposit.

Version 2.0 · Publication date: 31 August 2026 · ORUSD Token · © 2026 Osnias Clearing. All rights reserved.

Document status: public pre-audit technical architecture and deployed-reference document for developer, integrator and smart-contract review. The exact Solidity source, ABI and flattened source are published separately as companion technical files; the Sei Testnet deployment identified below is publicly inspectable and source-verified.

DOCUMENT SCOPE — This white paper is a technical document. It describes the architecture, deployment model, smart-contract behavior, security boundaries, operational roles and audit invariants of the ORUSD token. Legal, regulatory, contractual, licensing and compliance matters are outside the scope of this technical white paper and are addressed separately in dedicated Osnias Clearing regulatory and legal documentation. The absence of a regulatory or contractual analysis in this document therefore does not indicate that those matters are ignored; they are intentionally documented in a separate corpus.

AUDIT SCOPE PRINCIPLE

> ORUSD is a restricted USD-denominated clearing-register token. Its security model is defined by controlled issuance and destruction, EOA-only circulation, explicit oracle-request boundaries, non-replayable external proofs and manager-gated settlement execution.

1. Technical Role of ORUSD

ORUSD is designed as a clearing representation denominated in US dollars. The supplied Solidity contract implements an ERC-20 accounting instrument with six decimals while deliberately removing standard delegated-transfer pathways and rejecting ordinary transfers to deployed smart contracts.

The intended normal circulation path is direct peer-to-peer transfer between user-controlled externally owned accounts (EOAs). Mint and burn operations are reserved to the ORUSD manager, either manually or after an authorized oracle has created a verifiable on-chain request.

CORE REGISTER INVARIANT

> ORUSD must remain a clearing-register instrument. The protocol must not expose a native execution path that turns ORUSD into a freely routable smart-contract market asset or that allows an oracle to mint or burn directly.

2. Deployment Model: Sei Testnet Reference vs Production

Sei Testnet (atlantic-2) — public deployed reference and staging environment

ORUSD is publicly deployed on Sei Testnet (atlantic-2) as the current on-chain reference implementation. Sei Testnet is the Sei staging environment for testing and development, and the deployment can be independently inspected through SeiScan with standard EVM tooling.

Sei Mainnet (pacific-1) — intended canonical production environment

The current source is a Sei EVM test deployment. The intended production model is a separate Sei Mainnet deployment with independently published production addresses, privileged roles, oracle configuration and operational-security controls. The test deployment is therefore a reproducible reference, not a production issuance instance.

CANONICAL STATE PRINCIPLE

> The Sei Testnet deployment is a public staging reference. A future Sei Mainnet production deployment must be independently identified and must not inherit balances, proofs or administrative rights merely from the test instance.

2A. Network Locality Invariant

The supplied ORUSD contract contains no bridge, OFT, wrapping, migration or cross-chain token-transfer implementation. Its sourceChain and externalRef oracle fields identify external evidence; they do not move ORUSD between chains.

NETWORK LOCALITY INVARIANT

External evidence may inform a clearing request, but ORUSD issuance, destruction and circulation remain local to the deployed ORUSD contract instance.

Public Sei Testnet Deployment Registry

Field	Sei Testnet deployed reference	Audit / registry status
Network	Sei Testnet — atlantic-2	Public staging / test reference
EVM Chain ID	1328 (0x530)	Official Sei Testnet EVM chain identifier
EVM RPC	https://evm-rpc-testnet.sei-apis.com	Sei Foundation public test endpoint
Explorer	https://testnet.seiscan.io	Public Sei Testnet explorer
Contract name	OsniasClearing_Orusd	Verified Solidity contract
Contract address	0x0c3a19455b898913A1DaD93d724c24062293dB5F	Source Code Verified — Exact Match
Creation transaction hash	0xce15b298a4f1b0474e9c704996fd6d08c8d939d09def0eba8205998112c1ff15	Contract-creation transaction
Deployer / initial manager	0xEAcc9B9e1f18AaE5879e9259684578903d09C4CC	Initial Ownable manager encoded in source
Initial feeRecipient	0xAD7761445C9CdCC80DFB549E02279353156b5059	Receives protocol fee; deployment configuration
GasPool	0x29a1C3F326850502d2629eb117b2451f73A9F345	Restricted recipient; test configuration
Compiler	v0.8.22+commit.4fc1097e	Verified compiler version
Optimizer	Enabled — 200 runs	Matches verified deployment settings
EVM target	Paris	Matches verified deployment settings
Explorer license	None	No open-source license selected on SeiScan
Source reference	orusd-sei-testnet.sol	Exact Solidity test source
Flattened reference	orusd-sei-testnet-flat-5.sol	Single-file verification / audit reference
Production status	Not Sei Mainnet production	Production registry and final security configuration to be published separately

Sei Testnet network definition: Cosmos chain ID atlantic-2; EVM chain ID 1328 (0x530); official public EVM RPC https://evm-rpc-testnet.sei-apis.com; WebSocket wss://evm-ws-testnet.sei-apis.com; explorer https://testnet.seiscan.io. The native gas asset is SEI.

SOURCE PUBLICATION PRINCIPLE

The exact Solidity source, ABI and flattened contract are companion technical files. The deployed Sei Testnet bytecode has been verified on SeiScan as an Exact Match. This white paper describes architecture, security boundaries, deployment identifiers and audit invariants rather than duplicating implementation code.

3. Token Identity and Numeric Model

Property	Current contract value	Audit interpretation
Solidity	^0.8.22	Compiled and verified with v0.8.22+commit.4fc1097e.
Token name	Osnias Clearing	ERC-20 display name encoded by the constructor.
Symbol	ORUSD	Public ticker encoded by the contract.
Technical decimals	6	On-chain precision; interfaces may display fewer decimals.
Protocol fee	15 bps = 0.15%	Deducted from the transferred amount.
Minimum gross P2P transfer	0.501000 ORUSD	501,000 base units.
Oracle request validity	24 hours	Execution window from request creation time.
Administration	Ownable2Step	Two-step ownership-transfer mechanism.

The public source declares PROTOCOL = "Osnias Clearing", PROTOCOL_VERSION = "1.0" and TOKEN_FULL_NAME = "Osnias USD Clearing". The ERC-20 constructor encodes name = "Osnias Clearing" and symbol = "ORUSD". These metadata

constants identify the protocol and token implementation; they do not by themselves establish a bank deposit, legal-tender status, a redemption promise or an external USD reserve.

4. P2P Transfer Model

Ordinary P2P transfers follow a gross-debit / net-receipt model. The sender is debited by the exact value submitted. The protocol fee is calculated as value × 15 / 10,000 and credited to the designated feeRecipient. The recipient receives value minus the fee.

Example	Amount
Gross transfer request	1,000.000000 ORUSD
Protocol fee (0.15%)	1.500000 ORUSD
Recipient receives	998.500000 ORUSD
Sender total debit	1,000.000000 ORUSD

The feeRecipient is exempt from the fee when transferring collected ORUSD. This prevents recursive fee generation when protocol fees are moved.

TRANSFER INVARIANT

For an ordinary user transfer, sender debit = gross value; recipient credit + feeRecipient credit = gross value. No additional ORUSD is created by the fee mechanism.

5. EOA-Only Circulation and Smart-Contract Boundary

ORUSD deliberately disables standard ERC-20 delegated-transfer behavior. approve() reverts. transferFrom() reverts. The OpenZeppelin v5 _update() hook rejects ordinary transfers to addresses with deployed contract bytecode.

- EOA → EOA transfer: permitted, subject to minimum amount, fee rules and recipient blacklist.
- EOA → smart contract: rejected.
- approve(): rejected.
- transferFrom(): rejected.
- Mint to smart contract: rejected by manager mint and oracle-request account checks.
- Burn: permitted only through manager-controlled paths.

CLEARING-ONLY CIRCULATION INVARIANT

The intended design is stronger than the absence of a native swap function: ORUSD does not expose the approval, transferFrom or ordinary contract-recipient pathways normally required by standard smart-contract routing.

The ORUSD token contract contains no customer-deposit, custody, escrow, lending, collateral-valuation, collateral-ratio, withdrawal or bank-account logic. Osnias Clearing therefore maintains a strict separation between the clearing function and the deposit / custody function. ORUSD is the clearing register; customer deposits and collateral remain outside the ORUSD smart contract and must be handled by independent partner escrow or custody arrangements. For USD-denominated escrow architecture, USDC issued by Circle is the recommended collateral asset. This is an architectural and operational recommendation for external escrow arrangements only: USDC is not embedded in ORUSD, does not create a native redemption promise inside the token, and does not give the escrow or custodian any ORUSD mint, burn, oracle or manager authority.

The ORUSD token contract contains no customer-deposit, custody, lending or collateral-management function. Where an external USD-denominated escrow layer is used, USDC issued by Circle is the recommended collateral asset. The clearing register and the deposit / custody layer remain strictly separated, and custody of USDC does not confer any privileged authority over ORUSD.

NON-CUSTODY CONTRACT PRINCIPLE

The ORUSD token contract is a clearing register. The supplied source does not provide a customer custody or lending layer and does not itself encode an external collateral asset.

Any USD-denominated collateral or partner-custody policy should be published as a separate architectural requirement and audited against the clearing/custody segregation boundary; it is not inferred from the token bytecode alone.

6. Restricted Recipient Domain

The contract maintains the restricted-recipient mapping orusdBlacklisted. It is applied directly to ORUSD transfers in the deployed contract logic.

- Initial manager: restricted from receiving ordinary ORUSD through the transfer path.
- GasPool: restricted from receiving ORUSD.
- Authorized oracle: automatically restricted from receiving ORUSD.
- Former oracle: remains restricted after oracle authorization is removed.
- Protocol fee recipient: intentionally eligible to receive ORUSD so that transfer fees can be collected.

ROLE-SEPARATION PRINCIPLE

Infrastructure addresses that validate, manage or sponsor the system should not accumulate user clearing balances merely because they hold privileged operational roles.

7. Oracle Request Architecture

An authorized oracle does not mint or burn ORUSD. Its authority is limited to creating a request after observing or validating an external event. The request is written on-chain and becomes inspectable before any supply change occurs.

Request field	Purpose
Request category	MINT or BURN
Lifecycle status	NONE / PENDING / EXECUTED / REJECTED
Submitting oracle	Authorized source of the request
Affected account	EOA whose ORUSD balance may change
Amount	Requested ORUSD quantity
sourceChain	Compact identifier of the observed external source or system
externalRef	Reference to the external evidence
createdAt	Timestamp used to enforce the 24-hour validity window

ORACLE AUTHORITY INVARIANT

An oracle may attest and request. It may not settle. Only the ORUSD manager can execute or reject an oracle request.

8. External-Proof Uniqueness and Anti-Replay

Before a request is stored, the contract derives a proof key from sourceChain and externalRef. If the same proof key has already been used, a second request reverts. The proof remains consumed even if the request is later rejected.

Each request also receives a unique requestId derived from the submitting oracle, request type, account, amount, external evidence fields and a sequential nonce.

ANTI-REPLAY INVARIANT

A given (sourceChain, externalRef) pair can create at most one ORUSD oracle request. Rejection does not free the proof for reuse.

9. Request Lifecycle and Settlement

A valid oracle request begins in PENDING state. The manager may execute it within the 24-hour validity period or reject it. Expired requests cannot be executed through the normal execution path.

- MINT request: valid execution increases the designated account balance and total supply by the approved amount.
- BURN request: valid execution reduces the designated account balance and total supply by the approved amount, subject to sufficient balance.
- The request status is set to EXECUTED before the internal mint/burn operation completes, preventing normal replay of the stored request.
- A rejected request is permanently marked REJECTED.
- An expired request remains part of the audit history but is not executable.

SETTLEMENT FINALITY PRINCIPLE

Every supply-changing oracle operation must correspond to one identifiable request and one terminal status transition.

10. Manual Manager Mint and Burn

The supplied ORUSD reference implementation contains manager-controlled manual mint and burn functions. These constitute the strongest privileged supply authority in the contract.

Manual mint is limited to a non-zero EOA recipient and a positive amount. Manual burn is manager-controlled and can apply to a specified account, subject to non-zero address, positive amount and sufficient balance enforced by ERC-20 burn accounting.

AUDIT ATTENTION — PRIVILEGED SUPPLY

The manager can mint to an eligible EOA and burn from an ORUSD account through explicit privileged functions. Production operational-security controls must therefore treat manager authority as a critical supply-control role.

11. Administrative Authority Matrix

Action	Oracle	Manager / owner	User
Submit oracle request	Yes, if authorized	No special need	No
Execute oracle request	No	Yes	No
Reject oracle request	No	Yes	No
Manual mint	No	Yes	No
Manual burn	No	Yes	No
Authorize / remove oracle	No	Yes	No
Change feeRecipient	No	Yes	No
P2P transfer	Restricted as recipient role	Normal restrictions apply	Yes
approve / transferFrom	No	No	No

SEPARATION OF DUTIES

Oracle observation and manager settlement are separate trust domains. A compromised oracle alone must not be sufficient to create or destroy ORUSD.

12. ORUSD / OSNIAS Functional Separation

ORUSD is a clearing-register token, whereas OSNIAS serves a separate governance and strategic-reserve function. The ORUSD source does not provide a native conversion, liquidity, farming or collateral dependency path linking the two token classes.

- No protocol-native ORUSD ↔ OSNIAS conversion function in the supplied ORUSD source.
- No ORUSD / OSNIAS liquidity-pool logic in the supplied ORUSD contract.
- No OSNIAS-dependent ORUSD mint or burn condition.
- No OSNIAS administrative role is implemented in the ORUSD contract.

CROSS-DOMAIN SECURITY INVARIANT

Governance-token administration and ORUSD clearing administration should remain distinct privilege domains.

13. Supply and Accounting Invariants

Invariant	Required property	Manager override
No hidden mint path	Supply increases only through explicit manager mint or execution of a valid MINT request.	Not outside explicit functions
No hidden burn path	Supply decreases only through explicit manager burn or execution of a valid BURN request.	Not outside explicit functions
Oracle non-settlement	Oracle cannot directly mint or burn ORUSD.	Not permitted
Proof uniqueness	One external proof pair can be consumed once.	Not permitted
P2P conservation	Ordinary transfer redistributes existing ORUSD between recipient and feeRecipient.	Not permitted
No delegated transfer	approve / transferFrom remain disabled.	Not permitted by current code
EOA recipient rule	Ordinary transfers to deployed	Not permitted by current code

	contracts revert.	
No native bridge path	No bridge, OFT, wrapping or migration mechanism exists in the supplied source.	Not implemented

14. Security Validation Pipeline

ORUSD is intentionally compact, but its custom ERC-20 restrictions create protocol-specific invariants that must be tested beyond ordinary token compliance. Static analysis supports review; it does not replace unit tests, fuzzing, invariant testing or manual examination.

Security objective	Primary method	Audit interpretation
Public interface exposure	Slither + manual review	Map every externally callable state-changing path against the ABI.
Manager privilege boundaries	Slither + unit tests	Verify every supply and administration function is privilege-gated.
Oracle authorization	Unit + fuzz tests	Unauthorized request creation must always revert.
Proof replay	Invariant tests	The same sourceChain/externalRef pair cannot produce a second request.
Request expiry	Time-based tests	Execution after 24h must revert.
Mint/burn accounting	Invariant tests	Supply change must equal the authorized amount.
Fee conservation	Property tests	sender debit = recipient net + fee.
EOA-only transfer	Integration tests	Ordinary transfers to contracts must revert.
Delegated-transfer lockout	Unit tests	No allowance-based transfer route may be available.
Unexpected market route	Adversarial source review	Any executable path enabling unintended contract routing should be treated as an architecture finding.

ADVERSARIAL REVIEW PRINCIPLE

Any hidden, indirect, oracle-bypass or externally triggered path capable of unauthorized mint, burn or ordinary smart-contract routing must be treated as a critical security finding.

15. Recommended Pre-Audit Sequence

- 1. Reproducible Solidity compilation with pinned OpenZeppelin dependencies.
- 2. Slither static analysis focused on privilege, supply, transfer and oracle surfaces.
- 3. Unit tests for mint, burn, fee, blacklist and ownership controls.
- 4. Fuzz and invariant testing for supply conservation and proof replay.
- 5. Sei Testnet integration testing against the verified reference contract 0x0c3a19455b898913A1DaD93d724c24062293dB5F.
- 6. Oracle lifecycle tests covering authorization, replay, rejection and 24-hour expiry.
- 7. Wallet tests for direct EOA-to-EOA transfers.
- 8. Production deployment rehearsal with production-like addresses and operational controls.
- 9. Manual security review against the invariants in this white paper.
- 10. Final production deployment review and public registry publication.

16. Mainnet Privileged-Key Security

The current public test implementation uses a simplified privileged-key model suitable for development and testing. A production deployment should replace single-key critical administration with an operational-security architecture appropriate to manager supply authority.

- Production manager authority: multisignature control rather than a single EOA.
- Cold / emergency signer separation: recovery keys stored outside ordinary operational devices.
- Oracle separation: oracle signing infrastructure independent from manager settlement keys.

- Fee-wallet separation: feeRecipient should be distinct from oracle and manager keys unless explicitly justified.
- External registry: authoritative production contract addresses, privileged roles and recovery procedures maintained outside the token contract.
- Incident procedure: documented oracle removal, ownership transfer and emergency operational response.

MAINNET SECURITY PRINCIPLE

A single compromised key should not be sufficient to fabricate an external attestation and execute the corresponding ORUSD supply change.

17. Current Source and Production Delta

Category	Meaning in this document	Examples
IMPLEMENTED — reference source	Present in the supplied Solidity source and ABI.	6 decimals; fee logic; EOA-only recipient restriction; oracle workflow; manager-controlled supply operations; anti-replay; delegated transfers disabled.
CONFIGURATION — test deployment	Deployment-specific addresses and legacy comments in the current test source.	Sei Testnet manager, feeRecipient and GasPool addresses; verified contract registry; GasPool documentation normalized to SEI.
REQUIRED — production	Production-security or deployment requirements not claimed as enforced by current bytecode.	Sei Mainnet production registry; multisig manager; cold-key separation; final production address set.
RECOMMENDED HARDENING	Items to evaluate before production.	Ownership migration handling; renounceOwnership policy; feeRecipient / restricted-recipient validation; maintain Sei-native deployment wording.

The current ORUSD source is explicitly configured for Sei Testnet. Its deployment-specific manager, feeRecipient and GasPool addresses are test configuration. The documentation-normalized source revision now identifies SEI, rather than the legacy POL wording, as the native asset held by the GasPool for gas sponsorship. This wording correction is non-executable and does not alter the deployed Sei Testnet contract or its verified on-chain behavior.

- Retain ORUSD-specific identifiers (for example ORUSD_DECIMALS and orusdBlacklisted) consistently across source, ABI and documentation.
- Keep all deployment comments aligned with Sei Testnet / Sei Mainnet terminology. The GasPool documentation now identifies SEI as the native gas asset.
- Maintain the GasPool documentation as a SEI-native gas sponsorship wallet for the relevant Sei deployment.
- Replace deployment-specific manager, feeRecipient and GasPool addresses for production and publish them in the external registry.
- Verify all deployment-specific addresses satisfy the EOA restrictions required by the contract.
- Define ownership-transfer and ownership-renunciation procedures before production.
- Validate feeRecipient configuration against restricted-recipient rules.

PRE-MAINNET REVIEW QUESTION

Can the reviewed ORUSD reference implementation be independently deployed on Sei Mainnet while preserving the documented P2P, oracle, anti-replay and supply-control invariants and without introducing undocumented circulation or custody paths?

18. Auditor Acceptance Checklist

- No oracle can directly create or destroy supply.
- Only the manager can execute or reject oracle requests.
- Every executed oracle request was previously PENDING and unexpired.
- Every external proof pair is single-use.
- Manual mint and burn paths are fully enumerated and privilege-gated.
- Ordinary P2P transfer accounting conserves token units.
- approve() and transferFrom() cannot be used.

- Ordinary transfers to smart contracts revert.
- Restricted infrastructure addresses cannot receive ordinary ORUSD.
- Fee-recipient behavior cannot create recursive fees or a transfer deadlock.
- Ownership transfer, renunciation and role migration have explicit production procedures.
- The ORUSD token contract contains no customer-deposit, custody, lending or collateral-management function.
- The public Sei Testnet reference address and any future Sei Mainnet production address are clearly distinguished in documentation and registry records.
- Network wording and GasPool documentation are aligned with Sei terminology; any future source revision must preserve this consistency before production.

FINAL SECURITY OBJECTIVE

ORUSD should do one thing and do it well: provide direct peer-to-peer USD-denominated clearing transfers between user-controlled accounts, with controlled supply, non-replayable oracle evidence and explicit manager settlement authority.

PREPARED BY — Denis Bouzon · ORCID 0009-0007-8894-8902 · Osnias Clearing · www.osnias-clearing.com

Document revision: Version 2.0 — dated 31 August 2026. ORUSD Token technical white paper and associated documentation.