CMTAT (ERC-3643/ERC-7802) Integration with LayerZero

Specification

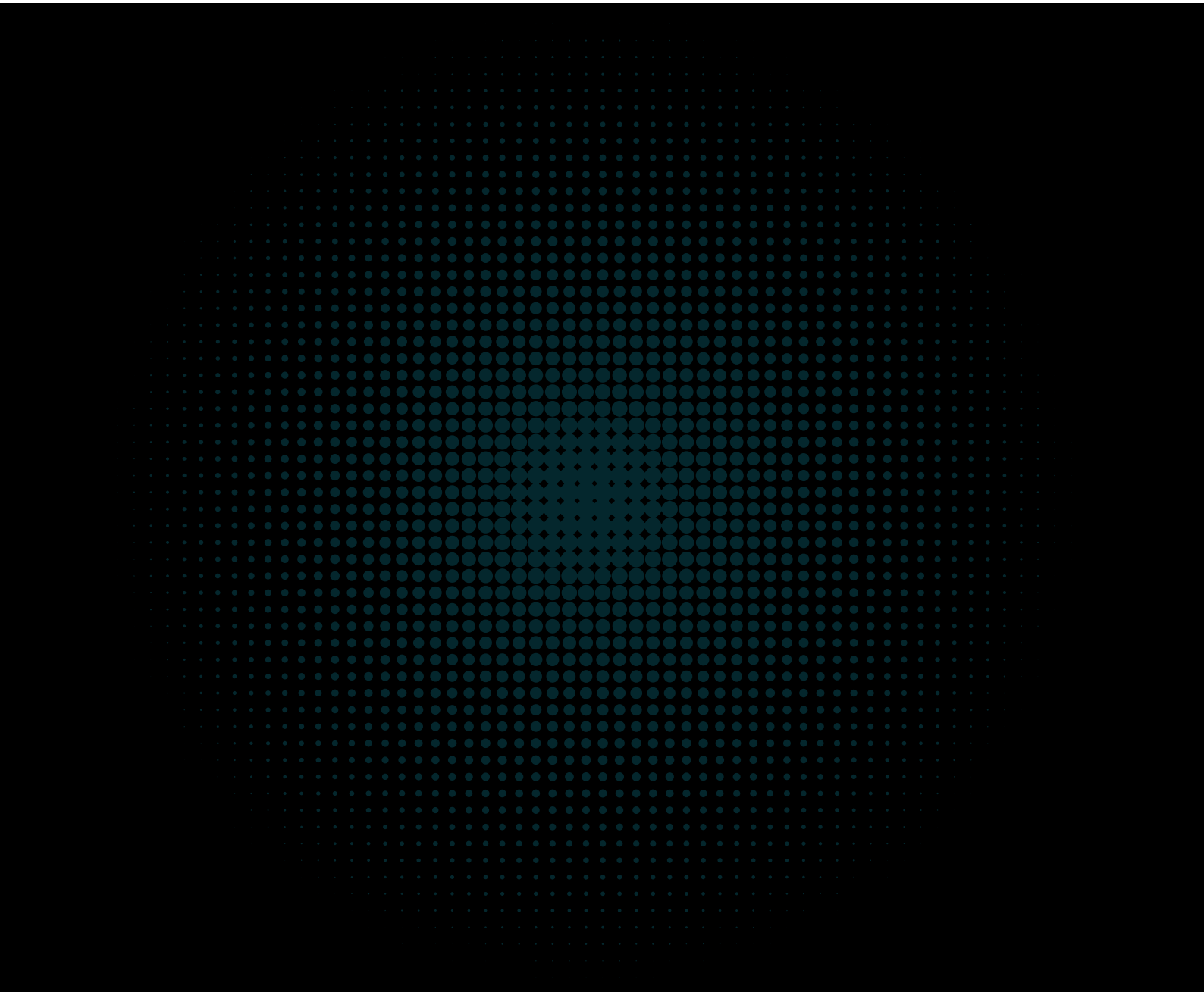
Functional specifications of the
CMTA Token to enable cross-chain bridge through LayerZero

Project version: v0.2.0

First published: March 2026

Project by Nox Labs in collaboration with CMTA

cmta.



CMTAT LayerZero Integration (ERC-3643 / ERC-7802)

This project integrates CMTAT (Capital Markets and Technology Association Token) with LayerZero, a protocol for cross-chain communication, enabling seamless token transfers across multiple EVM-compatible blockchains. Leveraging LayerZero's OFT (Omnichain Fungible Token) standard, CMTAT tokens can move easily across chains, increasing liquidity, accessibility, and interoperability for token holders.

This project was initially developed by Nox Labs in collaboration with CMTA and Taurus.

> **Note**: This project has not undergone an audit and is provided as-is without any warranties.

Table of Contents

Overview

This project provides a LayerZero adapter for CMTAT tokens, enabling cross-chain transfers between supported networks. The adapter implements the OFT (Omnichain Fungible Token) standard, allowing tokens to be burned on the source chain and minted on the destination chain.

Key Features

- **Cross-Chain Token Transfers**: Seamlessly bridge CMTAT tokens between different blockchain networks
- **LayerZero Integration**: Built on LayerZero V2 protocol for secure and efficient cross-chain messaging
- **ERC-3643 / ERC-7802**: Compatible with any tokens implementing ERC-3643 (`LayerZeroAdapter`) or ERC-7802 (`LayerZeroAdapterERC7802`)
- **CMTAT Compatibility**: Full integration with CMTAT's standard burn/mint functionality build on ERC-3643 as well as cross-chain burn/mint functionality build on ERC-7802.
- **Automated Scripts**: Ready-to-use Foundry scripts for deployment and operations

Adapter Selection

This project provides two adapter implementations. Choose the one that matches your token's interface:

Adapter	Base Class	Token Interface	Constructor Parameters
`LayerZeroAdapterERC7802`	`OFTAdapter`	ERC-7802 (`crosschainMint`/`crosschainBurn`)	`(token, minterBurner, lzEndpoint, delegate)`
`LayerZeroAdapter`	`MintBurnOFTAdapter`	`IMintableBurnable` (`mint`/`burn`)	`(token, minterBurner, lzEndpoint, delegate)`

For CMTAT, the `minterBurner` parameter is the token address itself.

When to use which:

All CMTAT deployment versions implement the ERC-3643 interface for burning and minting. Some deployment versions also implement ERC-7802 for cross-chain transfers. If ERC-7802 is supported, the `LayerZeroAdapterERC7802` is the preferred way to use LayerZero.

- **`LayerZeroAdapterERC7802` (recommended)**: Use with CMTAT tokens that implement ERC-7802.
- **`LayerZeroAdapter`**: Use with tokens implementing only the `IMintableBurnable` interface (ERC-3643).

Both adapters include:

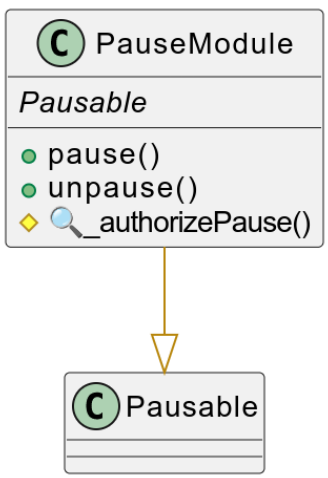
- **Pause functionality**: `pause()` and `unpause()` functions (owner only) to halt cross-chain transfers in emergencies
- **No approval required**: `approvalRequired()` returns `false` since they use mint/burn instead of transferFrom

UML

This section contains the schema (UML) of the different contracts.

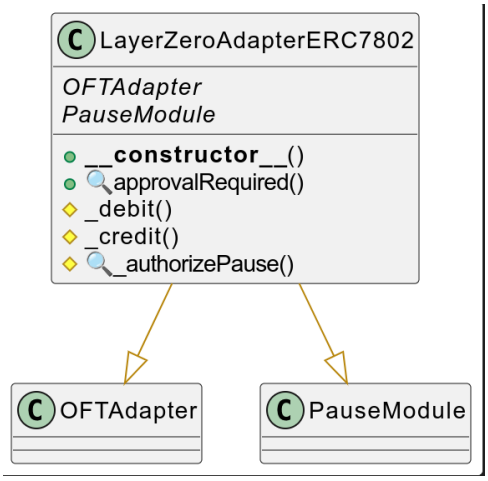
PauseModule

Public-facing functions for placing contracts into a paused state.



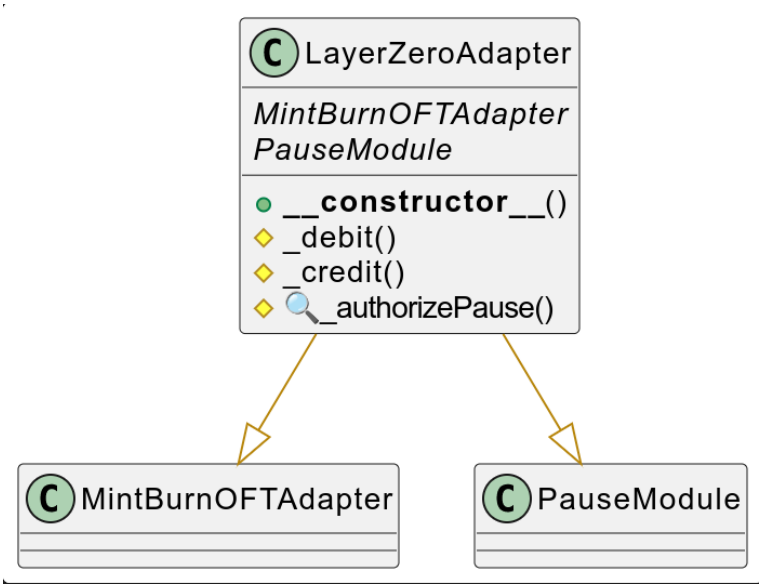
LayerZeroAdapterERC7802

LayerZero OFT adapter for tokens implementing ERC-7802



LayerZeroAdapter

LayerZero OFT adapter for tokens implementing LayerZero `IMintableBurnable` (ERC-3643 compatible)



Prerequisites

Before you begin, ensure you have the following installed:

- CMTAT v3.2.0-rc0
- Foundry (latest version)
- Node.js (v18 or higher)
- pnpm (v10.13.1 or higher)

Environment Variables

> Don't use it for production, only for testing.
> See getfoundry.sh - Key Management to securely broadcasting transactions through a script.

Create a `.env` file by running `cp .env.example .env` in the root directory with the following variables:

```
PRIVATE_KEY=your_private_key_here
ETHERSCAN_TOKEN=your_etherscan_api_key_here
```

Installation

1. **Clone the repository** (including submodules):

```
git clone <repository-url>
cd CMTAT-LayerZero
```

2. **Install dependencies**:

```
pnpm install
```

3. **Build the project**:

```
forge build
```

Configuration

Foundry Configuration

The project uses Foundry with the following key settings in `foundry.toml`:

- **Solidity Version**: 0.8.33
- **EVM Version**: Prague
- **Optimizer Runs**: 200
- **Sparse Mode**: Enabled (for faster compilation)
- **Via Ir**: Disabled

RPC Endpoints

RPC endpoints are configured in `foundry.toml`. You can add or modify endpoints in the `[rpc_endpoints]` section:

```
[rpc_endpoints]
mainnet-sepolia = "https://ethereum-sepolia-rpc.publicnode.com"
arbitrum-sepolia = "https://arbitrum-sepolia.gateway.tenderly.co"
```

Chain Configuration

Chain-specific settings (LayerZero endpoints and EIDs) are defined in `script/utils/Constants.sol`. To add a new chain:

1. Add the LayerZero endpoint address
2. Add the corresponding EID (Endpoint ID)
3. Update the mappings in the `Constants` contract

Deployment Guide

> Each command first asks for the chain name. You can use the chain names defined in `foundry.toml`.

Step 1: Deploy CMTAT Token

Deploy the CMTAT token on your source chain:

```
pnpm run deploy:token -- --broadcast --verify
```

This will:

- Deploy a new `CMTATStandalone` token contract
- Set you as the admin
- Save the deployment address to `deployments.json`

Step 2: Deploy LayerZero Adapter

Deploy the LayerZero adapter on the same chain. Choose the script based on your token's interface (see Adapter Selection):

Option A: ERC-7802 Adapter (recommended)

```
pnpm run deploy:adapter -- --broadcast --verify
```

This deploys `LayerZeroAdapterERC7802` for tokens implementing ERC-7802.

Option B: ERC-3643 Adapter

```
forge script DeployAdapterERC3643 -s "exec(string)" <chain-name> --broadcast --verify
```

This deploys `LayerZeroAdapter` for tokens implementing only ERC-3643/IMintableBurnable.

Both scripts will:

- Link the adapter to your CMTAT token
- Grant the required roles to the adapter:
  - ERC-7802 adapter: `CROSS_CHAIN_ROLE`
  - ERC-3643 adapter: `MINTER_ROLE` and `BURNER_ROLE`
- Save the adapter address to `deployments.json`

Step 3: Repeat Steps 1 and 2 for Other Chains

Step 4: Wire Adapters

Connect the adapters on both chains so they can communicate:

On source chain (e.g., arbitrum-sepolia):

```
pnpm run wire -- --broadcast --verify
```

On destination chain (e.g., mainnet-sepolia):

```
pnpm run wire -- --broadcast --verify
```

This sets up peer connections between the adapters, enabling cross-chain communication.

Usage

Minting Tokens

Mint tokens on a specific chain:

```
pnpm run token:mint -- --broadcast
```

Approving Tokens (if relevant)

> CMTAT Standard version does not require approval to perform burn/mint or crosschainmint/crosschainburn.
>
> This step is only useful if this project is used with a CMTAT version requiring the standard ERC-20 approval.

Approve the adapter to spend your tokens (required before bridging):

```
pnpm run token:approve -- --broadcast
```

Bridging Tokens

Send tokens from one chain to another:

```
pnpm run bridge -- --broadcast
```

This will:

1. Calculate the required LayerZero messaging fee
2. Burn tokens on the source chain
3. Send a cross-chain message via LayerZero
4. Mint tokens on the destination chain (after message delivery)

Note: The amount is specified without decimals. The script automatically applies the token's decimal places (6 decimals in this case).

Tracking Transactions

LayerZero Scan

All cross-chain transactions can be tracked on LayerZero Scan:

1. **Find your transaction**: Search by transaction hash from the source chain
2. **Monitor status**: Track the message delivery status
3. **View details**: See source/destination chains, amounts, and fees

Example contracts: deployments.json

Example transaction:
testnet.layerzeroscan.com/tx/0xd2182b0094d015e6670539e9206bbb141d69c1c179e5a544c1b24d6d8e10c84f

Made with CMTAT v3.1.0

Transaction Flow

1. **Source Chain**:
   - Tokens are burned via `crosschainBurn()`
   - LayerZero message is sent with destination details
   - Native gas fee is paid for cross-chain messaging
2. **LayerZero Network**:
   - Message is relayed through LayerZero's infrastructure

    - Delivery is verified by DVNs (Decentralized Verifier Networks)
3. **Destination Chain**:
    - Message is received by the adapter
    - Tokens are minted via `crosschainMint()`
    - Recipient receives the tokens

Handling Failed Transactions on Destination Chain

In rare cases, a transaction may fail on the destination chain after tokens have already been burned on the source chain. This can happen due to:

- Insufficient gas on the destination chain
- Contract execution errors
- Network congestion or LayerZero message delivery issues
- Contract paused

What happens: Tokens are burned on the source chain but not minted on the destination chain, leaving the tokens in a "stuck" state.

Recovery options:

1. **Check LayerZero Scan**: First, verify the transaction status on LayerZero Scan. Look for the message status - if it shows as "delivered" but tokens weren't minted, there may be an execution error.
2. **Retry the mint operation**: If the LayerZero message was successfully delivered but the mint failed, you may need to manually trigger the mint operation. This typically requires:
    - Access to the adapter contract on the destination chain
    - The original message payload and nonce
    - Sufficient gas to execute the mint
3. **Contact support**: If automatic recovery is not possible, you may need to:
    - Contact the LayerZero team through their Discord or support channels
    - Provide the transaction hash from the source chain
    - Provide the message GUID from LayerZero Scan
4. **Prevention**: To minimize the risk of failed transactions:
    - Ensure sufficient native tokens on both source and destination chains
    - Monitor gas prices and network conditions
    - Use appropriate gas limits in your transaction options
    - Test thoroughly on testnets before mainnet deployment

> **Important**: Always monitor your cross-chain transactions on LayerZero Scan to catch any issues early. Failed transactions may require manual intervention or support from LayerZero.

Project Structure

```
CMTAT-LayerZero/
├── src/
│   ├── LayerZeroAdapter.sol         # Adapter for IMintableBurnable tokens
(ERC-3643)
│   └── LayerZeroAdapterERC7802.sol  # Adapter for ERC-7802 tokens (default)
├── script/
│   ├── DeployToken.s.sol            # Deploy CMTAT token
```

```
│   ├── DeployAdapter.s.sol          # Deploy ERC-7802 adapter (recommended)
│   ├── DeployAdapterERC3643.s.sol   # Deploy ERC-3643 adapter
│   ├── WireAdapters.s.sol           # Connect adapters across chains
│   ├── Mint.s.sol                 # Mint tokens
│   ├── Approve.s.sol              # Approve adapter spending
│   ├── SendTokens.s.sol           # Bridge tokens cross-chain
│   └── utils/
│       ├── BaseScript.s.sol       # Base script utilities
│       ├── Constants.sol          # Chain configuration
│       └── FileHelpers.sol        # Deployment file management
├── lib/
│   ├── CMTAT/                     # CMTAT token contracts (submodule)
│   └── forge-std/                 # Foundry standard library
├── test/
│   ├── Setup.s.sol               # Shared test setup for cross-chain tests
│   ├── SendTokens.t.sol          # Cross-chain transfer tests
│   ├── DeployAdapter.t.sol       # Deployment script tests
│   └── utils/
│       └── TestBase.sol          # Shared test helpers (deploy, roles)
├── deployments.json               # Deployment addresses
├── foundry.toml                   # Foundry configuration
└── package.json                   # Node.js dependencies
```

Scripts Reference

Available Scripts

All scripts can be run using Foundry's `forge script` command. Here's a quick reference:

Script	Purpose	Example
`DeployToken`	Deploy CMTAT token	`forge script DeployToken -s "exec(string)" arbitrum-sepolia --broadcast`
`DeployAdapter`	Deploy LayerZero adapter	`forge script DeployAdapter -s "exec(string)" arbitrum-sepolia --broadcast`
`WireAdapters`	Connect adapters	`forge script WireAdapters -s "exec(string,string)" arbitrum-sepolia mainnet-sepolia --broadcast`
`Mint`	Mint tokens	`forge script Mint -s "exec(string,uint256)" arbitrum-sepolia 1000 --broadcast`
`Approve`	Approve adapter	`forge script Approve -s "exec(string)" arbitrum-sepolia --broadcast`
`SendTokens`	Bridge tokens	`forge script SendTokens -s "exec(string,string,uint256)" arbitrum-sepolia mainnet-sepolia 100 --broadcast`

Script Parameters

- **Chain names**: Use the chain names defined in `foundry.toml` (e.g., `arbitrum-sepolia`, `mainnet-sepolia`)
- **Amounts**: Specify amounts without decimals (the script applies decimals automatically)
- **Broadcast flag**: Use `--broadcast` to actually send transactions (omit for dry-run)

Testing

Run all tests:

```
forge test
```

Run specific test files:

```
# Cross-chain transfer tests
forge test --match-path test/SendTokens.t.sol -v

# Deployment script tests
forge test --match-path test/DeployAdapter.t.sol -v
```

Test Structure

File	Description
`test/SendTokens.t.sol`	Cross-chain transfer, pause, access control tests
`test/DeployAdapter.t.sol`	Deployment verification for both adapters
`test/utils/TestBase.sol`	Shared helpers: `_deployCMTAT()`, `_deployAdapterERC7802()`, `_deployAdapterERC3643()`

The shared `TestBase.sol` provides internal functions used by both test files and mirrors the deployment script logic, ensuring consistency between tests and actual deployments.

Security Considerations

1. **Private Keys**: Never expose your private keys. The `.env` file here used in this project should not be used for production. See [getfoundry.sh - Key Management]
2. **Gas Fees**: Ensure you have sufficient native tokens for gas and LayerZero messaging fees
3. **Testing**: Always test on testnets before deploying to mainnet
4. **Access Control**:
    1. The adapter requires `CROSS_CHAIN_ROLE` on the CMTAT token - ensure proper access control. If the adapter has a vulnerability or a bug, revoke the role directly on CMTAT or put the token in the pause state. `crosschainMint` and `crosschainBurn` will revert if CMTAT is in the pause state.
    2. Alternatively, ERC-20 standard approval or time-based approval can be added by modifying CMTAT codebase. In this case, each adapter user will need to approve the adapter to allow it to spend token on their behalf, which will reduce risk in case if the adapter is compromised.

5. **Adapter Pause**: Both adapters include `pause()` and `unpause()` functions (owner only). When paused, all cross-chain transfers are blocked.

Troubleshooting

Common Issues

Issue: "Insufficient funds" when bridging

- **Solution**: Ensure you have enough native tokens for both gas and LayerZero messaging fees

Issue: Tokens not arriving on destination chain

- **Solution**: Check LayerZero Scan to see if the message was delivered. Delivery can take a few minutes. If tokens were burned on the source chain but not minted on the destination, see the Handling Failed Transactions section above.

Additional Resources

- LayerZero Documentation
- CMTAT Documentation
- Foundry Book
- LayerZero Scan

Contributing

Contributions are welcome! Please ensure your code follows the project's style guidelines and includes appropriate tests.