Report

v. 1.0

Customer

Term Structure Labs

Smart Contract Audit

TMX

17th December 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	17.12.25	D. Khovratovich	Initial Draft
0.2	17.12.25	D. Khovratovich	Minor revision
1.0	17.12.25	D. Khovratovich	Release

2 Summary

This document presents the results of a smart contract audit performed by ABDK Consulting for the TermMax System, based on the TermMax documentation and the TMX.sol source. The engagement ran from 15th December through 17th December 2025 and was led by auditors Mikhail Vladimirov and Dmitry Khovratovich. The goal of the audit was to validate that the single-token implementation in `TMX.sol` aligns with the System's stated functionality and security expectations for the TMX token. The TermMax System is described as a next-generation loan AMM that simplifies DeFi borrowing, lending, and leveraging through one-click token trading and customizable pricing curves.

The contract inherits directly from `ERC20` via OpenZeppelin, sets a single, immutable total supply of one billion tokens, and mints the entire quantity to the provided admin address. The overriding of `decimals()` to return 18 keeps the standard ERC-20 interface intact, and no additional state or control flow introduces complexity beyond the well-known OpenZeppelin implementation.

Overview: No critical findings were identified during this audit. The reviewed asset consists of a straight-forward supply-and-mint flow with zero additional logic layers, fitting the stated System vision of deterministic, predictable token behavior that supports the TermMax System's fixed-rate tokens, gearing tokens, and customizable AMM parameters described in the documentation. The reliance on the audited OpenZeppelin library keeps dependency risk limited to a single, well-understood external component.

General recommendations: keep the admin key secure, especially because it initially controls the entire minted supply, and continue periodic monitoring to ensure that broader System features, such as the fixed-rate borrowing/lending mechanisms, range orders, and physical-delivery liquidation paths highlighted in the documentation, remain complementary to this foundational TMX token. Given the clean implementation, the final verdict is that the `TMX` contract is ready for deployment within the TermMax System once the surrounding operational controls are in place.

It is important to note that a security audit is not a guarantee of absolute security but rather a snapshot of the system's security posture at a specific point in time. While we strive to uncover all potential issues, the evolving nature of blockchain technology and DeFi means new vulnerabilities can emerge.

3 System overview

This section provides a high-level overview of an omnichain fungible token implementation designed to facilitate cross-chain token transfers across multiple blockchain networks. The System enables seamless movement of digital assets between different blockchain ecosystems while maintaining token fungibility and ensuring secure message passing between chains.

We were asked to review:

- Original Code

Files:

contracts/v2/tokenomics/TMX.sol

TMX.sol (line 3347-3358)

The System's primary purpose is to create a unified token standard that operates across multiple blockchain networks simultaneously. During deployment, the System mints one billion tokens exclusively on the Ethereum mainnet, establishing it as the primary issuance chain. This design enables users to transfer tokens from Ethereum to various supported destination chains and back, effectively extending the token's utility beyond a single blockchain environment. The architecture follows a burn and mint mechanism, where tokens are destroyed on the source chain and recreated on the destination chain, maintaining consistent total supply across the entire ecosystem.

The codebase is written in Solidity, utilizing version 0.8.20 as the baseline with the primary implementation contract requiring version 0.8.22 or higher. These versions provide built-in overflow protection and enhanced security features that are critical for token operations. The file structure indicates that Hardhat version 2.28.0 was employed as the development environment, which provides comprehensive tools for smart contract compilation, testing, and deployment. The use of recent Solidity versions ensures compatibility with modern Ethereum Virtual Machine implementations and enables the System to leverage the latest language safety features.

At the core of the System's architecture lies the integration with an omnichain messaging protocol that facilitates communication between different blockchain networks. This integration provides the fundamental infrastructure for cross-chain message passing, enabling the System to coordinate token transfers across disparate chains. The messaging layer handles the complexities of cross-chain communication, including message encoding, routing, verification, and delivery confirmation. The System interacts with endpoint contracts that serve as gateways to the broader omnichain network, managing both outbound messages to destination chains and inbound messages from source chains.

The token implementation adheres to the standard specification for fungible tokens, incorporating all expected functionality including transfers, approvals, and balance tracking. Built upon well-established contract libraries from a widely recognized security-focused organization, the System benefits from thoroughly audited and battle-tested code components. These libraries provide essential utilities for safe token operations, access control mechanisms, and context management. The System employs an ownership pattern that restricts certain administrative functions to a designated owner address, ensuring that critical configuration changes can only be executed by authorized parties.

The cross-chain transfer mechanism operates through a sophisticated token management system that coordinates between source and destination chains. When initiating a transfer to another chain, the System removes tokens from circulation on the source chain while simultaneously encoding a message that instructs the destination chain to create an equivalent amount of tokens for the recipient. This approach ensures that the total supply remains constant across the entire multi-chain ecosystem. The System incorporates decimal conversion logic to handle differences in token precision across various blockchain networks, preventing loss of value during transfers and maintaining accounting accuracy.

Security considerations are deeply embedded throughout the System's architecture. The implementation includes peer validation mechanisms that verify the authenticity of cross-chain messages, ensuring that only trusted counterpart contracts on other chains can trigger token operations. A simulation component enables pre-execution validation of incoming messages, allowing the detection of potential issues before they affect the System's state. Message inspection capabilities provide an additional layer of security by enabling optional validation of message content and execution parameters. The System maintains careful tracking of message nonces to ensure proper ordering and prevent replay attacks.

Configuration management within the System provides flexibility for adapting to different operational requirements across various destination chains. The architecture supports enforced execution options that can mandate minimum gas limits or required fees for cross-chain operations, protecting against insufficient execution resources on destination chains. The System allows configuration of library contracts responsible for message transmission and reception, enabling upgrades to the underlying messaging infrastructure without requiring changes to the core token logic. Delegate addresses can be assigned to manage configuration updates on behalf of the token contract, separating operational concerns from ownership.

The implementation distinguishes between different types of cross-chain operations, supporting both simple token transfers and more complex composed messages that can trigger additional logic on destination chains. This extensibility enables future enhancements where token transfers could initiate automated actions upon arrival at the destination chain. The System handles both native cryptocurrency fees and alternative token-based fee payment mechanisms for cross-chain messaging costs, providing users with flexibility in how they pay for cross-chain operations. Fee estimation capabilities allow users to determine the cost of cross-chain transfers before execution, preventing unexpected transaction failures due to insufficient fee provision.

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check whether the code actually does what it is supposed to do, whether the algorithms are optimal and correct, and whether proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** cover code style, best practices and general improvements.

5 Our findings

No vulnerabilities have been found.

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 300 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting