StabIR

STABIR

SMART CONTRACT AUDIT

zokyo

July 8th 2022 | v. 1.0

Security Audit Score

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

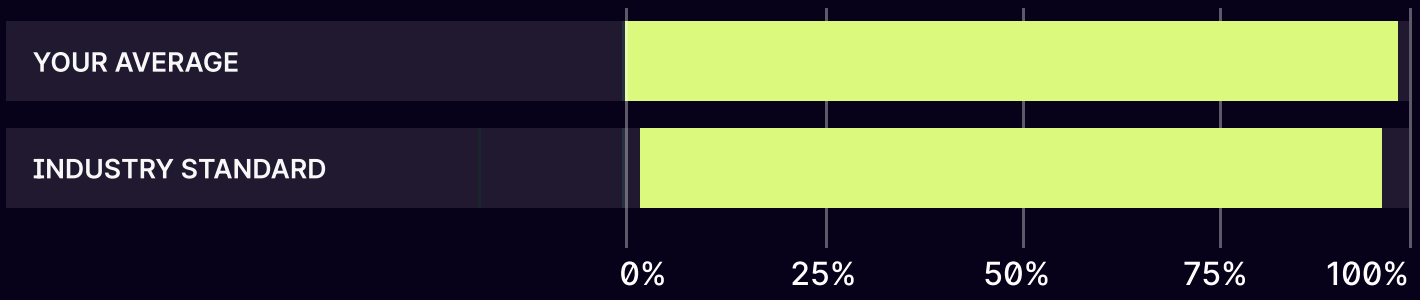
TECHNICAL SUMMARY

This document outlines the overall security of the StablR smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the StablR smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code meets necessary security requirements and conducts sufficient coverage.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the StablR team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the StablR repository.
https://github.com/StablR/stablr-tokens

Last audited commit: 4501f0f90e3d56b651303c2e60aedcf0647af442

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- upgradeability/AdminUpgradeabilityProxy.sol
- upgradeability/Proxy.sol
- upgradeability/UpgradeabilityProxy.sol
- v1/AbstractFiatTokenV1.sol
- v1/Blacklistable.sol
- v1/FiatTokenProxy.sol
- v1/FiatTokenV1.sol
- v1/Ownable.sol
- v1/Pausable.sol
- v1.1/FiatTokenV1_1.sol.sol
- v1.1/Rescuable.sol
- v2/AbstractFiatTokenV2.sol
- v2/EIP712Domain.sol
- v2/EIP2612.sol
- v2/EIP3009.sol
- v2/FiatTokenUtil.sol
- v2/FiatTokenV2_1.sol
- v2/FiatTokenV2.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of StablR smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.	04	Thorough, manual review of the codebase, line-by-line.

Executive Summary

StablR protocol represents the set of contracts for FiatToken. It is ERC20 compatible token, upgradeable, with several versions of it.
V1 - standard ERC20 token with:

- Ownable functionality which controls MasterMinter role;
- with mint/burn functionality controlled by minters which are set by the MasterMinter;
- with Pausable functionality for mint/burn, approves and transfers;
- with Blacklist functionality for mint/burn, approves and transfers.

V1_1 - V1 token extended with Rescue functionality for stucked ERC20 tokens on the contract
V2 - V1_1 token extended with EIP712 (domain), EIP2612 (permit) and EIP3009 (authorized send/receive)
V2_1 - V2 extended with the version "2" of the token.

Auditors team has carefully checked the upgradability functionality of the token and all standard and sub-standard features added to ERC20. All unclear functionality was verified with the StablR team. Contracts contain sufficient unit-test coverage together.

Though, with all security measures taken there is still the uncertainty with the token upgrade process. Nevertheless, the StablR team, is acknowledged of that and has taken the responsibility to provide the maximum control over the token upgrade procedure and monitor its status after the upgrade.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | ACKNOWLEDGED

Unprotected initializer.
v2/FiatTokenV2.sol, initializeV2()
v2/FiatTokenV2_1.sol, initializeV2_1()
Function serves as additional initializer after the token upgrade. Though, since the second version is not initialized in an atomic way (like it was for the first version initialized during the Proxy contract creation) and requires a separate call - it should be protected against unauthorized calls (e.g. restricted to the owner only). Or, there should be a proof that function will be called through the AdminProxy upgradeAndCall() (e.g. with the appropriate deployment script).
The issue is marked as high, since the initialization will change the storage to contain a new name and update initialization version. Though it is not critical, since it does not change critical for the protocol storage.

Recommendation:
Add appropriate protection against unauthorized call, or provide a deployment/upgrade script with transparent upgradeAndCall() call.

Post-audit:
StablR team is acknowledged of the issue and stated that is the acceptable risk. StablR team will mitigate it by running deployment and initialization before announcement, after that it is immutable. Additionally StablR team will carefully check what happens after deployment, as they can spot what activity takes place subsequently.

Additional protection checks.
v1/Pausable.sol.
Additional checks to not allow pause already paused or unpause already unpaused (for pause() and unpause()).
v1/Blacklistable.sol
Additional checks to not block already blocked or unblock already unblocked account (for blacklist() and unBlacklist()).
Such checks will increase the quality of the code and will stop misleading extra transactions for pause/unpause, block/unblock.

Recommendation:
Consider adding additional checks.

Post-audit:
StablR team stated that is acceptable risk, since only StablR is able to do these actions. Manually the risk is to pay additional gas fees because the repeating actions continue to respond with `200` successful while the contract or address is already paused or blacklisted. When automating these actions, StablR team has several ways to tackle this (both on chain as off chain)

INFORMATIONAL | VERIFIED

Master minter is not a minter by default.
FiatTokenV1.sol
This is not a security issue, though since masterMinter has the role by this name, it supposed to have default rights for minting. Though the initializer does not add masterMinter as default minter.

Recommendation:
Consider adding masterMinter as default minter as well.

Post-audit:
StablR team has verified that masterMinter is considered to be a control role instead of an actionable role, therefore it stays as is and is only able to assign Minter and allowance of the Minter.

INFORMATIONAL | VERIFIED

Extra tokens rescue, but not Ether.
Rescuable.sol
The contract has functionality to rescue extra token sent by mistake to the FiatToken contract. Though there is no "mirror" functionality for Ether sent by mistake.

Recommendation:
Consider adding Eth rescue functionality or verify that this functionality is not needed for the contract.

Post-audit:
StablR team has confirmed that they are keeping track of activity and can mitigate any Eth sent by mistake by Upgradability of the contract and adding this functionality if this would occur.

INFORMATIONAL | VERIFIED

Non-standard field.
FiatToken contract
The contract inherits ERC20 standard interface, though it contains non-standard field "currency". It seems to be readonly and not used throughout the contracts, thus needs verification from the StablR team for the purpose of the field.

Recommendation:
Verify correct usage of the additional field.

Post-audit:
StablR team has confirmed that this additional field is correctly used to make sure users of the contract can additionaly distinguish usage of the contract based on the currency.

INFORMATIONAL | VERIFIED

Cancel on behalf.
v2/FiatTokenV2.sol, cancelAuthorization()
v2/EIP3009.sol, _cancelAuthorization()
Function allows cancel on behalf of the authorizer in case if the user has access to the signature of the authorizer. Thus, in case of the "man in the middle" attack, when the user stores cancel receipts (which should be regular practice - to store potential cancel receipt for each provided transfer/receive signature), the 3rd party user may cancel valid requests on behalf.
The issue is marked as info due to the complexity of the attack, and due to the fact that in most cases it means leaked private key, which is out of the scope of the audit. Though, the point of allowance cancels on behalf refers to the business logic of the protocol, thus it should be reflected in the report.

Recommendation:
Verify, that cancel on behalf is valid within the business logic of the protocol or add appropriate check against msg.sender to _cancelAuthorization() function.

Post-audit:
The team has confirmed the correctness of the business-logic

	FiatTokenV1.sol	FiatTokenV1_1.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	FiatTokenV2.sol	FiatTokenV2_1.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by StablR team

As part of our work assisting StablR team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the StablR team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the StablR team. All of them were also carefully checked by the auditors' team.

Contract: FiatTokenV1: MintController_Tests MintController

- ✓ should mint through mint controller (2527ms)
- ✓ initial state (2356ms)
- ✓ only owner configures controller (76ms)
- ✓ remove controller (4573ms)
- ✓ only owner can remove controller (47ms)
- ✓ sets token
- ✓ only owner sets token (125ms)
- ✓ remove minter (6046ms)
- ✓ only controller removes a minter (40ms)
- ✓ only controller configures a minter (47ms)
- ✓ increment minter allowance (4238ms)
- ✓ only controller increments allowance (46ms)
- ✓ only active minters can have allowance incremented (2195ms)
- ✓ decrement minter allowance (4200ms)
- ✓ only controller decrements allowance (43ms)
- ✓ only active minters can have allowance decremented (2459ms)

Contract: FiatTokenV1_1: MintController_Tests MintController

- ✓ should mint through mint controller (2297ms)
- ✓ initial state (1966ms)
- ✓ only owner configures controller
- ✓ remove controller (4197ms)
- ✓ only owner can remove controller (40ms)
- ✓ sets token
- ✓ only owner sets token (61ms)
- ✓ remove minter (5212ms)
- ✓ only controller removes a minter (42ms)
- ✓ only controller configures a minter (41ms)
- ✓ increment minter allowance (3859ms)

- ✓ only controller increments allowance (40ms)
- ✓ only active minters can have allowance incremented (2012ms)
- ✓ decrement minter allowance (4072ms)
- ✓ only controller decrements allowance (49ms)
- ✓ only active minters can have allowance decremented (2059ms)

Contract: FiatTokenV2: MintController_Tests MintController

- ✓ should mint through mint controller (1951ms)
- ✓ initial state (1778ms)
- ✓ only owner configures controller (38ms)
- ✓ remove controller (3745ms)
- ✓ only owner can remove controller (41ms)
- ✓ sets token
- ✓ only owner sets token (69ms)
- ✓ remove minter (4676ms)
- ✓ only controller removes a minter (41ms)
- ✓ only controller configures a minter (42ms)
- ✓ increment minter allowance (4135ms)
- ✓ only controller increments allowance
- ✓ only active minters can have allowance incremented (2084ms)
- ✓ decrement minter allowance (4098ms)
- ✓ only controller decrements allowance (41ms)
- ✓ only active minters can have allowance decremented (2064ms)

Contract: FiatTokenV1: MINTp0_ArgumentTests MintController

- ✓ arg000 transferOwnership(msg.sender) works (1836ms)
- ✓ arg001 transferOwnership(0) reverts (41ms)
- ✓ arg002 transferOwnership(owner) works (1803ms)
- ✓ arg003 configureController(0, M) throws (44ms)
- ✓ arg004 configureController(msg.sender, M) works (1873ms)
- ✓ arg005 configureController(M, M) works (1960ms)
- ✓ arg006 configureController(C, 0) throws (38ms)
- ✓ arg007 removeController(0) throws (1918ms)
- ✓ arg008 setMinterManager(0) works (1945ms)
- ✓ arg009 setMinterManager(oldMinterManager) works (1918ms)
- ✓ arg010 setMinterManager(user_account) works (1903ms)
- ✓ arg011 setMinterManager(newToken) works (1953ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (1974ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (1976ms)
- ✓ arg014 configureMinter(MAX) works (2034ms)
- ✓ arg015 incrementMinterAllowance(0) throws (95ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (2252ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (225ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (214ms)

- ✓ arg019 configureController(0, 0) throws (96ms)
- ✓ arg020 removeController(C) works (2789ms)
- ✓ arg021 removeController throws if worker is already address(0) (2364ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2191ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2083ms)
- ✓ arg024 decrementMinterAllowance(0) throws (131ms)

Contract: FiatTokenV1_1: MINTp0_ArgumentTests MintController

- ✓ arg000 transferOwnership(msg.sender) works (1873ms)
- ✓ arg001 transferOwnership(0) reverts (46ms)
- ✓ arg002 transferOwnership(owner) works (2138ms)
- ✓ arg003 configureController(0, M) throws (51ms)
- ✓ arg004 configureController(msg.sender, M) works (2301ms)
- ✓ arg005 configureController(M, M) works (2732ms)
- ✓ arg006 configureController(C, 0) throws (62ms)
- ✓ arg007 removeController(0) throws (1979ms)
- ✓ arg008 setMinterManager(0) works (2514ms)
- ✓ arg009 setMinterManager(oldMinterManager) works (2585ms)
- ✓ arg010 setMinterManager(user_account) works (2361ms)
- ✓ arg011 setMinterManager(newToken) works (2453ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (2783ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (2551ms)
- ✓ arg014 configureMinter(MAX) works (2566ms)
- ✓ arg015 incrementMinterAllowance(0) throws (138ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (2401ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (150ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (175ms)
- ✓ arg019 configureController(0, 0) throws (48ms)
- ✓ arg020 removeController(C) works (3001ms)
- ✓ arg021 removeController throws if worker is already address(0) (2801ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2639ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2537ms)
- ✓ arg024 decrementMinterAllowance(0) throws (148ms)

Contract: FiatTokenV2: MINTp0_ArgumentTests MintController

- ✓ arg000 transferOwnership(msg.sender) works (2171ms)
- ✓ arg001 transferOwnership(0) reverts (40ms)
- ✓ arg002 transferOwnership(owner) works (1973ms)
- ✓ arg003 configureController(0, M) throws (41ms)
- ✓ arg004 configureController(msg.sender, M) works (2136ms)
- ✓ arg005 configureController(M, M) works (2011ms)
- ✓ arg006 configureController(C, 0) throws (42ms)
- ✓ arg007 removeController(0) throws (1938ms)
- ✓ arg008 setMinterManager(0) works (1996ms)

- ✓ arg009 setMinterManager(oldMinterManager) works (2462ms)
- ✓ arg010 setMinterManager(user_account) works (2002ms)
- ✓ arg011 setMinterManager(newToken) works (2038ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (1989ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (2060ms)
- ✓ arg014 configureMinter(MAX) works (2080ms)
- ✓ arg015 incrementMinterAllowance(0) throws (124ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (2073ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (131ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (134ms)
- ✓ arg019 configureController(0, 0) throws (48ms)
- ✓ arg020 removeController(C) works (2146ms)
- ✓ arg021 removeController throws if worker is already address(0) (2213ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2209ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2072ms)
- ✓ arg024 decrementMinterAllowance(0) throws (92ms)

Contract: FiatTokenV1: MINTp0_ArgumentTests MasterMinter

- ✓ arg000 transferOwnership(msg.sender) works (2021ms)
- ✓ arg001 transferOwnership(0) reverts
- ✓ arg002 transferOwnership(owner) works (2021ms)
- ✓ arg003 configureController(0, M) throws
- ✓ arg004 configureController(msg.sender, M) works (1850ms)
- ✓ arg005 configureController(M, M) works (1933ms)
- ✓ arg006 configureController(C, 0) throws
- ✓ arg007 removeController(0) throws (1985ms)
- ✓ arg008 setMinterManager(0) works (1858ms)
- ✓ arg009 setMinterManager(oldMinterManager) works (2095ms)
- ✓ arg010 setMinterManager(user_account) works (1950ms)
- ✓ arg011 setMinterManager(newToken) works (2069ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (1933ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (2102ms)
- ✓ arg014 configureMinter(MAX) works (1959ms)
- ✓ arg015 incrementMinterAllowance(0) throws (72ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (1894ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (188ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (218ms)
- ✓ arg019 configureController(0, 0) throws (50ms)
- ✓ arg020 removeController(C) works (2426ms)
- ✓ arg021 removeController throws if worker is already address(0) (2213ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2855ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2407ms)
- ✓ arg024 decrementMinterAllowance(0) throws (139ms)

Contract: FiatTokenV1_1: MINTp0_ArgumentTests MasterMinter

- ✓ arg000 transferOwnership(msg.sender) works (2586ms)
- ✓ arg001 transferOwnership(0) reverts (53ms)
- ✓ arg002 transferOwnership(owner) works (2135ms)
- ✓ arg003 configureController(0, M) throws (48ms)
- ✓ arg004 configureController(msg.sender, M) works (2947ms)
- ✓ arg005 configureController(M, M) works (3072ms)
- ✓ arg006 configureController(C, 0) throws (86ms)
- ✓ arg007 removeController(0) throws (2454ms)
- ✓ arg008 setMinterManager(0) works (3151ms)
- ✓ arg009 setMinterManager(oldMinterManager) works (2682ms)
- ✓ arg010 setMinterManager(user_account) works (2460ms)
- ✓ arg011 setMinterManager(newToken) works (2510ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (2502ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (2291ms)
- ✓ arg014 configureMinter(MAX) works (2293ms)
- ✓ arg015 incrementMinterAllowance(0) throws (108ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (2278ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (188ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (155ms)
- ✓ arg019 configureController(0, 0) throws (49ms)
- ✓ arg020 removeController(C) works (2338ms)
- ✓ arg021 removeController throws if worker is already address(0) (2608ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2440ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2092ms)
- ✓ arg024 decrementMinterAllowance(0) throws (77ms)

Contract: FiatTokenV2: MINTp0_ArgumentTests MasterMinter

- ✓ arg000 transferOwnership(msg.sender) works (1980ms)
- ✓ arg001 transferOwnership(0) reverts
- ✓ arg002 transferOwnership(owner) works (2143ms)
- ✓ arg003 configureController(0, M) throws (39ms)
- ✓ arg004 configureController(msg.sender, M) works (1953ms)
- ✓ arg005 configureController(M, M) works (1945ms)
- ✓ rg006 configureController(C, 0) throws (40ms)
- ✓ arg007 removeController(0) throws (2073ms)
- ✓ arg008 setMinterManager(0) works (1992ms)
- ✓ arg009 setMinterManager(oldMinterManager) works (1956ms)
- ✓ arg010 setMinterManager(user_account) works (2047ms)
- ✓ arg011 setMinterManager(newToken) works (2098ms)
- ✓ arg012 configureMinter(0) sets allowance to 0 (2052ms)
- ✓ arg013 configureMinter(oldAllowance) makes no changes (2211ms)
- ✓ arg014 configureMinter(MAX) works (2145ms)

- ✓ arg015 incrementMinterAllowance(0) throws (79ms)
- ✓ arg016 incrementMinterAllowance(oldAllowance) doubles the allowance (2059ms)
- ✓ arg017 incrementMinterAllowance(MAX) throws (153ms)
- ✓ arg018 incrementMinterAllowance(BIG) throws (123ms)
- ✓ arg019 configureController(0, 0) throws (40ms)
- ✓ arg020 removeController(C) works (2305ms)
- ✓ arg021 removeController throws if worker is already address(0) (2217ms)
- ✓ arg022 decrementMinterAllowance(oldAllowance) sets the allowance to 0 (2119ms)
- ✓ arg023 decrementMinterAllowance(MIN) sets the allowance to 0 (2232ms)
- ✓ arg024 decrementMinterAllowance(0) throws (79ms)

Contract: FiatTokenV1: MINTp0_BasicTests MintController

- ✓ bt001 Constructor - owner is msg.sender (85ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2079ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (42ms)
- ✓ bt004 configureController works when owner is msg.sender (1981ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2160ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (1941ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (2024ms)
- ✓ bt008 removeController works when owner is msg.sender (3892ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (4065ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller
- ✓ bt011 removeMinter sets minters[M] to 0 (4044ms)
- ✓ bt012 configureMinter reverts when msg.sender is not a controller (41ms)
- ✓ bt013 configureMinter works when controllers[msg.sender]=M (1951ms)
- ✓ bt014 incrementMinterAllowance reverts if msg.sender is not a controller (58ms)
- ✓ bt015 incrementMinterAllowance works when controllers[msg.sender]=M (2112ms)
- ✓ bt016 Constructor sets all controllers to 0 (1933ms)
- ✓ bt017 removeController reverts when controllers[C] is 0 (2011ms)
- ✓ bt018 removeController removes an arbitrary controller (4133ms)
- ✓ bt019 configureController works when controller[C]=0 (2140ms)
- ✓ bt020 configureController works when controller[C] != 0 (3986ms)
- ✓ bt021 configureController(C,C) works (2031ms)
- ✓ bt022 configureController works when setting controller[C]=msg.sender (2191ms)
- ✓ bt023 configureController(C, newM) works when controller[C]=newM (4137ms)
- ✓ bt024 Constructor sets minterManager
- ✓ bt026 setMinterManager(x) works when existing minterManager != 0 (3960ms)
- ✓ bt027 setMinterManager(x) works when x = msg.sender (2093ms)
- ✓ bt028 setMinterManager(x) works when x = minterManager (1929ms)
- ✓ bt030 removeMinter reverts when minterManager is 0 (2219ms)
- ✓ bt031 removeMinter reverts when minterManager is a user account (2033ms)
- ✓ bt032 removeMinter works when minterManager is ok (4426ms)
- ✓ bt034 configureMinter reverts when minterManager is a user account (2439ms)

- ✓ bt035 configureMinter works when minterManager is ok (2268ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2120ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2037ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2094ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2192ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2291ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2294ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2170ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2537ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (681ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (293ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2882ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2581ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2248ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2334ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2752ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2099ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2223ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (2404ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2183ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (71ms)
- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (2231ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (2314ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (2267ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2231ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2436ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2364ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (2538ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (2301ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2716ms)

Contract: FiatTokenV1_1: MINTp0_BasicTests MintController

- ✓ bt001 Constructor - owner is msg.sender (77ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2058ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (42ms)
- ✓ bt004 configureController works when owner is msg.sender (1887ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2081ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (2019ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (1938ms)
- ✓ bt008 removeController works when owner is msg.sender (4140ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (3929ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller (48ms)

- ✓ bt011 removeMinter sets minters[M] to 0 (4057ms)
- ✓ bt012 configureMinter reverts when msg.sender is not a controller (41ms)
- ✓ bt013 configureMinter works when controllers[msg.sender]=M (1989ms)
- ✓ bt014 incrementMinterAllowance reverts if msg.sender is not a controller (45ms)
- ✓ bt015 incrementMinterAllowance works when controllers[msg.sender]=M (2284ms)
- ✓ bt016 Constructor sets all controllers to 0 (2487ms)
- ✓ bt017 removeController reverts when controllers[C] is 0 (2274ms)
- ✓ bt018 removeController removes an arbitrary controller (4594ms)
- ✓ bt019 configureController works when controller[C]=0 (2127ms)
- ✓ bt020 configureController works when controller[C] != 0 (4323ms)
- ✓ bt021 configureController(C,C) works (2226ms)
- ✓ bt022 configureController works when setting controller[C]=msg.sender (2188ms)
- ✓ bt023 configureController(C, newM) works when controller[C]=newM (3939ms)
- ✓ bt024 Constructor sets minterManager
- ✓ bt026 setMinterManager(x) works when existing minterManager != 0 (3830ms)
- ✓ bt027 setMinterManager(x) works when x = msg.sender (2158ms)
- ✓ bt028 setMinterManager(x) works when x = minterManager (2022ms)
- ✓ bt030 removeMinter reverts when minterManager is 0 (2054ms)
- ✓ bt031 removeMinter reverts when minterManager is a user account (2101ms)
- ✓ bt032 removeMinter works when minterManager is ok (4184ms)
- ✓ bt034 configureMinter reverts when minterManager is a user account (2106ms)
- ✓ bt035 configureMinter works when minterManager is ok (2216ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2095ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2145ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2267ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2405ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2731ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2479ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2248ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2375ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (211ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (203ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2393ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2416ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2380ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2401ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2376ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2358ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2357ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (2527ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2431ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (57ms)

- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (2249ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (2246ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (2274ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2309ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2399ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2407ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (3119ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (2615ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2627ms)

Contract: FiatTokenV2: MINTp0_BasicTests MintController

- ✓ bt001 Constructor - owner is msg.sender (83ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2137ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (54ms)
- ✓ bt004 configureController works when owner is msg.sender (2200ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2191ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (2187ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (2656ms)
- ✓ bt008 removeController works when owner is msg.sender (4570ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (4807ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller (45ms)
- ✓ bt011 removeMinter sets minters[M] to 0 (4430ms)
- ✓ bt012 configureMinter reverts when msg.sender is not a controller (41ms)
- ✓ bt013 configureMinter works when controllers[msg.sender]=M (2267ms)
- ✓ bt014 incrementMinterAllowance reverts if msg.sender is not a controller (55ms)
- ✓ bt015 incrementMinterAllowance works when controllers[msg.sender]=M (2339ms)
- ✓ bt016 Constructor sets all controllers to 0 (2272ms)
- ✓ bt017 removeController reverts when controllers[C] is 0 (2410ms)
- ✓ bt018 removeController removes an arbitrary controller (5004ms)
- ✓ bt019 configureController works when controller[C]=0 (2294ms)
- ✓ bt020 configureController works when controller[C] != 0 (4579ms)
- ✓ bt021 configureController(C,C) works (2366ms)
- ✓ bt022 configureController works when setting controller[C]=msg.sender (2458ms)
- ✓ bt023 configureController(C, newM) works when controller[C]=newM (4034ms)
- ✓ bt024 Constructor sets minterManager
- ✓ bt026 setMinterManager(x) works when existing minterManager != 0 (4092ms)
- ✓ bt027 setMinterManager(x) works when x = msg.sender (2078ms)
- ✓ bt028 setMinterManager(x) works when x = minterManager (1998ms)
- ✓ bt030 removeMinter reverts when minterManager is 0 (2056ms)
- ✓ bt031 removeMinter reverts when minterManager is a user account (2047ms)
- ✓ bt032 removeMinter works when minterManager is ok (4239ms)
- ✓ bt034 configureMinter reverts when minterManager is a user account (2674ms)

- ✓ bt035 configureMinter works when minterManager is ok (2430ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2457ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2477ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2445ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2588ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2627ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2385ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2259ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2553ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (213ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (194ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2485ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2667ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2477ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2477ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2427ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2501ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2699ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (2553ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2387ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (47ms)
- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (2456ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (2459ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (2368ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2435ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2498ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2422ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (2654ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (2577ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2539ms)

Contract: FiatTokenV1: MINTp0_BasicTests MasterMinter

- ✓ bt001 Constructor - owner is msg.sender (95ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2200ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (44ms)
- ✓ bt004 configureController works when owner is msg.sender (2200ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2327ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (2378ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (2252ms)
- ✓ bt008 removeController works when owner is msg.sender (4667ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (4879ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller (41ms)

- ✓ bt011 removeMinter sets minters[M] to 0 (4669ms)
- ✓ bt012 configureMinter reverts when msg.sender is not a controller (48ms)
- ✓ bt013 configureMinter works when controllers[msg.sender]=M (2219ms)
- ✓ bt014 incrementMinterAllowance reverts if msg.sender is not a controller (51ms)
- ✓ bt015 incrementMinterAllowance works when controllers[msg.sender]=M (2410ms)
- ✓ bt016 Constructor sets all controllers to 0 (2200ms)
- ✓ bt017 removeController reverts when controllers[C] is 0 (2254ms)
- ✓ bt018 removeController removes an arbitrary controller (4621ms)
- ✓ bt019 configureController works when controller[C]=0 (2335ms)
- ✓ bt020 configureController works when controller[C] != 0 (4683ms)
- ✓ bt021 configureController(C,C) works (2305ms)
- ✓ bt022 configureController works when setting controller[C]=msg.sender (2292ms)
- ✓ bt023 configureController(C, newM) works when controller[C]=newM (4588ms)
- ✓ bt024 Constructor sets minterManager
- ✓ bt026 setMinterManager(x) works when existing minterManager != 0 (4485ms)
- ✓ bt027 setMinterManager(x) works when x = msg.sender (2426ms)
- ✓ bt028 setMinterManager(x) works when x = minterManager (2228ms)
- ✓ bt030 removeMinter reverts when minterManager is 0 (2200ms)
- ✓ bt031 removeMinter reverts when minterManager is a user account (2558ms)
- ✓ bt032 removeMinter works when minterManager is ok (5126ms)
- ✓ bt034 configureMinter reverts when minterManager is a user account (2414ms)
- ✓ bt035 configureMinter works when minterManager is ok (2490ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2409ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2570ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2953ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (3349ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2531ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2889ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (3483ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (3920ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (279ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (778ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (3181ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2695ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2517ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2792ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2697ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2518ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2772ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (3906ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (3844ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (68ms)

- ✓ bt035 configureMinter works when minterManager is ok (2385ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2471ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2437ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2456ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2430ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2511ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2361ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2517ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2569ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (199ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (192ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2485ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2681ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2558ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2651ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2597ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2504ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2779ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (3773ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2951ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (63ms)
- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (2804ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (2424ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (2398ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2506ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2743ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2611ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (3089ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (3743ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2680ms)

Contract: FiatTokenV2: MINTp0_BasicTests MasterMinter

- ✓ bt001 Constructor - owner is msg.sender (432ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2924ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (47ms)
- ✓ bt004 configureController works when owner is msg.sender (2370ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2721ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (2527ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (2192ms)
- ✓ bt008 removeController works when owner is msg.sender (4363ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (4677ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller (50ms)

- ✓ bt035 configureMinter works when minterManager is ok (2385ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2471ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2437ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2456ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2430ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2511ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2361ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2517ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2569ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (199ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (192ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2485ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2681ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2558ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (2651ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2597ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2504ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2779ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (3773ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2951ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (63ms)
- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (2804ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (2424ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (2398ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2506ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2743ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2611ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (3089ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (3743ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2680ms)

Contract: FiatTokenV2: MINTp0_BasicTests MasterMinter

- ✓ bt001 Constructor - owner is msg.sender (432ms)
- ✓ bt002 transferOwnership works when owner is msg.sender (2924ms)
- ✓ bt003 transferOwnership reverts when owner is not msg.sender (47ms)
- ✓ bt004 configureController works when owner is msg.sender (2370ms)
- ✓ bt005 configureController reverts when owner is not msg.sender (2721ms)
- ✓ bt006 setMinterManager works when owner is msg.sender (2527ms)
- ✓ bt007 setMinterManager reverts when owner is not msg.sender (2192ms)
- ✓ bt008 removeController works when owner is msg.sender (4363ms)
- ✓ bt009 removeController reverts when owner is not msg.sender (4677ms)
- ✓ bt010 removeMinter reverts when msg.sender is not a controller (50ms)

- ✓ bt011 removeMinter sets minters[M] to 0 (4810ms)
- ✓ bt012 configureMinter reverts when msg.sender is not a controller (74ms)
- ✓ bt013 configureMinter works when controllers[msg.sender]=M (2303ms)
- ✓ bt014 incrementMinterAllowance reverts if msg.sender is not a controller (43ms)
- ✓ bt015 incrementMinterAllowance works when controllers[msg.sender]=M (2173ms)
- ✓ bt016 Constructor sets all controllers to 0 (2067ms)
- ✓ bt017 removeController reverts when controllers[C] is 0 (2046ms)
- ✓ bt018 removeController removes an arbitrary controller (4101ms)
- ✓ bt019 configureController works when controller[C]=0 (2072ms)
- ✓ bt020 configureController works when controller[C] != 0 (4164ms)
- ✓ bt021 configureController(C,C) works (2314ms)
- ✓ bt022 configureController works when setting controller[C]=msg.sender (2181ms)
- ✓ bt023 configureController(C, newM) works when controller[C]=newM (4710ms)
- ✓ bt024 Constructor sets minterManager
- ✓ bt026 setMinterManager(x) works when existing minterManager != 0 (3892ms)
- ✓ bt027 setMinterManager(x) works when x = msg.sender (1981ms)
- ✓ bt028 setMinterManager(x) works when x = minterManager (2165ms)
- ✓ bt030 removeMinter reverts when minterManager is 0 (2435ms)
- ✓ bt031 removeMinter reverts when minterManager is a user account (2298ms)
- ✓ bt032 removeMinter works when minterManager is ok (4741ms)
- ✓ bt034 configureMinter reverts when minterManager is a user account (2481ms)
- ✓ bt035 configureMinter works when minterManager is ok (2419ms)
- ✓ bt037 incrementMinterAllowance reverts when minterManager is a user account (2121ms)
- ✓ bt038 incrementMinterAllowance works when minterManager is ok (2168ms)
- ✓ bt039 configureMinter(M, amt) works when minterManager.isMinter(M)=false (2116ms)
- ✓ bt040 configureMinter(M, amt) works when minterManager.isMinter(M)=true (2434ms)
- ✓ bt041 removeMinter(M) works when minterManager.isMinter(M)=false (2290ms)
- ✓ bt042 removeMinter(M) works when minterManager.isMinter(M)=true (2586ms)
- ✓ bt043 incrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2468ms)
- ✓ bt044 incrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2393ms)
- ✓ bt045 constructor - minterManager.isMinter[ALL] is false (202ms)
- ✓ bt046 constructor - minterManager.minterAllowance[ALL] = 0 (207ms)
- ✓ bt047 incrementMinterAllowance(M,amt) works when minterAllowance is 0 (2389ms)
- ✓ bt048 incrementMinterAllowance(M, amt) works when minterAllowance > 0 (2360ms)
- ✓ bt049 incrementMinterAllowance(M,amt) reverts when minterAllowance[M] + amt > 2^256 (2668ms)
- ✓ bt050 configureMinter(M,amt) works when minterAllowance=0 (3021ms)
- ✓ bt051 configureMinter(M,amt) works when minterAllowance>0 (2969ms)
- ✓ bt052 configureMinter(M,amt) works when amt+minterAllowance > 2^256 (2439ms)
- ✓ bt053 removeMinter works when the minterAllowance is 0 (2274ms)
- ✓ bt054 removeMinter works when the minterAllowance is not zero (2420ms)
- ✓ bt055 removeMinter works when the minterAllowance is big (2728ms)
- ✓ bt056 decrementMinterAllowance reverts if msg.sender is not a controller (44ms)

- ✓ bt057 decrementMinterAllowance works when controllers[msg.sender]=M (3530ms)
- ✓ bt058 decrementMinterAllowance reverts when minterManager is 0 (3260ms)
- ✓ bt059 decrementMinterAllowance reverts when minterManager is a user account (3304ms)
- ✓ bt060 decrementMinterAllowance works when minterManager is ok (2917ms)
- ✓ bt061 decrementMinterAllowance(M, amt) reverts when minterManager.isMinter(M)=false (2447ms)
- ✓ bt062 decrementMinterAllowance(M, amt) works when minterManager.isMinter(M)=true (2418ms)
- ✓ bt063 decrementMinterAllowance(M,amt) works when minterAllowance is MAX (2620ms)
- ✓ bt064 decrementMinterAllowance(M, amt) works when minterAllowance > 0 (2504ms)
- ✓ bt065 decrementMinterAllowance(M,amt) sets allowance to 0 when minterAllowance[M] - amt < 0 (2419ms)

Contract: FiatTokenV1: MINTp0_EndToEndTests MintController

- ✓ te000 New owner can configure controllers (2213ms)
- ✓ ete001 New owner can remove controllers (2224ms)
- ✓ ete002 New controller can configure minter (2391ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4444ms)
- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (4393ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (4770ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2940ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2150ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (2352ms)
- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (2670ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (2756ms)
- ✓ ete011 New controller can increment minter allowance (2934ms)
- ✓ ete012 New controller can remove minter (2472ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (4568ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (2495ms)
- ✓ ete015 Configure a minter and make sure it can mint (2451ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2540ms)
- ✓ ete017 Configure a minter and make sure it can burn (2618ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2449ms)

Contract: FiatTokenV1_1: MINTp0_EndToEndTests MintController

- ✓ ete000 New owner can configure controllers (2400ms)
- ✓ ete001 New owner can remove controllers (2264ms)
- ✓ ete002 New controller can configure minter (2348ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4418ms)

- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (4764ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (4523ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2284ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2296ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (3029ms)
- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (3782ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (3772ms)
- ✓ ete011 New controller can increment minter allowance (2763ms)
- ✓ ete012 New controller can remove minter (2625ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (5632ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (2582ms)
- ✓ ete015 Configure a minter and make sure it can mint (2522ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2746ms)
- ✓ ete017 Configure a minter and make sure it can burn (2782ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2796ms)

Contract: FiatTokenV2: MINTp0_EndToEndTests MintController

- ✓ ete000 New owner can configure controllers (2527ms)
- ✓ ete001 New owner can remove controllers (2507ms)
- ✓ ete002 New controller can configure minter (2548ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4973ms)
- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (5084ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (5176ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2770ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2496ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (2859ms)
- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (2670ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (2489ms)
- ✓ ete011 New controller can increment minter allowance (2235ms)
- ✓ ete012 New controller can remove minter (2335ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (5887ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (3082ms)

- ✓ ete015 Configure a minter and make sure it can mint (2602ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2701ms)
- ✓ ete017 Configure a minter and make sure it can burn (2283ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2249ms)

Contract: FiatTokenV1: MINTp0_EndToEndTests MasterMinter

- ✓ ete000 New owner can configure controllers (2101ms)
- ✓ ete001 New owner can remove controllers (2055ms)
- ✓ ete002 New controller can configure minter (2157ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4185ms)
- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (4448ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (5160ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2289ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2240ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (2212ms)
- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (2324ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (2419ms)
- ✓ ete011 New controller can increment minter allowance (2132ms)
- ✓ ete012 New controller can remove minter (1988ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (4384ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (2364ms)
- ✓ ete015 Configure a minter and make sure it can mint (2336ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2194ms)
- ✓ ete017 Configure a minter and make sure it can burn (2359ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2131ms)

Contract: FiatTokenV1_1: MINTp0_EndToEndTests MasterMinter

- ✓ ete000 New owner can configure controllers (2012ms)
- ✓ ete001 New owner can remove controllers (2067ms)
- ✓ ete002 New controller can configure minter (1959ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4010ms)
- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (4117ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (4103ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2187ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2243ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (2503ms)

- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (2406ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (2272ms)
- ✓ ete011 New controller can increment minter allowance (1992ms)
- ✓ ete012 New controller can remove minter (2191ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (4267ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (2185ms)
- ✓ ete015 Configure a minter and make sure it can mint (2104ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2642ms)
- ✓ ete017 Configure a minter and make sure it can burn (2721ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2687ms)

Contract: FiatTokenV2: MINTp0_EndToEndTests MasterMinter

- ✓ ete000 New owner can configure controllers (2079ms)
- ✓ ete001 New owner can remove controllers (2076ms)
- ✓ ete002 New controller can configure minter (2105ms)
- ✓ ete003 Configure two controllers for the same minter and make sure they can both configureMinter (4220ms)
- ✓ ete004 Configure two controllers for the same minter, one adds a minter and the other removes it (4346ms)
- ✓ ete005 Configure two controllers for different minters and make sure 2nd cant remove (4330ms)
- ✓ ete006 Configure two controllers for different minters and make sure 2nd cant configure (2156ms)
- ✓ ete007 Remove a controller and make sure it can't modify minter (2238ms)
- ✓ ete008 Change minter manager and make sure existing controllers still can configure their minters (2570ms)
- ✓ ete009 Change minter manager and make sure existing controllers still can remove their minters (2725ms)
- ✓ ete010 Change minter manager and make sure existing controllers can increment allowances (2738ms)
- ✓ ete011 New controller can increment minter allowance (2402ms)
- ✓ ete012 New controller can remove minter (2311ms)
- ✓ ete013 Change minter manager, configure a minter, then change back and make sure changes DID NOT propogate (4472ms)
- ✓ ete014 Remove a minter and then try to increment its allowance reverts (2362ms)
- ✓ ete015 Configure a minter and make sure it can mint (2305ms)
- ✓ ete016 Remove a minter and make sure it cannot mint (2607ms)
- ✓ ete017 Configure a minter and make sure it can burn (2237ms)
- ✓ ete018 Remove a minter and make sure it cannot burn (2414ms)

Contract: FiatTokenV1: MINTp0_EventTests MintController

- ✓ et100 transferOwnership emits OwnershipTransferred event (59ms)
- ✓ et101 configureController emits ControllerConfigured event (67ms)
- ✓ et102 removeController emits ControllerRemoved event (85ms)

- ✓ et103 setMinterManager emits MinterManagerSet event (61ms)
- ✓ et104 removeMinter emits MinterRemoved event (198ms)
- ✓ et105 configureMinter emits MinterConfigured event (102ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (170ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (166ms)

Contract: FiatTokenV1_1: MINTp0_EventTests MintController

- ✓ et100 transferOwnership emits OwnershipTransferred event (56ms)
- ✓ et101 configureController emits ControllerConfigured event (52ms)
- ✓ et102 removeController emits ControllerRemoved event (89ms)
- ✓ et103 setMinterManager emits MinterManagerSet event (55ms)
- ✓ et104 removeMinter emits MinterRemoved event (158ms)
- ✓ et105 configureMinter emits MinterConfigured event (101ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (163ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (159ms)

Contract: FiatTokenV2: MINTp0_EventTests MintController

- ✓ et100 transferOwnership emits OwnershipTransferred event (59ms)
- ✓ et101 configureController emits ControllerConfigured event (49ms)
- ✓ et102 removeController emits ControllerRemoved event (85ms)
- ✓ et103 setMinterManager emits MinterManagerSet event (56ms)
- ✓ et104 removeMinter emits MinterRemoved event (138ms)
- ✓ et105 configureMinter emits MinterConfigured event (102ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (163ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (175ms)

Contract: FiatTokenV1: MINTp0_EventTests MasterMinter

- ✓ et100 transferOwnership emits OwnershipTransferred event (42ms)
- ✓ et101 configureController emits ControllerConfigured event (55ms)
- ✓ et102 removeController emits ControllerRemoved event (91ms)
- ✓ et103 setMinterManager emits MinterManagerSet event (57ms)
- ✓ et104 removeMinter emits MinterRemoved event (174ms)
- ✓ et105 configureMinter emits MinterConfigured event (90ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (152ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (141ms)

Contract: FiatTokenV1_1: MINTp0_EventTests MasterMinter

- ✓ et100 transferOwnership emits OwnershipTransferred event (43ms)
- ✓ et101 configureController emits ControllerConfigured event (52ms)
- ✓ et102 removeController emits ControllerRemoved event (88ms)
- ✓ et103 setMinterManager emits MinterManagerSet event (54ms)
- ✓ et104 removeMinter emits MinterRemoved event (146ms)
- ✓ et105 configureMinter emits MinterConfigured event (98ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (151ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (139ms)

Contract: FiatTokenV2: MINTp0_EventTests MasterMinter

- ✓ et100 transferOwnership emits OwnershipTransferred event (48ms)

- ✓ et101 configureController emits ControllerConfigured event (45ms)
- ✓ et102 removeController emits ControllerRemoved event (89ms)
- ✓ et103 setMinterManager emits MinterManagerSet event (42ms)
- ✓ et104 removeMinter emits MinterRemoved event (138ms)
- ✓ et105 configureMinter emits MinterConfigured event (90ms)
- ✓ et106 incrementMinterAllowance emits MinterAllowanceIncremented event (223ms)
- ✓ et107 decrementMinterAllowance emits MinterAllowanceDecremented event (148ms)

Contract: public to external

changing access modifier from public to external

- ✓ does not affect existing contracts' ability to call (141ms)
- ✓ does not affect existing contracts' ability to call via a proxy (295ms)

Contract: ECRecover

recover

- ✓ recovers signer address from a valid signature (91ms)
- ✓ reverts if an invalid signature is given (392ms)
- ✓ reverts if an invalid v value is given (137ms)
- ✓ reverts if an high-s value is given

Contract: EIP712

makeDomainSeparator

- ✓ generates a EIP712 domain separator (102ms)

recover

- ✓ recovers the signer's address from signed data (144ms)

Contract: FiatTokenV1_1

behaves like a Rescuable

updateRescuer

- ✓ allows owner to change rescuer (71ms)
- ✓ does not allow non-owner to change rescuer (55ms)
- ✓ does not allow updating the rescuer to the zero address (49ms)

rescueERC20

- ✓ allows rescuer to rescue ERC20 (full amount) (91ms)
- ✓ allows rescuer to rescue ERC20 (partial amount) (109ms)
- ✓ reverts when the requested amount is greater than balance (62ms)
- ✓ reverts when the the given contract address is not ERC20 (58ms)
- ✓ does not allow non-rescuer to rescue ERC20 (105ms)

uses original storage slot positions

- ✓ retains original storage slots 0 through 13 (72ms)
- ✓ retains slot 14 for rescuer
- ✓ retains original storage slots for blacklisted mapping
- ✓ retains original storage slots for balances mapping
- ✓ retains original storage slots for allowed mapping
- ✓ retains original storage slots for minters mapping
- ✓ retains original storage slots for minterAllowed mapping

Contract: Rescuable

✓ initially sets rescuer to be the zero address (203ms)

behaves like a Rescuable

updateRescuer

✓ allows owner to change rescuer (78ms)

✓ does not allow non-owner to change rescuer (50ms)

✓ does not allow updating the rescuer to the zero address (45ms)

rescueERC20

✓ allows rescuer to rescue ERC20 (full amount) (116ms)

✓ allows rescuer to rescue ERC20 (partial amount) (110ms)

✓ reverts when the requested amount is greater than balance (61ms)

✓ reverts when the the given contract address is not ERC20 (89ms)

✓ does not allow non-rescuer to rescue ERC20 (94ms)

Contract: FiatTokenV1: FiatToken ABI hacking

✓ abi004 FiatToken pause() is public (2663ms)

✓ abi040 Blacklistable constructor is not a function

✓ abi042 Ownable constructor is not a function (141ms)

✓ abi005 Pausable constructor is not a function

✓ abi043 FiatTokenProxy constructor is not a function

✓ abi027 UpgradeabilityProxy constructor

✓ abi055 Proxy constructor is not a function

✓ abi056 Proxy _delegate is internal

✓ abi057 Proxy _willFallback is internal (56ms)

✓ abi058 Proxy _fallback is internal

✓ abi050 Upgradeability implementation is internal

✓ abi051 AdminUpgradeabillityProxy constructor is not a function

✓ abi053 AdminUpgradeabillityProxy _setAdmin is internal

✓ abi041 FiatToken constructor is not a function

✓ abi025 setOwner is internal

✓ abi028 UpgradeabilityProxy._upgradeTo is internal

Contract: FiatTokenV1_1: FiatToken ABI hacking

✓ abi004 FiatToken pause() is public (2079ms)

✓ abi040 Blacklistable constructor is not a function

✓ abi042 Ownable constructor is not a function

✓ abi005 Pausable constructor is not a function

✓ abi043 FiatTokenProxy constructor is not a function

✓ abi027 UpgradeabilityProxy constructor

✓ abi055 Proxy constructor is not a function

✓ abi056 Proxy _delegate is internal

✓ abi057 Proxy _willFallback is internal

✓ abi058 Proxy _fallback is internal

✓ abi050 Upgradeability implementation is internal

✓ abi051 AdminUpgradeabillityProxy constructor is not a function

✓ abi053 AdminUpgradeabillityProxy _setAdmin is internal (48ms)

- ✓ abi041 FiatToken constructor is not a function
- ✓ abi025 setOwner is internal
- ✓ abi028 UpgradeabilityProxy._upgradeTo is internal

Contract: FiatTokenV2: FiatToken ABI hacking

- ✓ et000 should check MinterConfigured event (44ms)
- ✓ et001 should check Mint/Transfer events (82ms)
- ✓ et002 should check Burn/Transfer events (123ms)
- ✓ et003 should check MinterRemoved event (78ms)
- ✓ et004 should check MasterMinterChanged event (45ms)
- ✓ et005 should check Blacklisted event
- ✓ et006 should check UnBlacklisted event
- ✓ et007 should check BlacklisterChanged event (41ms)
- ✓ et008 should check Upgraded event (110ms)
- ✓ et009 should check AdminChanged event (41ms)
- ✓ et010 should check OwnershipTransferred event
- ✓ et011 should check Approval event (39ms)
- ✓ et012 should check Pause event
- ✓ et013 should check Unpause event
- ✓ et014 should check PauserChanged event (42ms)
- ✓ et015 should check Transfer event (131ms)
- ✓ et016 should check Transfer event in transferFrom (197ms)

Contract: FiatTokenV1_1: FiatToken events

- ✓ et000 should check MinterConfigured event (39ms)
- ✓ et001 should check Mint/Transfer events (93ms)
- ✓ et002 should check Burn/Transfer events (149ms)
- ✓ et003 should check MinterRemoved event (82ms)
- ✓ et004 should check MasterMinterChanged event
- ✓ et005 should check Blacklisted event
- ✓ et006 should check UnBlacklisted event
- ✓ et007 should check BlacklisterChanged event (43ms)
- ✓ et008 should check Upgraded event (139ms)
- ✓ et009 should check AdminChanged event
- ✓ et010 should check OwnershipTransferred event
- ✓ et011 should check Approval event (40ms)
- ✓ et012 should check Pause event (39ms)
- ✓ et013 should check Unpause event
- ✓ et014 should check PauserChanged event (45ms)
- ✓ et015 should check Transfer event (119ms)
- ✓ et016 should check Transfer event in transferFrom (169ms)

Contract: FiatTokenV2: FiatToken events

- ✓ et000 should check MinterConfigured event (46ms)
- ✓ et001 should check Mint/Transfer events (105ms)
- ✓ et002 should check Burn/Transfer events (149ms)

- ✓ et003 should check MinterRemoved event (94ms)
- ✓ et004 should check MasterMinterChanged event
- ✓ et005 should check Blacklisted event (42ms)
- ✓ et006 should check UnBlacklisted event
- ✓ et007 should check BlacklisterChanged event (50ms)
- ✓ et008 should check Upgraded event (130ms)
- ✓ et009 should check AdminChanged event
- ✓ et010 should check OwnershipTransferred event (42ms)
- ✓ et011 should check Approval event (38ms)
- ✓ et012 should check Pause event (41ms)
- ✓ et013 should check Unpause event
- ✓ et014 should check PauserChanged event
- ✓ et015 should check Transfer event (116ms)
- ✓ et016 should check Transfer event in transferFrom (188ms)

Contract: FiatTokenV1: FiatToken extended positive

- ✓ ept001 should changeAdmin while paused (2045ms)
- ✓ ept002 should updateMasterMinter while paused (2094ms)
- ✓ ept003 should updateBlacklister while paused (2033ms)
- ✓ ept004 should updatePauser while paused (2016ms)
- ✓ ept005 should transferOwnership while paused (2154ms)
- ✓ ept006 should removeMinter while paused (4230ms)
- ✓ ept008 should upgrade while paused (2424ms)
- ✓ ept013 should changeAdmin when msg.sender blacklisted (2353ms)
- ✓ ept014 should updateMasterMinter when msg.sender blacklisted (2257ms)
- ✓ ept015 should updateBlacklister when msg.sender blacklisted (2125ms)
- ✓ ept016 should updatePauser when msg.sender blacklisted (2097ms)
- ✓ ept017 should transferOwnership when msg.sender blacklisted (2148ms)
- ✓ ept018 should pause when msg.sender blacklisted (2088ms)
- ✓ ept019 should unpause when msg.sender blacklisted (4432ms)
- ✓ ept020 should blacklist when msg.sender blacklisted (2660ms)
- ✓ ept021 should unBlacklist when msg.sender blacklisted (4126ms)
- ✓ ept022 should upgrade when msg.sender blacklisted (2091ms)
- ✓ ept023 should upgrade to blacklisted address (2315ms)
- ✓ ept024 should blacklist a blacklisted address (2635ms)
- ✓ ept025 should changeAdmin to blacklisted address (2068ms)
- ✓ ept026 should updateMasterMinter to blacklisted address (2109ms)
- ✓ ept027 should updateBlacklister to blacklisted address (1998ms)
- ✓ ept028 should updatePauser to blacklisted address (2115ms)
- ✓ ept029 should transferOwnership to blacklisted address (2171ms)
- ✓ ept030 should configureMinter when masterMinter is blacklisted (2052ms)
- ✓ ept032 should configureMinter when minter is blacklisted (2169ms)
- ✓ ept033 should removeMinter when masterMinter is blacklisted (4316ms)
- ✓ ept034 should removeMinter when minter is blacklisted (4411ms)

- ✓ ept035 should unBlacklist while contract is paused (4645ms)
- ✓ ept036 should blacklist while contract is paused (4238ms)
- ✓ ept037 should pause while contract is paused (4174ms)

Contract: FiatTokenV1_1: FiatToken extended positive

- ✓ ept001 should changeAdmin while paused (2054ms)
- ✓ ept002 should updateMasterMinter while paused (2115ms)
- ✓ ept003 should updateBlacklister while paused (1895ms)
- ✓ ept004 should updatePauser while paused (2080ms)
- ✓ ept005 should transferOwnership while paused (2033ms)
- ✓ ept006 should removeMinter while paused (4272ms)
- ✓ ept008 should upgrade while paused (2042ms)
- ✓ ept013 should changeAdmin when msg.sender blacklisted (2004ms)
- ✓ ept014 should updateMasterMinter when msg.sender blacklisted (2307ms)
- ✓ ept015 should updateBlacklister when msg.sender blacklisted (2061ms)
- ✓ ept016 should updatePauser when msg.sender blacklisted (2112ms)
- ✓ ept017 should transferOwnership when msg.sender blacklisted (2340ms)
- ✓ ept018 should pause when msg.sender blacklisted (2189ms)
- ✓ ept019 should unpause when msg.sender blacklisted (4073ms)
- ✓ ept020 should blacklist when msg.sender blacklisted (2281ms)
- ✓ ept021 should unBlacklist when msg.sender blacklisted (4242ms)
- ✓ ept022 should upgrade when msg.sender blacklisted (2309ms)
- ✓ ept023 should upgrade to blacklisted address (2367ms)
- ✓ ept024 should blacklist a blacklisted address (2372ms)
- ✓ ept025 should changeAdmin to blacklisted address (2851ms)
- ✓ ept026 should updateMasterMinter to blacklisted address (2317ms)
- ✓ ept027 should updateBlacklister to blacklisted address (2305ms)
- ✓ ept028 should updatePauser to blacklisted address (2356ms)
- ✓ ept029 should transferOwnership to blacklisted address (2542ms)
- ✓ ept030 should configureMinter when masterMinter is blacklisted (2436ms)
- ✓ ept032 should configureMinter when minter is blacklisted (2448ms)
- ✓ ept033 should removeMinter when masterMinter is blacklisted (4604ms)
- ✓ ept034 should removeMinter when minter is blacklisted (4280ms)
- ✓ ept035 should unBlacklist while contract is paused (4230ms)
- ✓ ept036 should blacklist while contract is paused (4419ms)
- ✓ ept037 should pause while contract is paused (4320ms)

Contract: FiatTokenV2: FiatToken extended positive

- ✓ ept001 should changeAdmin while paused (2064ms)
- ✓ ept002 should updateMasterMinter while paused (1981ms)
- ✓ ept003 should updateBlacklister while paused (2226ms)
- ✓ ept004 should updatePauser while paused (2170ms)
- ✓ ept005 should transferOwnership while paused (2089ms)
- ✓ ept006 should removeMinter while paused (4350ms)
- ✓ ept008 should upgrade while paused (2264ms)

- ✓ ept013 should changeAdmin when msg.sender blacklisted (2188ms)
- ✓ ept014 should updateMasterMinter when msg.sender blacklisted (2288ms)
- ✓ ept015 should updateBlacklister when msg.sender blacklisted (2078ms)
- ✓ ept016 should updatePauser when msg.sender blacklisted (2083ms)
- ✓ ept017 should transferOwnership when msg.sender blacklisted (2390ms)
- ✓ ept018 should pause when msg.sender blacklisted (2131ms)
- ✓ ept019 should unpause when msg.sender blacklisted (4260ms)
- ✓ ept020 should blacklist when msg.sender blacklisted (2228ms)
- ✓ ept021 should unBlacklist when msg.sender blacklisted (4175ms)
- ✓ ept022 should upgrade when msg.sender blacklisted (2121ms)
- ✓ ept023 should upgrade to blacklisted address (2208ms)
- ✓ ept024 should blacklist a blacklisted address (2141ms)
- ✓ ept025 should changeAdmin to blacklisted address (2056ms)
- ✓ ept026 should updateMasterMinter to blacklisted address (2011ms)
- ✓ ept027 should updateBlacklister to blacklisted address (2154ms)
- ✓ ept028 should updatePauser to blacklisted address (2111ms)
- ✓ ept029 should transferOwnership to blacklisted address (2177ms)
- ✓ ept030 should configureMinter when masterMinter is blacklisted (2145ms)
- ✓ ept032 should configureMinter when minter is blacklisted (2036ms)
- ✓ ept033 should removeMinter when masterMinter is blacklisted (4053ms)
- ✓ ept034 should removeMinter when minter is blacklisted (4446ms)
- ✓ ept035 should unBlacklist while contract is paused (4699ms)
- ✓ ept036 should blacklist while contract is paused (4285ms)
- ✓ ept037 should pause while contract is paused (4394ms)

Contract: FiatTokenV1

uses original storage slot positions

- ✓ retains original storage slots 0 through 13 (65ms)
- ✓ retains original storage slots for blacklisted mapping
- ✓ retains original storage slots for balances mapping
- ✓ retains original storage slots for allowed mapping
- ✓ retains original storage slots for minters mapping
- ✓ retains original storage slots for minterAllowed mapping

Contract: FiatTokenV1: legacy

- ✓ should start with a totalSupply of 0
- ✓ should add multiple mints to a given address in address balance (428ms)
- ✓ should fail to mint to a null address (179ms)
- ✓ should add multiple mints to a given address in address balance (393ms)
- ✓ should add multiple mints to total supply (621ms)
- ✓ should fail to mint from blacklisted minter (173ms)
- ✓ should fail to mint to blacklisted address (220ms)
- ✓ should fail to mint from a non-minter call (257ms)
- ✓ should complete transferFrom (362ms)
- ✓ should approve (56ms)

- ✓ should complete sample transfer (412ms)
- ✓ should complete transfer from non-owner (306ms)
- ✓ should set allowance and balances before and after approved transfer (386ms)
- ✓ should fail on unauthorized approved transfer and not change balances (417ms)
- ✓ should fail on invalid approved transfer amount and not change balances (396ms)
- ✓ should fail on invalid transfer recipient (zero-account) and not change balances (360ms)
- ✓ should test consistency of transfer(x) and approve(x) + transferFrom(x) (630ms)
- ✓ should pause and should not be able to transfer (371ms)
- ✓ should pause and should not be able to transfer, then unpause and be able to transfer (1000ms)
- ✓ should pause and should not be able to transferFrom (336ms)
- ✓ should pause and should not be able to approve (330ms)
- ✓ should pause and should not be able to mint (348ms)
- ✓ should try to pause with non-pauser and fail to pause (264ms)
- ✓ should try to pause with non-pauser and fail to pause (265ms)
- ✓ should approve and fail to transfer more than balance (288ms)
- ✓ should blacklist and make transfer impossible (304ms)
- ✓ should blacklist recipient and make transfer to recipient impossible (486ms)
- ✓ should blacklist and make transferFrom impossible with the approved transferer (379ms)
- ✓ should make transferFrom impossible with the approved and blacklisted transferer (407ms)
- ✓ should blacklist recipient and make transfer to recipient using transferFrom impossible (366ms)
- ✓ should blacklist and make approve impossible (324ms)
- ✓ should make giving approval to blacklisted account impossible (298ms)
- ✓ should blacklist then unblacklist to make a transfer possible (379ms)
- ✓ should fail to blacklist with non-blacklister account (463ms)
- ✓ should unblacklist when paused (391ms)
- ✓ should change the minter and mint as well as fail to mint with the old minter (252ms)
- ✓ should remove a minter even if the contract is paused (176ms)
- ✓ should pause contract even when contract is already paused (132ms)
- ✓ should unpause contract even when contract is already unpaused (58ms)
- ✓ should fail to updateMinterAllowance from non-masterMinter (89ms)
- ✓ should have correct name
- ✓ should have correct symbol
- ✓ should have correct decimals
- ✓ should have correct currency
- ✓ should mint and burn tokens (434ms)
- ✓ should try to burn tokens from a non-minter and fail (44ms)
- ✓ should fail to burn from a blacklisted address (367ms)
- ✓ should try to burn more tokens than balance and fail (320ms)
- ✓ should upgrade and preserve data (548ms)
- ✓ should updateRoleAddress for masterMinter (150ms)
- ✓ should updateRoleAddress for blacklister (113ms)
- ✓ should updateRoleAddress for pauser (119ms)
- ✓ should updateUpgraderAddress for upgrader (415ms)

- ✓ should fail to updateUpgraderAddress for upgrader using non-upgrader account (49ms)
- ✓ should updateRoleAddress for roleAddressChanger (127ms)
- ✓ should fail to updateRoleAddress from a non-roleAddressChanger (51ms)

Contract: FiatTokenV1_1: legacy

- ✓ should start with a totalSupply of 0
- ✓ should add multiple mints to a given address in address balance (416ms)
- ✓ should fail to mint to a null address (193ms)
- ✓ should add multiple mints to a given address in address balance (417ms)
- ✓ should add multiple mints to total supply (720ms)
- ✓ should fail to mint from blacklisted minter (195ms)
- ✓ should fail to mint to blacklisted address (212ms)
- ✓ should fail to mint from a non-minter call (269ms)
- ✓ should complete transferFrom (375ms)
- ✓ should approve (66ms)
- ✓ should complete sample transfer (374ms)
- ✓ should complete transfer from non-owner (295ms)
- ✓ should set allowance and balances before and after approved transfer (325ms)
- ✓ should fail on unauthorized approved transfer and not change balances (510ms)
- ✓ should fail on invalid approved transfer amount and not change balances (368ms)
- ✓ should fail on invalid transfer recipient (zero-account) and not change balances (305ms)
- ✓ should test consistency of transfer(x) and approve(x) + transferFrom(x) (667ms)
- ✓ should pause and should not be able to transfer (337ms)
- ✓ should pause and should not be able to transfer, then unpause and be able to transfer (841ms)
- ✓ should pause and should not be able to transferFrom (361ms)
- ✓ should pause and should not be able to approve (337ms)
- ✓ should pause and should not be able to mint (498ms)
- ✓ should try to pause with non-pauser and fail to pause (308ms)
- ✓ should try to pause with non-pauser and fail to pause (277ms)
- ✓ should approve and fail to transfer more than balance (348ms)
- ✓ should blacklist and make transfer impossible (331ms)
- ✓ should blacklist recipient and make transfer to recipient impossible (565ms)
- ✓ should blacklist and make transferFrom impossible with the approved transferer (402ms)
- ✓ should make transferFrom impossible with the approved and blacklisted transferer (459ms)
- ✓ should blacklist recipient and make transfer to recipient using transferFrom impossible (386ms)
- ✓ should blacklist and make approve impossible (354ms)
- ✓ should make giving approval to blacklisted account impossible (288ms)
- ✓ should blacklist then unblacklist to make a transfer possible (560ms)
- ✓ should fail to blacklist with non-blacklister account (348ms)
- ✓ should unblacklist when paused (404ms)
- ✓ should change the minter and mint as well as fail to mint with the old minter (297ms)
- ✓ should remove a minter even if the contract is paused (148ms)
- ✓ should pause contract even when contract is already paused (93ms)
- ✓ should unpause contract even when contract is already unpaused (76ms)

- ✓ should fail to updateMinterAllowance from non-masterMinter (82ms)
- ✓ should have correct name
- ✓ should have correct symbol
- ✓ should have correct decimals
- ✓ should have correct currency
- ✓ should mint and burn tokens (439ms)
- ✓ should try to burn tokens from a non-minter and fail (74ms)
- ✓ should fail to burn from a blacklisted address (464ms)
- ✓ should try to burn more tokens than balance and fail (456ms)
- ✓ should upgrade and preserve data (533ms)
- ✓ should updateRoleAddress for masterMinter (139ms)
- ✓ should updateRoleAddress for blacklister (142ms)
- ✓ should updateRoleAddress for pauser (143ms)
- ✓ should updateUpgraderAddress for upgrader (437ms)
- ✓ should fail to updateUpgraderAddress for upgrader using non-upgrader account (51ms)
- ✓ should updateRoleAddress for roleAddressChanger (119ms)
- ✓ should fail to updateRoleAddress from a non-roleAddressChanger (41ms)

Contract: FiatTokenV2: legacy

- ✓ should start with a totalSupply of 0
- ✓ should add multiple mints to a given address in address balance (422ms)
- ✓ should fail to mint to a null address (188ms)
- ✓ should add multiple mints to a given address in address balance (441ms)
- ✓ should add multiple mints to total supply (658ms)
- ✓ should fail to mint from blacklisted minter (173ms)
- ✓ should fail to mint to blacklisted address (197ms)
- ✓ should fail to mint from a non-minter call (276ms)
- ✓ should complete transferFrom (362ms)
- ✓ should approve (73ms)
- ✓ should complete sample transfer (367ms)
- ✓ should complete transfer from non-owner (302ms)
- ✓ should set allowance and balances before and after approved transfer (358ms)
- ✓ should fail on unauthorized approved transfer and not change balances (397ms)
- ✓ should fail on invalid approved transfer amount and not change balances (375ms)
- ✓ should fail on invalid transfer recipient (zero-account) and not change balances (323ms)
- ✓ should test consistency of transfer(x) and approve(x) + transferFrom(x) (647ms)
- ✓ should pause and should not be able to transfer (371ms)
- ✓ should pause and should not be able to transfer, then unpause and be able to transfer (892ms)
- ✓ should pause and should not be able to transferFrom (384ms)
- ✓ should pause and should not be able to approve (379ms)
- ✓ should pause and should not be able to mint (329ms)
- ✓ should try to pause with non-pauser and fail to pause (287ms)
- ✓ should try to pause with non-pauser and fail to pause (284ms)
- ✓ should approve and fail to transfer more than balance (321ms)

- ✓ should blacklist and make transfer impossible (287ms)
- ✓ should blacklist recipient and make transfer to recipient impossible (529ms)
- ✓ should blacklist and make transferFrom impossible with the approved transferer (547ms)
- ✓ should make transferFrom impossible with the approved and blacklisted transferer (345ms)
- ✓ should blacklist recipient and make transfer to recipient using transferFrom impossible (367ms)
- ✓ should blacklist and make approve impossible (308ms)
- ✓ should make giving approval to blacklisted account impossible (315ms)
- ✓ should blacklist then unblacklist to make a transfer possible (383ms)
- ✓ should fail to blacklist with non-blacklister account (311ms)
- ✓ should unblacklist when paused (383ms)
- ✓ should change the minter and mint as well as fail to mint with the old minter (251ms)
- ✓ should remove a minter even if the contract is paused (185ms)
- ✓ should pause contract even when contract is already paused (110ms)
- ✓ should unpause contract even when contract is already unpaused (59ms)
- ✓ should fail to updateMinterAllowance from non-masterMinter (102ms)
- ✓ should have correct name
- ✓ should have correct symbol
- ✓ should have correct decimals
- ✓ should have correct currency
- ✓ should mint and burn tokens (453ms)
- ✓ should try to burn tokens from a non-minter and fail (46ms)
- ✓ should fail to burn from a blacklisted address (484ms)
- ✓ should try to burn more tokens than balance and fail (333ms)
- ✓ should upgrade and preserve data (510ms)
- ✓ should updateRoleAddress for masterMinter (205ms)
- ✓ should updateRoleAddress for blacklister (141ms)
- ✓ should updateRoleAddress for pauser (123ms)
- ✓ should updateUpgraderAddress for upgrader (425ms)
- ✓ should fail to updateUpgraderAddress for upgrader using non-upgrader account (49ms)
- ✓ should updateRoleAddress for roleAddressChanger (122ms)
- ✓ should fail to updateRoleAddress from a non-roleAddressChanger (41ms)

Contract: FiatTokenV1: FiatToken misc

- ✓ ms001 no payable function
- ✓ ms002 should transfer to self has correct final balance (2228ms)
- ✓ ms003 should transferFrom to self from approved account and have correct final balance (2257ms)
- ✓ ms004 should transferFrom to self from approved self and have correct final balance (2058ms)
- ✓ ms005 should mint to self with correct final balance (4280ms)
- ✓ ms006 should approve correct allowance for self (4566ms)
- ✓ ms007 should configureMinter for masterMinter (2322ms)
- ✓ ms009 should configure two minters (2050ms)
- ✓ ms010 should configure two minters and each mint distinct amounts (2445ms)
- ✓ ms011 should configure two minters, each minting distinct amounts and then remove one minter (2224ms)

- ✓ ms012 should configure two minters and adjust both allowances (4210ms)
- ✓ ms013 should configure two minters, one with zero allowance fails to mint (2104ms)
- ✓ ms014 should configure two minters and fail to mint when paused (2144ms)
- ✓ ms015 should configure two minters, blacklist one and ensure it cannot mint, then unblacklist and ensure it can mint (4176ms)
- ✓ ms016 should configure two minters, each mints to themselves and then burns certain amount (4560ms)
- ✓ ms018 should approve 0 token allowance with unchanged state (2092ms)
- ✓ ms019 should transferFrom 0 tokens with unchanged state (2136ms)
- ✓ ms020 should transfer 0 tokens with unchanged state (2100ms)
- ✓ ms036 should get allowance for same address (70ms)
- ✓ ms039 should return true on mint (2015ms)
- ✓ ms040 should return true on approve
- ✓ ms041 should return true on transferFrom (4408ms)
- ✓ ms042 should return true on transfer (4134ms)
- ✓ ms043 should return true on configureMinter
- ✓ ms044 should return true on removeMinter
- ✓ ms045 initialized should be in slot 8, byte 21
- ✓ ms046 initialized should be 0 before initialization (225ms)
- ✓ ms047 configureMinter works on amount=2^256-1 (1963ms)
- ✓ ms048 mint works on amount=2^256-1 (4047ms)
- ✓ ms049 burn on works on amount=2^256-1 (4302ms)
- ✓ ms050 approve works on amount=2^256-1 (4449ms)
- ✓ ms051 transfer works on amount=2^256-1 (5169ms)
- ✓ ms052 transferFrom works on amount=2^256-1 (5023ms)

Contract: FiatTokenV1_1: FiatToken misc

- ✓ ms001 no payable function
- ✓ ms002 should transfer to self has correct final balance (2338ms)
- ✓ ms003 should transferFrom to self from approved account and have correct final balance (2529ms)
- ✓ ms004 should transferFrom to self from approved self and have correct final balance (2333ms)
- ✓ ms005 should mint to self with correct final balance (4543ms)
- ✓ ms006 should approve correct allowance for self (4283ms)
- ✓ ms007 should configureMinter for masterMinter (2078ms)
- ✓ ms009 should configure two minters (2161ms)
- ✓ ms010 should configure two minters and each mint distinct amounts (2147ms)
- ✓ ms011 should configure two minters, each minting distinct amounts and then remove one minter (2239ms)
- ✓ ms012 should configure two minters and adjust both allowances (4176ms)
- ✓ ms013 should configure two minters, one with zero allowance fails to mint (2384ms)
- ✓ ms014 should configure two minters and fail to mint when paused (2218ms)
- ✓ ms015 should configure two minters, blacklist one and ensure it cannot mint, then unblacklist and ensure it can mint (4557ms)

- ✓ ms016 should configure two minters, each mints to themselves and then burns certain amount (4535ms)
- ✓ ms018 should approve 0 token allowance with unchanged state (2230ms)
- ✓ ms019 should transferFrom 0 tokens with unchanged state (2232ms)
- ✓ ms020 should transfer 0 tokens with unchanged state (2243ms)
- ✓ ms036 should get allowance for same address (83ms)
- ✓ ms039 should return true on mint (2162ms)
- ✓ ms040 should return true on approve
- ✓ ms041 should return true on transferFrom (4619ms)
- ✓ ms042 should return true on transfer (4218ms)
- ✓ ms043 should return true on configureMinter
- ✓ ms044 should return true on removeMinter
- ✓ ms045 initialized should be in slot 8, byte 21
- ✓ ms046 initialized should be 0 before initialization (218ms)
- ✓ ms047 configureMinter works on amount=2^256-1 (2124ms)
- ✓ ms048 mint works on amount=2^256-1 (4333ms)
- ✓ ms049 burn on works on amount=2^256-1 (4342ms)
- ✓ ms050 approve works on amount=2^256-1 (4400ms)
- ✓ ms051 transfer works on amount=2^256-1 (4356ms)
- ✓ ms052 transferFrom works on amount=2^256-1 (4505ms)

Contract: FiatTokenV2: FiatToken misc

- ✓ ms001 no payable function
- ✓ ms002 should transfer to self has correct final balance (2085ms)
- ✓ ms003 should transferFrom to self from approved account and have correct final balance (2153ms)
- ✓ ms004 should transferFrom to self from approved self and have correct final balance (2212ms)
- ✓ ms005 should mint to self with correct final balance (4138ms)
- ✓ ms006 should approve correct allowance for self (4509ms)
- ✓ ms007 should configureMinter for masterMinter (2605ms)
- ✓ ms009 should configure two minters (2409ms)
- ✓ ms010 should configure two minters and each mint distinct amounts (2478ms)
- ✓ ms011 should configure two minters, each minting distinct amounts and then remove one minter (2422ms)
- ✓ ms012 should configure two minters and adjust both allowances (4667ms)
- ✓ ms013 should configure two minters, one with zero allowance fails to mint (2379ms)
- ✓ ms014 should configure two minters and fail to mint when paused (2513ms)
- ✓ ms015 should configure two minters, blacklist one and ensure it cannot mint, then unblacklist and ensure it can mint (4751ms)
- ✓ ms016 should configure two minters, each mints to themselves and then burns certain amount (4833ms)
- ✓ ms018 should approve 0 token allowance with unchanged state (2256ms)
- ✓ ms019 should transferFrom 0 tokens with unchanged state (2322ms)
- ✓ ms020 should transfer 0 tokens with unchanged state (2181ms)
- ✓ ms036 should get allowance for same address (68ms)

- ✓ ms039 should return true on mint (2362ms)
- ✓ ms040 should return true on approve
- ✓ ms041 should return true on transferFrom (4846ms)
- ✓ ms042 should return true on transfer (4815ms)
- ✓ ms043 should return true on configureMinter
- ✓ ms044 should return true on removeMinter
- ✓ ms045 initialized should be in slot 8, byte 21
- ✓ ms046 initialized should be 0 before initialization (282ms)
- ✓ ms047 configureMinter works on amount=2^256-1 (2420ms)
- ✓ ms048 mint works on amount=2^256-1 (5381ms)
- ✓ ms049 burn on works on amount=2^256-1 (4871ms)
- ✓ ms050 approve works on amount=2^256-1 (4637ms)
- ✓ ms051 transfer works on amount=2^256-1 (4607ms)
- ✓ ms052 transferFrom works on amount=2^256-1 (4729ms)

Contract: FiatTokenV1: FiatToken negative

- ✓ nt001 should fail to mint when paused (2301ms)
- ✓ nt002 should fail to mint when msg.sender is not a minter (1980ms)
- ✓ nt003 should fail to mint when msg.sender is blacklisted (2369ms)
- ✓ nt004 should fail to mint when recipient is blacklisted (2200ms)
- ✓ nt005 should fail to mint when allowance of minter is less than amount (2164ms)
- ✓ nt006 should fail to mint to 0x0 address (2172ms)
- ✓ nt008 should fail to approve when spender is blacklisted (2071ms)
- ✓ nt009 should fail to approve when msg.sender is blacklisted (2144ms)
- ✓ nt010 should fail to approve when contract is paused (2291ms)
- ✓ nt012 should fail to transferFrom to 0x0 address (4751ms)
- ✓ nt013 should fail to transferFrom an amount greater than balance (4749ms)
- ✓ nt014 should fail to transferFrom to blacklisted recipient (4409ms)
- ✓ nt015 should fail to transferFrom from blacklisted msg.sender (4919ms)
- ✓ nt016 should fail to transferFrom when from is blacklisted (4591ms)
- ✓ nt017 should fail to transferFrom an amount greater than allowed for msg.sender (4350ms)
- ✓ nt018 should fail to transferFrom when paused (4505ms)
- ✓ nt020 should fail to transfer to 0x0 address (4257ms)
- ✓ nt021 should fail to transfer an amount greater than balance (4284ms)
- ✓ nt022 should fail to transfer to blacklisted recipient (4784ms)
- ✓ nt023 should fail to transfer when sender is blacklisted (4186ms)
- ✓ nt024 should fail to transfer when paused (4698ms)
- ✓ nt026 should fail to configureMinter when sender is not masterMinter (2184ms)
- ✓ nt028 should fail to configureMinter when paused (2006ms)
- ✓ nt029 should fail to removeMinter when sender is not masterMinter (2189ms)
- ✓ nt031 should fail to burn when balance is less than amount (2075ms)
- ✓ nt032 should fail to burn when amount is -1 (2197ms)
- ✓ nt033 should fail to burn when sender is blacklisted (4197ms)
- ✓ nt034 should fail to burn when paused (4453ms)

- ✓ nt035 should fail to burn when sender is not minter (4551ms)
- ✓ nt036 should fail to burn after removeMinter (6584ms)
- ✓ nt050 should fail to updatePauser when sender is not owner (2231ms)
- ✓ nt049 should fail to updateMasterMinter when sender is not owner (2238ms)
- ✓ nt048 should fail to updateBlacklister when sender is not owner (2162ms)
- ✓ nt040 should fail to pause when sender is not pauser (2125ms)
- ✓ nt041 should fail to unpause when sender is not pauser (2134ms)
- ✓ nt042 should fail to blacklist when sender is not blacklister (2326ms)
- ✓ nt043 should fail to unblacklist when sender is not blacklister (2243ms)
- ✓ nt054 should fail to transferOwnership when sender is not owner (2078ms)
- ✓ nt055 should fail to mint when amount = 0 (2250ms)
- ✓ nt056 should fail to burn when amount = 0 (2335ms)
- ✓ nt064 transferOwnership should fail on 0x0 (51ms)
- ✓ nt057 updateMasterMinter should fail on 0x0 (49ms)
- ✓ nt058 updatePauser should fail on 0x0 (44ms)
- ✓ nt059 updateBlacklister should fail on 0x0 (50ms)
- ✓ nt060 initialize should fail when _masterMinter is 0x0 (167ms)
- ✓ nt061 initialize should fail when _pauser is 0x0 (161ms)
- ✓ nt062 initialize should fail when _blacklister is 0x0 (186ms)
- ✓ nt063 initialize should fail when _owner is 0x0 (176ms)

Contract: FiatTokenV1_1: FiatToken negative

- ✓ nt001 should fail to mint when paused (2266ms)
- ✓ nt002 should fail to mint when msg.sender is not a minter (2002ms)
- ✓ nt003 should fail to mint when msg.sender is blacklisted (2069ms)
- ✓ nt004 should fail to mint when recipient is blacklisted (2028ms)
- ✓ nt005 should fail to mint when allowance of minter is less than amount (2057ms)
- ✓ nt006 should fail to mint to 0x0 address (2066ms)
- ✓ nt008 should fail to approve when spender is blacklisted (2062ms)
- ✓ nt009 should fail to approve when msg.sender is blacklisted (2272ms)
- ✓ nt010 should fail to approve when contract is paused (2227ms)
- ✓ nt012 should fail to transferFrom to 0x0 address (3964ms)
- ✓ nt013 should fail to transferFrom an amount greater than balance (4226ms)
- ✓ nt014 should fail to transferFrom to blacklisted recipient (4197ms)
- ✓ nt015 should fail to transferFrom from blacklisted msg.sender (4310ms)
- ✓ nt016 should fail to transferFrom when from is blacklisted (4096ms)
- ✓ nt017 should fail to transferFrom an amount greater than allowed for msg.sender (4009ms)
- ✓ nt018 should fail to transferFrom when paused (4390ms)
- ✓ nt020 should fail to transfer to 0x0 address (4512ms)
- ✓ nt021 should fail to transfer an amount greater than balance (4587ms)
- ✓ nt022 should fail to transfer to blacklisted recipient (4359ms)
- ✓ nt023 should fail to transfer when sender is blacklisted (4067ms)
- ✓ nt024 should fail to transfer when paused (4370ms)
- ✓ nt026 should fail to configureMinter when sender is not masterMinter (2138ms)

- ✓ nt028 should fail to configureMinter when paused (2238ms)
- ✓ nt029 should fail to removeMinter when sender is not masterMinter (2069ms)
- ✓ nt031 should fail to burn when balance is less than amount (2129ms)
- ✓ nt032 should fail to burn when amount is -1 (2086ms)
- ✓ nt033 should fail to burn when sender is blacklisted (4173ms)
- ✓ nt034 should fail to burn when paused (4053ms)
- ✓ nt035 should fail to burn when sender is not minter (4119ms)
- ✓ nt036 should fail to burn after removeMinter (6563ms)
- ✓ nt050 should fail to updatePauser when sender is not owner (2223ms)
- ✓ nt049 should fail to updateMasterMinter when sender is not owner (2289ms)
- ✓ nt048 should fail to updateBlacklister when sender is not owner (2260ms)
- ✓ nt040 should fail to pause when sender is not pauser (2319ms)
- ✓ nt041 should fail to unpause when sender is not pauser (2399ms)
- ✓ nt042 should fail to blacklist when sender is not blacklister (2471ms)
- ✓ nt043 should fail to unblacklist when sender is not blacklister (2263ms)
- ✓ nt054 should fail to transferOwnership when sender is not owner (2056ms)
- ✓ nt055 should fail to mint when amount = 0 (2400ms)
- ✓ nt056 should fail to burn when amount = 0 (2308ms)
- ✓ nt064 transferOwnership should fail on 0x0 (47ms)
- ✓ nt057 updateMasterMinter should fail on 0x0 (55ms)
- ✓ nt058 updatePauser should fail on 0x0 (42ms)
- ✓ nt059 updateBlacklister should fail on 0x0 (43ms)
- ✓ nt060 initialize should fail when _masterMinter is 0x0 (189ms)
- ✓ nt061 initialize should fail when _pauser is 0x0 (174ms)
- ✓ nt062 initialize should fail when _blacklister is 0x0 (176ms)
- ✓ nt063 initialize should fail when _owner is 0x0 (189ms)

Contract: FiatTokenV2: FiatToken negative

- ✓ nt001 should fail to mint when paused (2497ms)
- ✓ nt002 should fail to mint when msg.sender is not a minter (2398ms)
- ✓ nt003 should fail to mint when msg.sender is blacklisted (2344ms)
- ✓ nt004 should fail to mint when recipient is blacklisted (2195ms)
- ✓ nt005 should fail to mint when allowance of minter is less than amount (2223ms)
- ✓ nt006 should fail to mint to 0x0 address (2080ms)
- ✓ nt008 should fail to approve when spender is blacklisted (2305ms)
- ✓ nt009 should fail to approve when msg.sender is blacklisted (2040ms)
- ✓ nt010 should fail to approve when contract is paused (2217ms)
- ✓ nt012 should fail to transferFrom to 0x0 address (4097ms)
- ✓ nt013 should fail to transferFrom an amount greater than balance (4297ms)
- ✓ nt014 should fail to transferFrom to blacklisted recipient (4315ms)
- ✓ nt015 should fail to transferFrom from blacklisted msg.sender (4422ms)
- ✓ nt016 should fail to transferFrom when from is blacklisted (4787ms)
- ✓ nt017 should fail to transferFrom an amount greater than allowed for msg.sender (4648ms)
- ✓ t018 should fail to transferFrom when paused (4127ms)

- ✓ nt020 should fail to transfer to 0×0 address (4112ms)
- ✓ nt021 should fail to transfer an amount greater than balance (4414ms)
- ✓ nt022 should fail to transfer to blacklisted recipient (4416ms)
- ✓ nt023 should fail to transfer when sender is blacklisted (4267ms)
- ✓ nt024 should fail to transfer when paused (4596ms)
- ✓ nt026 should fail to configureMinter when sender is not masterMinter (2020ms)
- ✓ nt028 should fail to configureMinter when paused (2146ms)
- ✓ nt029 should fail to removeMinter when sender is not masterMinter (2112ms)
- ✓ nt031 should fail to burn when balance is less than amount (2102ms)
- ✓ nt032 should fail to burn when amount is -1 (2245ms)
- ✓ nt033 should fail to burn when sender is blacklisted (4125ms)
- ✓ nt034 should fail to burn when paused (4249ms)
- ✓ nt035 should fail to burn when sender is not minter (4204ms)
- ✓ nt036 should fail to burn after removeMinter (7269ms)
- ✓ nt050 should fail to updatePauser when sender is not owner (2367ms)
- ✓ nt049 should fail to updateMasterMinter when sender is not owner (2177ms)
- ✓ nt048 should fail to updateBlacklister when sender is not owner (2183ms)
- ✓ nt040 should fail to pause when sender is not pauser (2235ms)
- ✓ nt041 should fail to unpause when sender is not pauser (2232ms)
- ✓ nt042 should fail to blacklist when sender is not blacklister (2258ms)
- ✓ nt043 should fail to unblacklist when sender is not blacklister (2145ms)
- ✓ nt054 should fail to transferOwnership when sender is not owner (2122ms)
- ✓ nt055 should fail to mint when amount = 0 (2193ms)
- ✓ nt056 should fail to burn when amount = 0 (2107ms)
- ✓ nt064 transferOwnership should fail on 0×0 (45ms)
- ✓ nt057 updateMasterMinter should fail on 0×0 (43ms)
- ✓ nt058 updatePauser should fail on 0×0 (46ms)
- ✓ nt059 updateBlacklister should fail on 0×0 (42ms)
- ✓ nt060 initialize should fail when _masterMinter is 0×0 (188ms)
- ✓ nt061 initialize should fail when _pauser is 0×0 (176ms)
- ✓ nt062 initialize should fail when _blacklister is 0×0 (290ms)
- ✓ nt063 initialize should fail when _owner is 0×0 (172ms)

Contract: Pausable

- ✓ constructor owner
- ✓ constructor pauser
- ✓ paused after pausing (244ms)
- ✓ update pauser (298ms)
- ✓ fail to update pauser from wrong account (42ms)
- ✓ fail to pause from wrong account (42ms)
- ✓ fail to unpause from wrong account (146ms)

Contract: FiatTokenV1: FiatToken positive

- ✓ pt000 should check that default variable values are correct (2120ms)
- ✓ pt011 should pause and set paused to true (2074ms)

- ✓ pt006 should unpause and set paused to false (4027ms)
- ✓ pt020 should approve a spend and set allowed amount (2048ms)
- ✓ pt019 should blacklist and set blacklisted to true (2141ms)
- ✓ pt018 should blacklist and set blacklisted to true, then unblacklist and set blacklisted to false (4135ms)
- ✓ pt015 should configureMinter, setting the minter to true and mintingAllowance to amount (2129ms)
- ✓ pt012 should mint the amount, increasing balance of recipient by amount, increasing total supply by amount, and decreasing minterAllowed by amount (3994ms)
- ✓ pt017 should burn amount of tokens and reduce balance and total supply by amount (4528ms)
- ✓ pt010 should removeMinter, setting the minter to false and minterAllowed to 0 (4626ms)
- ✓ pt008 should transfer, reducing sender balance by amount and increasing recipient balance by amount (6130ms)
- ✓ pt007 should transferFrom, reducing sender balance by amount and increasing recipient balance by amount (6445ms)
- ✓ pt004 should updateMasterMinter (2105ms)
- ✓ pt005 should updateBlacklister (2124ms)
- ✓ pt003 should updatePauser (2128ms)
- ✓ pt009 should set owner to _newOwner (2116ms)

Contract: FiatTokenV1_1: FiatToken positive

- ✓ pt000 should check that default variable values are correct (1963ms)
- ✓ pt011 should pause and set paused to true (2034ms)
- ✓ pt006 should unpause and set paused to false (4095ms)
- ✓ pt020 should approve a spend and set allowed amount (2100ms)
- ✓ pt019 should blacklist and set blacklisted to true (2154ms)
- ✓ pt018 should blacklist and set blacklisted to true, then unblacklist and set blacklisted to false (3929ms)
- ✓ pt015 should configureMinter, setting the minter to true and mintingAllowance to amount (1972ms)
- ✓ pt012 should mint the amount, increasing balance of recipient by amount, increasing total supply by amount, and decreasing minterAllowed by amount (4151ms)
- ✓ pt017 should burn amount of tokens and reduce balance and total supply by amount (4027ms)
- ✓ pt010 should removeMinter, setting the minter to false and minterAllowed to 0 (4664ms)
- ✓ pt008 should transfer, reducing sender balance by amount and increasing recipient balance by amount (6654ms)
- ✓ pt007 should transferFrom, reducing sender balance by amount and increasing recipient balance by amount (6498ms)
- ✓ pt004 should updateMasterMinter (2086ms)
- ✓ pt005 should updateBlacklister (2132ms)
- ✓ pt003 should updatePauser (1954ms)
- ✓ pt009 should set owner to _newOwner (2007ms)

Contract: FiatTokenV2: FiatToken positive

- ✓ pt000 should check that default variable values are correct (1890ms)
- ✓ pt011 should pause and set paused to true (2142ms)
- ✓ pt006 should unpause and set paused to false (4120ms)
- ✓ pt020 should approve a spend and set allowed amount (1974ms)
- ✓ pt019 should blacklist and set blacklisted to true (2152ms)

- ✓ pt018 should blacklist and set blacklisted to true, then unblacklist and set blacklisted to false (4044ms)
- ✓ pt015 should configureMinter, setting the minter to true and mintingAllowance to amount (2079ms)
- ✓ pt012 should mint the amount, increasing balance of recipient by amount, increasing total supply by amount, and decreasing minterAllowed by amount (4234ms)
- ✓ pt017 should burn amount of tokens and reduce balance and total supply by amount (4022ms)
- ✓ pt010 should removeMinter, setting the minter to false and minterAllowed to 0 (4024ms)
- ✓ pt008 should transfer, reducing sender balance by amount and increasing recipient balance by amount (6569ms)
- ✓ pt007 should transferFrom, reducing sender balance by amount and increasing recipient balance by amount (6501ms)
- ✓ pt004 should updateMasterMinter (2208ms)
- ✓ pt005 should updateBlacklister (2268ms)
- ✓ pt003 should updatePauser (2061ms)
- ✓ pt009 should set owner to _newOwner (2154ms)

Contract: FiatTokenV1: FiatToken proxy negative

- ✓ nut002 should fail to switch adminAccount with non-adminAccount as caller (1939ms)
- ✓ nut003 should fail to upgradeTo to null contract address (2025ms)
- ✓ nut004 should fail to upgradeToAndCall to null contract address (2381ms)
- ✓ nut005 should fail to initialize contract twice (2081ms)
- ✓ nut006 should fail to call contract function with adminAccount (1996ms)
- ✓ nut008 shoud fail to update proxy storage if state-changing function called directly in FiatToken (2010ms)
- ✓ nut009 should fail to call upgradeTo with non-adminAccount (2164ms)
- ✓ nut010 should fail to call updateToAndCall with non-adminAccount (2304ms)
- ✓ nut011 should fail to upgradeToAndCall with initialize (already set variables) (2201ms)

Contract: FiatTokenV1_1: FiatToken proxy negative

- ✓ nut002 should fail to switch adminAccount with non-adminAccount as caller (2204ms)
- ✓ nut003 should fail to upgradeTo to null contract address (2013ms)
- ✓ nut004 should fail to upgradeToAndCall to null contract address (2032ms)
- ✓ nut005 should fail to initialize contract twice (2099ms)
- ✓ nut006 should fail to call contract function with adminAccount (1985ms)
- ✓ nut008 shoud fail to update proxy storage if state-changing function called directly in FiatToken (2259ms)
- ✓ nut009 should fail to call upgradeTo with non-adminAccount (2254ms)
- ✓ nut010 should fail to call updateToAndCall with non-adminAccount (2139ms)
- ✓ nut011 should fail to upgradeToAndCall with initialize (already set variables) (2596ms)

Contract: FiatTokenV2: FiatToken proxy negative

- ✓ nut002 should fail to switch adminAccount with non-adminAccount as caller (2170ms)
- ✓ nut003 should fail to upgradeTo to null contract address (2250ms)
- ✓ nut004 should fail to upgradeToAndCall to null contract address (2095ms)
- ✓ nut005 should fail to initialize contract twice (2462ms)
- ✓ nut006 should fail to call contract function with adminAccount (2073ms)

- ✓ nut008 shoud fail to update proxy storage if state-changing function called directly in FiatToken (2183ms)
- ✓ nut009 should fail to call upgradeTo with non-adminAccount (2043ms)
- ✓ nut010 should fail to call updateToAndCall with non-adminAccount (2265ms)
- ✓ nut011 should fail to upgradeToAndCall with initialize (already set variables) (2303ms)

Contract: FiatTokenV1: FiatToken proxy positive

- ✓ upt001 should upgradeTo new contract and preserve data field values (2278ms)
- ✓ upt002 should upgradeToandCall to contract with new data fields set on initVX and ensure new fields are correct and old data is preserved (2607ms)
- ✓ upt003 should upgradeToAndCall to contract with new data fields set on initVX and new logic and ensure old data preserved,new logic works, and new fields correct (2905ms)
- ✓ upt008 should deploy upgraded version of contract with new data fields and without previous deployment and ensure new fields correct (2730ms)
- ✓ upt010 should deploy upgraded version of contract with new data fields and logic without previous deployment and ensure new logic works, and new fields correct (2820ms)
- ✓ upt004 should update proxy adminAccount with previous adminAccount (2068ms)
- ✓ upt005 should receive Transfer event on transfer when proxied after upgrade (2422ms)
- ✓ upt006 should upgrade while paused and upgraded contract should be paused as a result; then unpause should unpause contract (4251ms)
- ✓ upt007 should upgrade contract to original address (2270ms)
- ✓ upt009 should check that admin is set correctly by proxy constructor
- ✓ upt011 should upgradeToAndCall while paused and upgraded contract should be paused as a result (2422ms)
- ✓ upt012 should upgradeToAndCall while upgrader is blacklisted (2787ms)
- ✓ upt013 should upgradeToAndCall while new logic is blacklisted (2544ms)
- ✓ upt014 should upgradeTo while new logic is blacklisted (2209ms)

Contract: FiatTokenV1_1: FiatToken proxy positive

- ✓ upt001 should upgradeTo new contract and preserve data field values (2502ms)
- ✓ upt002 should upgradeToandCall to contract with new data fields set on initVX and ensure new fields are correct and old data is preserved (2997ms)
- ✓ upt003 should upgradeToAndCall to contract with new data fields set on initVX and new logic and ensure old data preserved,new logic works, and new fields correct (3070ms)
- ✓ upt008 should deploy upgraded version of contract with new data fields and without previous deployment and ensure new fields correct (2637ms)
- ✓ upt010 should deploy upgraded version of contract with new data fields and logic without previous deployment and ensure new logic works, and new fields correct (2766ms)
- ✓ upt004 should update proxy adminAccount with previous adminAccount (1942ms)
- ✓ upt005 should receive Transfer event on transfer when proxied after upgrade (2416ms)
- ✓ upt006 should upgrade while paused and upgraded contract should be paused as a result; then unpause should unpause contract (4353ms)
- ✓ upt007 should upgrade contract to original address (2163ms)
- ✓ upt009 should check that admin is set correctly by proxy constructor

- ✓ upt011 should upgradeToAndCall while paused and upgraded contract should be paused as a result (2472ms)
- ✓ upt012 should upgradeToAndCall while upgrader is blacklisted (2756ms)
- ✓ upt013 should upgradeToAndCall while new logic is blacklisted (2538ms)
- ✓ upt014 should upgradeTo while new logic is blacklisted (2318ms)

Contract: FiatTokenV2: FiatToken proxy positive

- ✓ upt001 should upgradeTo new contract and preserve data field values (2282ms)
- ✓ upt002 should upgradeToandCall to contract with new data fields set on initVX and ensure new fields are correct and old data is preserved (2814ms)
- ✓ upt003 should upgradeToAndCall to contract with new data fields set on initVX and new logic and ensure old data preserved,new logic works, and new fields correct (2685ms)
- ✓ upt008 should deploy upgraded version of contract with new data fields and without previous deployment and ensure new fields correct (2641ms)
- ✓ upt010 should deploy upgraded version of contract with new data fields and logic without previous deployment and ensure new logic works, and new fields correct (2704ms)
- ✓ upt004 should update proxy adminAccount with previous adminAccount (2243ms)
- ✓ upt005 should receive Transfer event on transfer when proxied after upgrade (2446ms)
- ✓ upt006 should upgrade while paused and upgraded contract should be paused as a result; then unpause should unpause contract (4651ms)
- ✓ upt007 should upgrade contract to original address (2547ms)
- ✓ upt009 should check that admin is set correctly by proxy constructor
- ✓ upt011 should upgradeToAndCall while paused and upgraded contract should be paused as a result (2837ms)
- ✓ upt012 should upgradeToAndCall while upgrader is blacklisted (2845ms)
- ✓ upt013 should upgradeToAndCall while new logic is blacklisted (2700ms)
- ✓ upt014 should upgradeTo while new logic is blacklisted (2310ms)

Contract: FiatTokenV2

- ✓ has the expected domain separator
- ✓ disallows calling initializeV2 twice (41ms)

behaves like a Rescuable

updateRescuer

- ✓ allows owner to change rescuer (79ms)
- ✓ does not allow non-owner to change rescuer (44ms)
- ✓ does not allow updating the rescuer to the zero address (41ms)

rescueERC20

- ✓ allows rescuer to rescue ERC20 (full amount) (98ms)
- ✓ allows rescuer to rescue ERC20 (partial amount) (102ms)
- ✓ reverts when the requested amount is greater than balance (43ms)
- ✓ reverts when the the given contract address is not ERC20 (58ms)
- ✓ does not allow non-rescuer to rescue ERC20 (138ms)

uses original storage slot positions

- ✓ retains original storage slots 0 through 13 (65ms)
- ✓ retains slot 14 for rescuer

- ✓ retains original storage slots for blacklisted mapping
- ✓ retains original storage slots for balances mapping
- ✓ retains original storage slots for allowed mapping
- ✓ retains original storage slots for minters mapping
- ✓ retains original storage slots for minterAllowed mapping

safe allowance

increaseAllowance

- ✓ increases allowance by a given increment (148ms)
- ✓ reverts if the increase causes an integer overflow (100ms)
- ✓ reverts if the contract is paused (78ms)
- ✓ reverts if either the owner or the spender is blacklisted (200ms)

decreaseAllowance

- ✓ decreases allowance by a given decrement (117ms)
- ✓ reverts if the decrease is greater than current allowance (47ms)
- ✓ reverts if the decrease causes an integer overflow (46ms)
- ✓ reverts if the contract is paused (84ms)
- ✓ reverts if either the owner or the spender is blacklisted (220ms)

GasAbstraction

EIP-3009

transferWithAuthorization

- ✓ has the expected type hash
- ✓ executes a transfer when a valid authorization is given (203ms)
- ✓ reverts if the signature does not match given parameters (55ms)
- ✓ reverts if the signature is not signed with the right key (63ms)
- ✓ reverts if the authorization is not yet valid (41ms)
- ✓ reverts if the authorization is expired (50ms)
- ✓ reverts if the authorization has already been used (99ms)
- ✓ reverts if the authorization has a nonce that has already been used by the signer (117ms)
- ✓ reverts if the authorization includes invalid transfer parameters (67ms)
- ✓ reverts if the contract is paused (92ms)
- ✓ reverts if the payer or the payee is blacklisted (270ms)

receiveWithAuthorization

- ✓ has the expected type hash
- ✓ executes a transfer when a valid authorization is given and the caller is the payee (206ms)
- ✓ reverts if the caller is not the payee (53ms)
- ✓ reverts if the signature does not match given parameters (50ms)
- ✓ reverts if the signature is not signed with the right key (66ms)
- ✓ reverts if the authorization is not yet valid (54ms)
- ✓ reverts if the authorization is expired (46ms)
- ✓ reverts if the authorization has already been used (101ms)
- ✓ reverts if the authorization has a nonce that has already been used by the signer (97ms)
- ✓ reverts if the authorization includes invalid transfer parameters (68ms)
- ✓ reverts if the contract is paused (104ms)

- ✓ everts if the payer or the payee is blacklisted (250ms)

cancelAuthorization

- ✓ has the expected type hash
- ✓ cancels unused transfer authorization if signature is valid (145ms)
- ✓ cannot be used to cancel someone else's authorization (170ms)
- ✓ everts if the authorization has already been used (106ms)
- ✓ reverts if the authorization has already been canceled (103ms)
- ✓ reverts if the contract is paused (91ms)

transferWithMultipleAuthorizations

- ✓ reverts if no transfer is provided (86ms)
- ✓ reverts if the FiatToken contract is paused (127ms)
- ✓ reverts if the length of params is invalid (47ms)
- ✓ reverts if the length of signatures is invalid (51ms)
- ✓ can execute one transfer (184ms)
- ✓ can execute multiple transfers (262ms)
- ✓ does not revert if one of the transfers fail, but atomic is false (227ms)
- ✓ reverts if one of the transfers fail and atomic is true (100ms)
- ✓ reverts with a generic message if the call fails with a reason string (187ms)
- ✓ reverts with a generic message if the call fails with no reason string (158ms)

EIP-2612

permit

- ✓ has the expected type hash
- ✓ grants allowance when a valid permit is given (226ms)
- ✓ reverts if the signature does not match given parameters (54ms)
- ✓ reverts if the signature is not signed with the right key (130ms)
- ✓ reverts if the permit is expired (53ms)
- ✓ reverts if the nonce given does not match the next nonce expected (84ms)
- ✓ reverts if the permit has already been used (108ms)
- ✓ reverts if the permit has a nonce that has already been used by the signer (112ms)
- ✓ reverts if the permit includes invalid approval parameters (71ms)
- ✓ reverts if the permit is not for an approval (61ms)
- ✓ reverts if the contract is paused (92ms)
- ✓ reverts if the owner or the spender is blacklisted (200ms)

Contract: FiatTokenV2_1

- ✓ has the expected domain separator
- ✓ disallows calling initializeV2 twice (39ms)

behaves like a Rescuable

updateRescuer

- ✓ allows owner to change rescuer (56ms)
- ✓ does not allow non-owner to change rescuer (41ms)
- ✓ does not allow updating the rescuer to the zero address (44ms)

rescueERC20

- ✓ llows rescuer to rescue ERC20 (full amount) (88ms)

- ✓ allows rescuer to rescue ERC20 (partial amount) (92ms)
- ✓ reverts when the requested amount is greater than balance (50ms)
- ✓ reverts when the the given contract address is not ERC20 (45ms)
- ✓ does not allow non-rescuer to rescue ERC20 (87ms)

uses original storage slot positions

- ✓ retains original storage slots 0 through 13 (73ms)
- ✓ retains slot 14 for rescuer
- ✓ retains original storage slots for blacklisted mapping
- ✓ retains original storage slots for balances mapping
- ✓ retains original storage slots for allowed mapping
- ✓ retains original storage slots for minters mapping
- ✓ retains original storage slots for minterAllowed mapping

safe allowance

increaseAllowance

- ✓ increases allowance by a given increment (134ms)
- ✓ reverts if the increase causes an integer overflow (94ms)
- ✓ reverts if the contract is paused (100ms)
- ✓ reverts if either the owner or the spender is blacklisted (236ms)

decreaseAllowance

- ✓ decreases allowance by a given decrement (172ms)
- ✓ reverts if the decrease is greater than current allowance (51ms)
- ✓ reverts if the decrease causes an integer overflow (65ms)
- ✓ reverts if the contract is paused (99ms)
- ✓ reverts if either the owner or the spender is blacklisted (237ms)

GasAbstraction

EIP-3009

transferWithAuthorization

- ✓ has the expected type hash
- ✓ executes a transfer when a valid authorization is given (179ms)
- ✓ reverts if the signature does not match given parameters (59ms)
- ✓ reverts if the signature is not signed with the right key (64ms)
- ✓ reverts if the authorization is not yet valid (45ms)
- ✓ reverts if the authorization is expired (50ms)
- ✓ reverts if the authorization has already been used (113ms)
- ✓ reverts if the authorization has a nonce that has already been used by the signer (121ms)
- ✓ reverts if the authorization includes invalid transfer parameters (61ms)
- ✓ reverts if the contract is paused (85ms)
- ✓ reverts if the payer or the payee is blacklisted (212ms)

receiveWithAuthorization

- ✓ has the expected type hash
- ✓ executes a transfer when a valid authorization is given and the caller is the payee (164ms)
- ✓ reverts if the caller is not the payee (58ms)
- ✓ reverts if the signature does not match given parameters (58ms)

- ✓ reverts if the signature is not signed with the right key (61ms)
- ✓ reverts if the authorization is not yet valid (52ms)
- ✓ reverts if the authorization is expired (48ms)
- ✓ reverts if the authorization has already been used (87ms)
- ✓ reverts if the authorization has a nonce that has already been used by the signer (100ms)
- ✓ reverts if the authorization includes invalid transfer parameters (72ms)
- ✓ reverts if the contract is paused (148ms)
- ✓ reverts if the payer or the payee is blacklisted (219ms)

cancelAuthorization

- ✓ has the expected type hash
- ✓ cancels unused transfer authorization if signature is valid (144ms)
- ✓ cannot be used to cancel someone else's authorization (164ms)
- ✓ reverts if the authorization has already been used (99ms)
- ✓ reverts if the authorization has already been canceled (96ms)
- ✓ reverts if the contract is paused (89ms)

transferWithMultipleAuthorizations

- ✓ reverts if no transfer is provided (48ms)
- ✓ reverts if the FiatToken contract is paused (100ms)
- ✓ reverts if the length of params is invalid (44ms)
- ✓ reverts if the length of signatures is invalid (52ms)
- ✓ can execute one transfer (184ms)
- ✓ can execute multiple transfers (273ms)
- ✓ does not revert if one of the transfers fail, but atomic is false (256ms)
- ✓ reverts if one of the transfers fail and atomic is true (103ms)
- ✓ reverts with a generic message if the call fails with a reason string (197ms)
- ✓ reverts with a generic message if the call fails with no reason string (144ms)

EIP-2612

permit

- ✓ has the expected type hash
- ✓ grants allowance when a valid permit is given (193ms)
- ✓ reverts if the signature does not match given parameters (72ms)
- ✓ reverts if the signature is not signed with the right key (55ms)
- ✓ reverts if the permit is expired (48ms)
- ✓ reverts if the nonce given does not match the next nonce expected (75ms)
- ✓ reverts if the permit has already been used (112ms)
- ✓ reverts if the permit has a nonce that has already been used by the signer (112ms)
- ✓ reverts if the permit includes invalid approval parameters (64ms)
- ✓ reverts if the permit is not for an approval (78ms)
- ✓ reverts if the contract is paused (90ms)
- ✓ reverts if the owner or the spender is blacklisted (230ms)

initializeV2_1

- ✓ transfers locked funds to a given address (153ms)
- ✓ blocks transfers to the contract address (123ms)

```
    ✓ disallows calling initializeV2_1 twice (88ms)
  version
    ✓ returns the version string
Contract: V2_1Upgrader
  upgrade
    ✓ upgrades, transfers proxy admin role to newProxyAdmin, runs tests, and self-destructs (1489ms)
    ✓ reverts if there is an error (390ms)
  abortUpgrade
    ✓ transfers proxy admin role to newProxyAdmin and self-destructs (315ms)
Contract: V2Upgrader
  upgrade
    ✓ upgrades, transfers proxy admin role to newProxyAdmin, runs tests, and self-destructs (996ms)
    ✓ reverts if there is an error (363ms)
  abortUpgrade
    ✓ transfers proxy admin role to newProxyAdmin and self-destructs (280ms)
  1655 passing (56m)
```

Notice! Zokyo Secured team has checked the whole set of contracts to operate correctly, with special attention to the upgradeability functionality. The team has provided additional exploratory testing in order to verify several edge-cases connected to correct EIP3009 and EIP2612 implementation, verify correctness of Proxy usage and upgrade logic.

Zokyo Secured team has carefully checked the whole set of unit tests and checked the original coverage of those tests. During the audit tests were verified for the correctness, wholesomeness, sufficient coverage, meaning of the scenarios, overall structure and the correct implementation (in spite of the covered functionality).

As the result - all tests are verified and the coverage is evaluated as conforming for security requirements. Original functionality has necessary coverage, standard and sub-standard functionality was verified by Zokyo Secured auditors in separate set of exploratory testing scenarios.

We are grateful to have been given the opportunity to work with the StablR team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the StablR team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo