Avant Max Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Jorge

August 28, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The protocol implements a token minting and burning system that bridges external assets into wrapped versions through a request-based mechanism. At its core, `RequestsManager` serves as the central orchestrator, accepting deposits of whitelisted tokens from approved providers who request minting of the wrapped token. These requests enter a pending state until a designated service role completes them with the final mint amount, which may be greater but not lesser than the requested amount due to exchange rates.

The system works bidirectionally - users can also burn their wrapped tokens to request redemption of the underlying assets, following the same request-complete pattern. The treasury address receives deposited tokens from completed mints and provides tokens for burns, maintaining clear fund flow separation from the operational contracts.

The architecture employs several supporting contracts to ensure operational security and flexibility:

- `SimpleToken` provides the ERC20 implementation with idempotent mint/burn functions that prevent double-processing through unique keys
- `AddressesWhitelist` maintains an optional KYC layer, allowing the protocol to restrict participation to verified addresses when regulatory compliance is required
- `PriceStorage` implements a price oracle system with configurable bounds checking, preventing extreme price movements between updates by enforcing upper and lower percentage limits on consecutive price changes

5 Audit Scope

The scope of this audit was limited to:

```
script/Deployment.s.sol
src/AddressesWhitelist.sol
src/PriceStorage.sol
```

```
src/RequestsManager.sol
src/SimpleToken.sol
```

Additionally we were asked to investigate the on-chain state of the following deployed contracts:

```
avBTCx (Avalanche)
 - SimpleToken: 0xa7C10C510df4B1702E1F36451dd29D7C3EDC760C (impl:
 ↪  0xf7cf101e9c3d6035a9f832a0c02efbce56f7dfc7)
 - PriceStorage: 0x40B418cF176731089B2537D027A14c78a86F2166 (impl:
 ↪  0x0c66fcd3eac84a04a198f2ffe00ca64f3c9272af)
 - RequestsManager: 0x5F0AEf33a03Bf0028fC46ddDD4a86ee3d29E2972
 - AddressesWhitelist: 0x89245A4Bd8948713Fd5f6DA7c84CF6D2b76BEd7B
 - avUSDx Token: 0xDd1cDFA52E7D8474d434cd016fd346701db6B3B9 (impl:
 ↪  0xaefa3ffe45781680d5ad99627b7eb9d79192b29a)
 - avUSDx Pricing Contract: 0x7b4e8103bdDD5bcA79513Fda22892BEE53bA9777 (impl:
 ↪  0x0ef7be66249217e7588ad9277806080472357923)
 - avUSDx Requests Manager: 0x4C129d3aA27272211D151CA39a0a01E4C16Fc887

avETHx (Ethereum)
 - SimpleToken: 0x2E8b7190eE84E7AC757Ddff42Ba14d4EAe24B865 (impl:
 ↪  0x1411f6d18d4e0015ee2dc22d4f7b6893dedbdbe1)
 - PriceStorage: 0x985b5eF57dBE19B4EE1eD43b9AdEB1A61f2f6f23 (impl:
 ↪  0x511c025ed8c04e363d398c21d165563adddbee99)
 - RequestsManager: 0xb8CE3a51233299F1aBAdc84e78820cca457F413a
 - AddressesWhitelist: 0x570C0EB59655407D137B207E60cE3aB103CC1c4d
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Avant Max smart contracts provided by Avant Protocol. In this period, a total of 14 issues were found.

We identified 2 Low severity issues with the remaining findings being informational and gas optimizations.

Code & Test Suite Analysis

The contracts were relatively simple with limited complexity and had been previously audited. No test suite was provided but we wrote one as part of the audit achieving 100% coverage which we provided to the client as an additional deliverable merged in PR12:

```
$ forge coverage
------------------------+------------------+------------------+-----------------+-----------------
| File                  | % Lines          | % Statements     | % Branches      | % Funcs         |
+=============================================================================================================+
| Deployment.s.sol      | 100.00% (30/30)  | 100.00% (36/36)  | 100.00% (0/0)   | 100.00% (1/1)   |
|-----------------------+------------------+------------------+-----------------+-----------------|
| AddressesWhitelist.sol | 100.00% (10/10)  | 100.00% (9/9)    | 100.00% (2/2)   | 100.00% (3/3)   |
|-----------------------+------------------+------------------+-----------------+-----------------|
| PriceStorage.sol      | 100.00% (31/31)  | 100.00% (37/37)  | 100.00% (7/7)   | 100.00% (5/5)   |
|-----------------------+------------------+------------------+-----------------+-----------------|
| RequestsManager.sol   | 100.00% (127/127) | 100.00% (114/114) | 100.00% (15/15) | 100.00% (27/27) |
|-----------------------+------------------+------------------+-----------------+-----------------|
| SimpleToken.sol       | 100.00% (22/22)  | 100.00% (12/12)  | 100.00% (2/2)   | 100.00% (8/8)   |
|-----------------------+------------------+------------------+-----------------+-----------------|
| Total                 | 100.00% (220/220) | 100.00% (208/208) | 100.00% (26/26) | 100.00% (44/44) |
------------------------+------------------+------------------+-----------------+-----------------
```

On-Chain State Analysis

The deployed contracts on Ethereum and Avalanche were essentially the same contracts as those being audited; the major difference was their manner of deployment resulted in them being no longer upgradeable. The deployed

contracts listed in the "Audit Scope" section on both Ethereum and Avalanche have no public upgrade functions and there is no `ProxyAdmin` as there would be for `TransparentUpgradeableProxy`. We verified this by using:

- block explorers to analyze the deployed code
- cast to see available public functions (eg `cast interface 0x2E8b7190eE84E7AC757Ddff42Ba14d4EAe24B865` and `cast interface 0xa7C10C510df4B1702E1F36451dd29D7C3EDC760C --chain avalanche`)
- cast to see implementation addresses (eg `cast implementation 0x2E8b7190eE84E7AC757Ddff42Ba14d4EAe24B865` and `cast implementation --rpc-url https://api.avax.network/ext/bc/C/rpc 0xa7C10C510df4B1702E1F36451dd29D7C3EDC760C`) then using `cast interface` on the implementation
- block explorers to analyze deployment-related transactions

We noted that on Ethereum the admin/owner are currently an EOA address and in finding L-2 recommended transferring ownership to a Multi-Sig which the client has planned.

Centralization Risks

The protocol operates under a trusted model with significant centralization in the `SERVICE_ROLE` which controls fulfillment of minting/burning operations and setting of oracle prices. By calling `RequestsManager::emergencyWithdraw` protocol admins can drain the `RequestsManager` contract of any ERC20 tokens including those which have been transferred into the contract by users requesting mints or burns; users interacting with the protocol must have complete trust in the protocol team.

The protocol provides this explanation for the existence of the `emergencyWithdraw` function: "`emergencyWithdraw` is a common function in smart contracts, designed to recover funds that may become stuck or accidentally sent to the contract. In Avant's case, this does not introduce additional risk since fund management is already handled through a trusted, controlled process with appropriate safeguards."

Summary

Project Name	Avant Max
Repository	Avant-Contracts-Max
Commit	1e2f0ad6ca38...
Fix Commit	cc074dbd6aeb...
Audit Timeline	Aug 22nd - Aug 26th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	7
Gas Optimizations	5
Total Issues	14

Summary of Findings

[L-1] `PriceStorage::setPrice` maximum lower and upper bounds can be easily bypassed by repeatedly calling the function in the same block	Acknowledged
[L-2] Use multi-sig wallet for contract admin and owner	Acknowledged
[I-1] In Solidity don't initialize to default values	Resolved
[I-2] Emit missing events	Resolved
[I-3] Remove obsolete `return` statements when using named return variables	Resolved
[I-4] Use named mapping parameters to make explicit the purpose of keys and values	Resolved
[I-5] Add `deadline` parameter for mint and burn requests in `RequestsManager`	Acknowledged
[I-6] Enable whitelist in `RequestsManager::constructor`	Acknowledged
[I-7] Add an identifier or descriptor to `PriceStorage` which indicates what token or other entity is being priced	Acknowledged
[G-1] Enable the optimizer	Resolved
[G-2] Improve `PriceStorage` storage packing	Resolved
[G-3] Use `msg.sender` when calling `safeTransfer` in `RequestsManager::cancelMint, cancelBurn`	Resolved
[G-4] Modifier `mintRequestExist` can be safely removed from `RequestsManager::cancelMint, cancelBurn` saving 1 identical storage read	Resolved
[G-5] Cache identical storage reads	Resolved

7 Findings

7.1 Low Risk

7.1.1 `PriceStorage::setPrice` maximum lower and upper bounds can be easily bypassed by repeatedly calling the function in the same block

Description: `PriceStorage::setPrice` limits the price movement to a maximum lower/upper range delta based on the current price, to prevent sudden extreme changes in price:

```
uint256 lastPriceValue = lastPrice.price;
if (lastPriceValue != 0) {
  uint256 upperBound = lastPriceValue + ((lastPriceValue * upperBoundPercentage) /
  ↪  BOUND_PERCENTAGE_DENOMINATOR);
  uint256 lowerBound = lastPriceValue - ((lastPriceValue * lowerBoundPercentage) /
  ↪  BOUND_PERCENTAGE_DENOMINATOR);
  if (_price > upperBound || _price < lowerBound) {
    revert InvalidPriceRange(_price, lowerBound, upperBound);
  }
}
```

But this can be easily bypassed by repeatedly calling the `setPrice` function multiple times in the same block, each time decreasing or increasing the `lastPrice` by the current maximum allowed lower/upper bound.

Impact: The limitation of wild price fluctuations can be trivially bypassed so is ineffective, though only by entities having `SERVICE_ROLE`.

Recommended Mitigation: Add a configurable parameter `minPriceUpdateDelay` to the`PriceStorage` contract which only `DEFAULT_ADMIN_ROLE` can change. Then in `setPrice`:

```
error PriceUpdateTooSoon(uint256 lastUpdate, uint256 minWaitTime);

// In setPrice:
if(lastPrice.timestamp != 0) {
    if(block.timestamp < lastPrice.timestamp + minPriceUpdateDelay) {
        revert PriceUpdateTooSoon(lastPrice.timestamp, minPriceUpdateDelay);
    }
}
```

Avant: Acknowledged: the suggestion is valid, and we might consider the change in the future if we automate price setting. Currently, the price is calculated manually after a careful NAV process conducted weekly and will most likely be outsourced to an independent party for increased transparency. Once calculated, the price update transaction is also posted manually and requires a quorum of approvals before being submitted on-chain. This setup ensures that no repeated calls or incorrect parameters are ever attempted, and for now, the current code constraints suit our requirements.

7.1.2 Use multi-sig wallet for contract admin and owner

Description: As part of the audit we were asked to investigate the on-chain state of the deployed contracts. Using Ethereum as the example:

- the primary deployer appears to be 0xA5Ab0683d4f4AD107766a9fc4dDd49B5a960e661
- it has transferred ownership and admin to 0xD47777Cf34305Dec4F1095F164792C1A4AFB327e

Both of these are EOA addresses:

```
cast code 0xA5Ab0683d4f4AD107766a9fc4dDd49B5a960e661 --rpc-url $ETH_RPC_URL
cast code 0xD47777Cf34305Dec4F1095F164792C1A4AFB327e --rpc-url $ETH_RPC_URL
0x
0x
```

Impact: EOAs present significant security risks:

- Single point of failure (one compromised private key = total control)
- No ability to implement time delays, spending limits, or approval requirements
- No recovery mechanism if keys are lost
- Vulnerable to phishing, device compromise, or coercion

Recommended Mitigation: Transfer admin and ownership to a multi-signature wallet (e.g., Gnosis Safe) with:

- Minimum 3-of-5 or 2-of-3 threshold
- Time delays for critical operations
- Trusted Third-Party Entity who can veto/cancel time-locked admin actions
- Signers using hardware wallets
- Documented key management procedures

Avant: Acknowledged; Avant intends to transfer contract ownership and admin privileges to EOAs within the ForDeFi custodian ecosystem used to hold the protocol's funds. The MPC wallets associated with the RBAC structure provided by their platform function similarly to the suggested multisig configuration. Avant's setup ensures that only a quorum of admin-level users will be allowed to perform admin contract calls.

7.2 Informational

7.2.1 In Solidity don't initialize to default values

Description: In Solidity don't initialize to default values:

```
RequestsManager.sol
72:     for (uint256 i = 0; i < _allowedTokenAddresses.length; i++) {
```

Avant: Fixed in commit 0b590fa.

Cyfrin: Verified.

7.2.2 Emit missing events

Description: Emit missing events:

- `RequestsManager::constructor` should emit `AllowedTokenAdded`

Avant: Fixed in commit 7a3587c.

Cyfrin: Verified.

7.2.3 Remove obsolete `return` statements when using named return variables

Description: Remove obsolete `return` statements when using named return variables:

- `RequestManager::_addMintRequest, _addBurnRequest`

Avant: Fixed in commit 7a3587c.

Cyfrin: Verified.

7.2.4 Use named mapping parameters to make explicit the purpose of keys and values

Description: Named mapping parameters are already being used in almost all of the codebase, the one exception is:

```
SimpleToken.sol
13:   mapping(bytes32 => bool) private mintIds;
14:   mapping(bytes32 => bool) private burnIds;
```

Avant: Fixed in commit 1cc43e3.

Cyfrin: Verified.

7.2.5 Add `deadline` parameter for mint and burn requests in `RequestsManager`

Description: `RequestsManager::requestMint` and `requestBurn` allow callers to specify the minimum output amount but don't allow callers to specify a deadline for the request to be completed.

Minimum output amounts become "stale" over time; what users would expect as the minimum today could be different tomorrow and different again next week.

It would be ideal to allow callers of `RequestsManager::requestMint` and `requestBurn` to specify a deadline by which the requests must be completed. Past the deadline completion should revert but cancellation must remain possible.

Avant: Considering users can cancel mint and burn requests at any time, and they have already specified their minimum expected output amount, we believe the suggested extra constraint might not add much value while increasing the complexity of the UX.

7.2.6 Enable whitelist in `RequestsManager::constructor`

Description: Currently `RequestsManager::constructor` has the whitelist enablement commented out:

```
constructor(
  address _issueTokenAddress,
  address _treasuryAddress,
  address _providersWhitelistAddress,
  address[] memory _allowedTokenAddresses
) AccessControlDefaultAdminRules(1 days, msg.sender) {
  // *snip : irrelevant stuff* //

  // @audit commented out, starts in permissionless state
  // isWhitelistEnabled = true;
}
```

It is more defensive to enable the whitelist in the constructor to start in a restricted state, rather than starting in a permissionless state.

Avant: Acknowledged: The whitelisting feature was not on Avant's short-term roadmap, hence the comment. We agree that starting with it adds marginal defense, but since minting and redeeming are two-step request/complete processes that we control, we accepted the tradeoff.

7.2.7 Add an identifier or descriptor to `PriceStorage` which indicates what token or other entity is being priced

Description: The `PriceStorage` contract contains no identifier or descriptor that would readily indicate what is being priced.

Consider adding an identifier or a descriptor such as a string that has the name of the token or entity being priced.

Avant: Acknowledged: Avant will consider adding token identifiers in future `PriceStorage` deployments.

7.3 Gas Optimization

7.3.1 Enable the optimizer

Description: Enable the Foundry optimizer in `foundry.toml`:

```
+ optimizer = true
+ optimizer_runs = 1_000
```

Avant: Fixed in commit 871140d.

Cyfrin: Verified.

7.3.2 Improve `PriceStorage` storage packing

Description: * `PriceStorage::upperBoundPercentage` and `lowerBoundPercentage` can be safely declared as `uint128` to pack them into the same storage slot:

```
- uint256 public upperBoundPercentage;
- uint256 public lowerBoundPercentage;
+ uint128 public upperBoundPercentage;
+ uint128 public lowerBoundPercentage;
```

This reduces gas costs of every call to `PriceStorage::setPrice` where they are read together.

- `IPriceStorage::Price` can safely use `uint128` for `price` and `timestamp` to pack each `Price` struct into the same storage slot:

```
interface IPriceStorage {
  struct Price {
-   uint256 price;
-   uint256 timestamp;
+   uint128 price;
+   uint128 timestamp;
  }
```

This saves two storage writes in `PriceStorage::setPrice` when writing to `prices[key]` and `lastPrice`.

Avant: Fixed in commits 0325fcd, d40fc3d.

Cyfrin: Verified.

7.3.3 Use `msg.sender` when calling `safeTransfer` in `RequestsManager::cancelMint, cancelBurn`

Description: `RequestsManager::cancelMint` first enforces that `request.provider == msg.sender`:

```
    _assertAddress(request.provider, msg.sender);
```

Hence it can directly use `msg.sender` instead of `request.provider` to save 1 storage read when calling `safeTransfer`:

```
-   depositedToken.safeTransfer(request.provider, request.amount);
+   depositedToken.safeTransfer(msg.sender, request.amount);
```

The same applies to `RequestsManager::cancelBurn`.

Avant: Fixed in commit 7a3587c.

Cyfrin: Verified.

7.3.4 Modifier `mintRequestExist` can be safely removed from `RequestsManager::cancelMint, cancelBurn` saving 1 identical storage read

Description: `RequestsManager::cancelMint` validates that `request.provider == msg.sender`:

```
L152:     _assertAddress(request.provider, msg.sender);
```

Hence the modifier `mintRequestExist(_id)` can be safely removed to save 1 identical storage read of `mintRequests[_id].provider`.

The same applies to `RequestsManager::cancelBurn`.

Avant: Fixed in commit f74d1fc.

Cyfrin: Verified.

7.3.5 Cache identical storage reads

Description: Reading from storage is expensive; cache identical storage reads:

- `RequestsManager.sol`:

```
// cache `request.amount` in `RequestsManager::completeBurn`
239:     issueToken.burn(_idempotencyKey, address(this), request.amount);
244:     emit BurnRequestCompleted(_id, request.amount, _withdrawalAmount);
```

Avant: Fixed in commit 9bf3b60.

Cyfrin: Verified.