ZKMinter Delay Mod V1 Review

August 2, 2025

Prepared for zkSync

Conducted by:

Richie Humphrey (devtooligan)

About the zkSync ZKMinter Delay Mod V1 Review

zkSync Era is a Layer 2 ZK rollup that uses cryptographic validity proofs to provide scalable and low-cost transactions on Ethereum. The CappedMinter enables allocation of ZK tokens through a contract that allows the owner to mint up to a predetermined amount of tokens.

The ZKMinter Delay Mod V1 extends the Capped Minter with a delay mechanism for token minting, requiring authorized minters to create mint requests that must wait for a configurable delay period before execution.

About Offbeat Security

Offbeat Security is a boutique security company providing unique security solutions for complex and novel crypto projects. Our mission is to elevate the blockchain security landscape through invention and collaboration.

Summary & Scope

The src folder of the `zkminters` repo was reviewed at commit 858e320.

The following **2 files** were in scope:

- src/ZkMinterDelayV1.sol
- src/ZkMinterDelayV1Factory.sol

The delay module implements time-locked minting operations where authorized minters create requests that can only be executed after a specified delay period. The system includes veto functionality for emergency intervention.

The review identified 3 LOW and 5 INFO severity issues. Key findings include incorrect event emission in the mint request process, missing address validation that can lead to unexecutable mint requests, and missing validation for already-vetoed requests.

Summary of Findings

Identifier	Title	Severity	Fixed
L-01	Incorrect mint request ID emitted in MintRequested event	Low	Fixed in PR#24
L-02	MintRequested event passes creation timestamp instead of executable timestamp	Low	Fixed in PR#24
L-03	Missing address validation causes mint requests which can never be executed	Low	Fixed in PR#23
I-01	Veto function fails silently when request is already vetoed	Info	Fixed in PR#25
I-02	Missing zero amount validation in mint function	Info	Acknowledged
I-03	Unused error declarations in ZkMinterDelayV1 contract	Info	Fixed in PR#25
I-04	Delay timing fence-post error requires extra time beyond intended delay	Info	Fixed in PR#25
I-05	MintRequested event missing key parameters for effective monitoring	Info	Fixed in PR#24

Additional Recommendations

Code Quality

- The `updateMintDelay()` function is callable even when the contract is closed, unlike other state-changing functions that check for the closed state. Consider adding a `_revertIfClosed()` check to maintain consistency and avoid confusion about which operations are permitted on a closed contract.
- Consider initializing `nextMintRequestId` to 1 instead of 0. Since uninitialized storage variables in the EVM default to 0, integrators storing mint request IDs in mappings cannot differentiate between a missing key (which returns 0) and a valid mint request with ID 0. Starting IDs at 1 would eliminate this ambiguity and follow common practice for ID generation.

Detailed Findings

Low Findings

[L-01] Incorrect mint request ID emitted in MintRequested event

Description

The `mint()` function in `ZkMinterDelayV1` emits the `MintRequested` event with an incorrect mint request ID. The function stores the mint request using the current value of `nextMintRequestId`, then increments this value, but emits the event using the incremented value instead of the actual ID used for storage.

```
function mint(address _to, uint256 _amount) external virtual {
  // ... validation checks ...

  mintRequests[nextMintRequestId] = MintRequest({...}); // Uses current nextMint
  nextMintRequestId++;                                  // Increments the value
  emit MintRequested(nextMintRequestId, _createdAt);    // Emits incremented va
}
```

This results in the event emitting an ID that is one higher than the actual mint request ID, causing confusion for off-chain systems and users monitoring these events.

Recommendation

Store the current mint request ID before incrementing and use it in the event emission:

```
function mint(address _to, uint256 _amount) external virtual {
  _revertIfClosed();
  _requireNotPaused();
  _checkRole(MINTER_ROLE, msg.sender);

  uint48 _createdAt = uint48(block.timestamp);
```

```
+ uint256 currentMintRequestId = nextMintRequestId++;

- mintRequests[nextMintRequestId] =
+ mintRequests[currentMintRequestId] =
    MintRequest({minter: msg.sender, to: _to, amount: _amount, createdAt: _create

- nextMintRequestId++;

- emit MintRequested(nextMintRequestId, _createdAt);
+ emit MintRequested(currentMintRequestId, _createdAt);
}
```

Also we recommend adding a test to ensure it's working properly.

[L-02] MintRequested event passes creation timestamp instead of executable timestamp

Description

The `MintRequested` event in the ZkMinterDelayV1 contract has a parameter named `executableAt` that suggests it should contain the timestamp when the mint request becomes executable. However, the `mint()` function passes the creation timestamp (`_createdAt`) instead of the actual executable timestamp (`_createdAt + mintDelay`).

The event is defined as:

```
event MintRequested(uint256 indexed mintRequestId, uint48 executableAt);
```

But in the `mint()` function, it emits:

```
emit MintRequested(nextMintRequestId, _createdAt);
```

This creates confusion for off-chain systems listening to this event, as they would expect to receive the timestamp when the mint request can be executed, but instead receive when it was created.

Recommendation

Update the `mint()` function to pass the correct executable timestamp:

```
function mint(address _to, uint256 _amount) external virtual {
    _revertIfClosed();
    _requireNotPaused();
    _checkRole(MINTER_ROLE, msg.sender);

    uint48 _createdAt = uint48(block.timestamp);

    mintRequests[nextMintRequestId] =
      MintRequest({minter: msg.sender, to: _to, amount: _amount, createdAt: _crea
```

```diff
    nextMintRequestId++;

-    emit MintRequested(nextMintRequestId, _createdAt);
+    emit MintRequested(nextMintRequestId, _createdAt + mintDelay);
}
```

[L-03] Missing address validation causes mint requests which can never be executed

Description

The `mint()` function in ZkMinterDelayV1 does not validate that the `_to` parameter is not the zero address. Mint requests with `address(0)` as the recipient will be accepted and queued, but will fail during execution after the full delay period has elapsed.

When `executeMint()` is called, it attempts to mint tokens by calling `mintable.mint(mintRequest.to, mintRequest.amount)`. The underlying mintable contract (based on OpenZeppelin's `ERC20Upgradeable.sol`) includes a requirement that the recipient address is not zero:

```
require(account != address(0), "ERC20: mint to the zero address");
```

This means that mint requests created with `_to = address(0)` will waste the entire delay period before failing, requiring users to submit a new request and wait the full delay again.

Recommendation

Add address validation to the `mint()` function to reject requests with zero addresses immediately:

```diff
function mint(address _to, uint256 _amount) external virtual {
    _revertIfClosed();
    _requireNotPaused();
    _checkRole(MINTER_ROLE, msg.sender);

+    if (_to == address(0)) {
+        revert ZkMinterDelayV1__InvalidZeroAddress();
+    }

    uint48 _createdAt = uint48(block.timestamp);
    // ... rest of function
}
```

Informational Findings

[I-01] Veto function fails silently when request is already vetoed

Description

The `vetoMintRequest()` function in ZkMinterDelayV1 does not check whether a mint request has already been vetoed before processing the veto operation. This allows the function to execute successfully on already-vetoed requests, setting the `vetoed` flag redundantly and emitting the `MintRequestVetoed` event again.

The function performs validation checks for invalid requests and already-executed requests, but omits the check for already-vetoed requests:

```
function vetoMintRequest(uint256 _mintRequestId) external virtual {
    _checkRole(VETO_ROLE, msg.sender);

    MintRequest storage mintRequest = mintRequests[_mintRequestId];

    // check if the mint request is valid
    if (mintRequest.createdAt == 0) {
        revert ZkMinterDelayV1__InvalidMintRequest(_mintRequestId);
    }

    // revert if mint request has already been executed
    if (mintRequest.executed) {
        revert ZkMinterDelayV1__MintAlreadyExecuted(_mintRequestId);
    }

    // Missing check for already vetoed requests

    mintRequest.vetoed = true;  // Redundant if already vetoed
    emit MintRequestVetoed(_mintRequestId);  // Duplicate event emission
}
```

This silent failure behavior could mask bugs in the caller's logic. If a caller attempts to veto an already-vetoed request, they may not realize there's an issue with their state tracking or business logic, as the function will appear to succeed without indicating that no meaningful action was taken. This could lead to undetected errors in calling contracts or off-chain systems that rely on the veto mechanism.

Recommendation

Add a validation check to prevent vetoing already-vetoed requests:

```
function vetoMintRequest(uint256 _mintRequestId) external virtual {
    _checkRole(VETO_ROLE, msg.sender);

    MintRequest storage mintRequest = mintRequests[_mintRequestId];

    // check if the mint request is valid
    if (mintRequest.createdAt == 0) {
        revert ZkMinterDelayV1__InvalidMintRequest(_mintRequestId);
    }

    // revert if mint request has already been executed
    if (mintRequest.executed) {
        revert ZkMinterDelayV1__MintAlreadyExecuted(_mintRequestId);
    }

+   // revert if mint request has already been vetoed
+   if (mintRequest.vetoed) {
+       revert ZkMinterDelayV1__MintRequestVetoed(_mintRequestId);
+   }

    mintRequest.vetoed = true;
    emit MintRequestVetoed(_mintRequestId);
}
```

[I-02] Missing zero amount validation in mint function

Description

The `mint()` function in ZkMinterDelayV1 does not validate that the `_amount` parameter is greater than zero. This allows users to create mint requests for zero tokens, which serves no practical purpose and wastes gas and storage. The contract already defines a `ZkMinterDelayV1__InvalidAmount()` error, suggesting this validation was considered but not implemented.

Recommendation

Add a validation check for zero amounts in the `mint()` function:

```
function mint(address _to, uint256 _amount) external virtual {
    _revertIfClosed();
    _requireNotPaused();
    _checkRole(MINTER_ROLE, msg.sender);

+   if (_amount == 0) {
+       revert ZkMinterDelayV1__InvalidAmount();
+   }

    uint48 _createdAt = uint48(block.timestamp);
    ...
}
```

[I-03] Unused error declarations in ZkMinterDelayV1 contract

Description

The `ZkMinterDelayV1` contract declares two custom errors that are never used:

- `ZkMinterDelayV1__InvalidZeroAddress()`
- `ZkMinterDelayV1__InvalidAmount()`

These errors are defined but never thrown anywhere in the contract code, indicating either missing validation logic or unnecessary error declarations.

Recommendation

Consider removing the unused error declarations to clean up the contract if they are not needed.

[I-04] Delay timing fence-post error

Description

The `executeMint()` function in ZkMinterDelayV1 uses a strict inequality check that requires users to wait one additional second beyond the configured mint delay. The current implementation uses `<=` in the delay validation:

```
// check if the mint delay has elapsed
if (block.timestamp <= mintRequest.createdAt + mintDelay) {
  revert ZkMinterDelayV1__MintRequestNotReady(_mintRequestId);
}
```

This condition requires `block.timestamp` to be strictly greater than `mintRequest.createdAt + mintDelay`, meaning a mint request created at timestamp 100 with a 60-second delay can only be executed at timestamp 161 or later, not at timestamp 160 as expected.

The expectation is that the request can be executed once the mint delay is passed and the `MintRequested` event refers to this time as "executable at".

Recommendation

Update the delay check to use the correct boundary condition:

```diff
// check if the mint delay has elapsed
- if (block.timestamp <= mintRequest.createdAt + mintDelay) {
+ if (block.timestamp < mintRequest.createdAt + mintDelay) {
    revert ZkMinterDelayV1__MintRequestNotReady(_mintRequestId);
}
```

[I-05] MintRequested event missing key parameters for effective monitoring

Description

The `MintRequested` event in ZkMinterDelayV1 only emits the `mintRequestId` and `executableAt` timestamp, but excludes the `to` address and `amount` parameters. This limits the utility of the event for off-chain monitoring and integration systems that need to track mint request details without making additional contract calls.

```
// Current event definition
event MintRequested(uint256 indexed mintRequestId, uint48 executableAt);
```

Off-chain systems monitoring mint requests would need to make separate `getMintRequest()` calls to retrieve the destination address and amount.

Recommendation

Consider updating the `MintRequested` event to include the essential mint request parameters:

```diff
- event MintRequested(uint256 indexed mintRequestId, uint48 executableAt);
+ event MintRequested(
+   uint256 indexed mintRequestId,
+   address indexed to,
+   uint256 amount,
+   uint48 executableAt
+ );

  function mint(address _to, uint256 _amount) external virtual {
    // ... existing validation ...

    uint48 _createdAt = uint48(block.timestamp);

    mintRequests[nextMintRequestId] = MintRequest({
      minter: msg.sender,
      to: _to,
      amount: _amount,
      createdAt: _createdAt,
      executed: false,
      vetoed: false
    });

    nextMintRequestId++;

-   emit MintRequested(nextMintRequestId, _createdAt);
+   emit MintRequested(nextMintRequestId, _to, _amount, _createdAt);
  }
```