Zellic

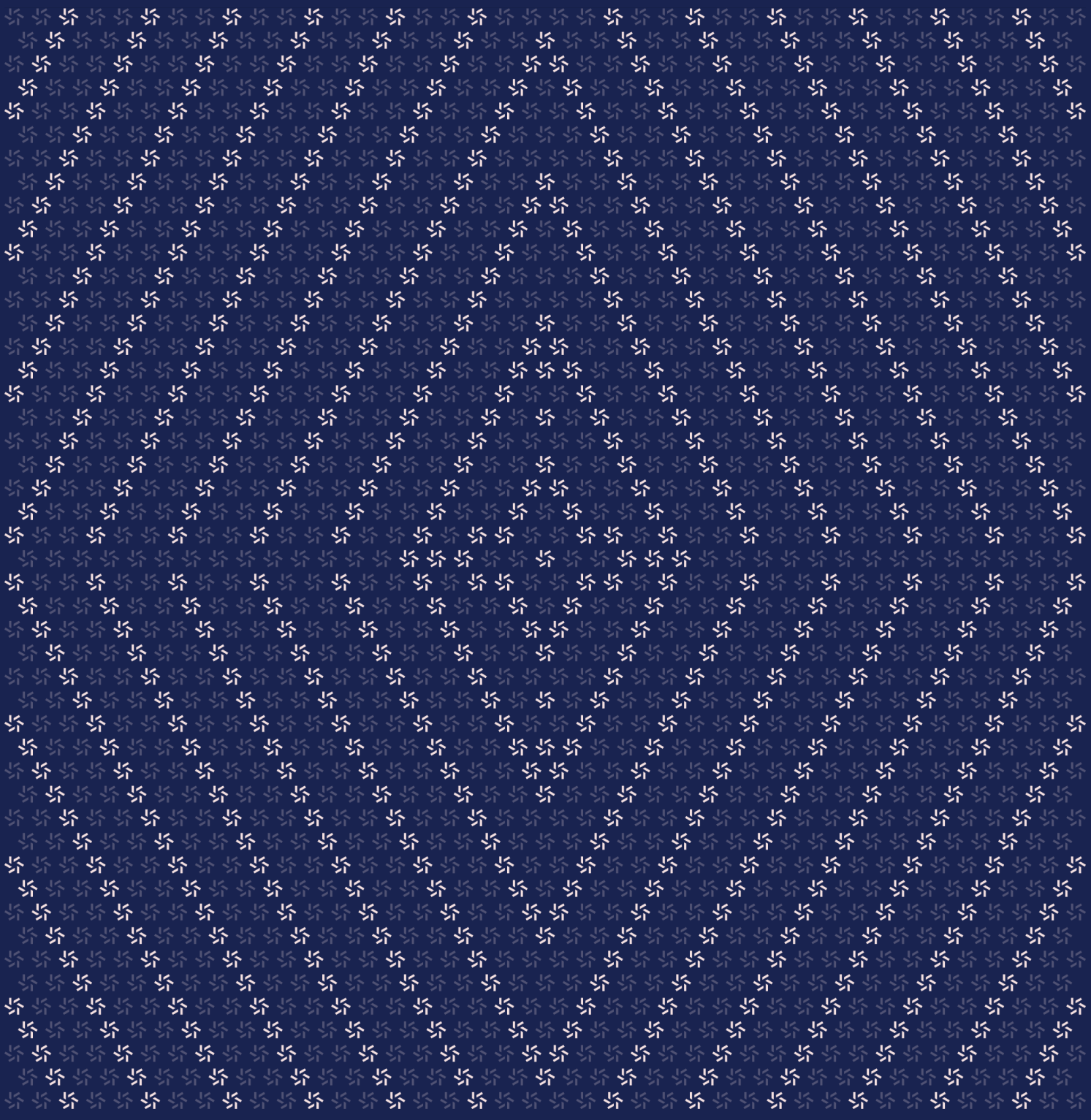
Prepared for
Péter Garamvölgyi
Scroll

Prepared by
Qingying Jie
Quentin Lemauf
Zellic

October 27, 2025

Cloak V1

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Scroll from October 16th to October 22nd, 2025. During this engagement, Zellic reviewed Cloak V1's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any ways of withdrawing more funds than intended, either via the canonical bridge or via the fast-withdraw vault?
- Are access controls implemented effectively to prevent unauthorized operations?
- Are there any vulnerabilities that could result in the loss of user funds?
- Are there ways that would allow third parties or operators to bypass the finalization process (validity proofs)?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

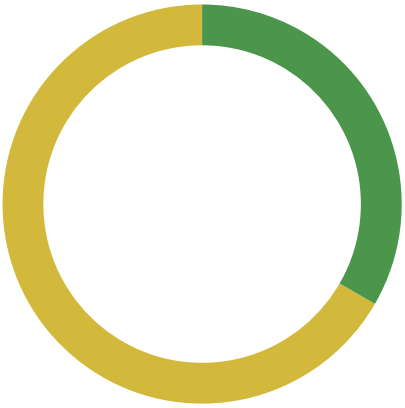
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Cloak V1 contracts, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	2
Low	1
Informational	0



2. Introduction

2.1. About Cloak V1

Scroll contributed the following description of Cloak V1:

> Cloak is a privacy solution deployed on Scroll as a layer-3 chain. It uses Scroll's zero-knowledge proof technology to ensure correctness of the L3 ledger, and uses strict access control for ensuring confidentiality and auditability.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect

its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Cloak V1 Contracts

Type	Solidity
Platform	EVM-compatible
Target	scroll-contracts
Repository	https://github.com/scroll-tech/scroll-contracts ↗
Version	`cbca22befd6bd0def75d74eea54453f08089d886`

Programs	`ScrollChainValidium.sol`
	`codec/BatchHeaderValidiumV0Codec.sol`
	`L1ScrollMessengerValidium.sol`
	`L1ERC20GatewayValidium.sol`
	`L1WETHGatewayValidium.sol`
	`FastWithdrawVault.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.1 person-weeks. The assessment was conducted by two consultants over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Qingying Jie
Engineer
qingying@zellic.io ↗

Quentin Lemauf
Engineer
quentin@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

October 16, 2025	Kick-off call
October 16, 2025	Start of primary review period
October 22, 2025	End of primary review period

3. Detailed Findings

3.1. The function `finalizeWithdrawERC20` does not implement the logic for forwarding data

Target	L1ERC20GatewayValidium		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The comment in the function `finalizeWithdrawERC20` states that the parameter `_data` is optional data that will be forwarded to the recipient. However, this function does not implement the logic to forward `_data` to the recipient.

```
/// @param _data Optional data to forward to recipient's account.
function finalizeWithdrawERC20(
    // [...]
    bytes calldata _data
) external payable;
```

Impact

When the function `finalizeWithdrawERC20` is called, the user is unable to receive the expected forwarded `_data`.

Recommendations

Consider implementing the logic to forward `_data` to the recipient.

Remediation

This issue has been acknowledged by Scroll, and a fix was implemented in PR #157 ↗. They intend to merge these changes into the production branch.

3.2. Front-running the encryption-key–registration transaction can deny service to function `finalizeBundle`

Target	ScrollChainValidium		
Category	Business Logic	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

Each encryption key is associated with a message index, meaning that this encryption key will be used starting from that message index. The first encryption key will be registered after the contract is deployed, meaning the key registration and the contract deployment are two separate transactions.

```
function registerNewEncryptionKey(bytes memory _key)
    external onlyRole(KEY_MANAGER_ROLE) {
    if (_key.length != 33) revert ErrorInvalidEncryptionKeyLength();
    uint256 _keyId = encryptionKeys.length;

    // The message from `nextCrossDomainMessageIndex` will utilise the newly
    registered encryption key.
    uint256 _msgIndex
    = IL1MessageQueueV2(messageQueueV2).nextCrossDomainMessageIndex();
    encryptionKeys.push(EncryptionKey(_msgIndex, _key));

    emit NewEncryptionKey(_keyId, _msgIndex, _key);
}
```

Impact

Malicious users can queue any cross-chain messages before the encryption-key–registration transaction, causing the return value of the `nextCrossDomainMessageIndex` function to be greater than zero.

When finalizing the bundle, the encryption key for message index 0 will be retrieved. Since the first encryption key is associated with a message index that is not zero, the `_getEncryptionKey` function will throw the `ErrorUnknownEncryptionKey` error, causing the bundle to fail finalization.

```
function _finalizeBundle(
    bytes calldata batchHeader,
```

```
    uint256 totalL1MessagesPoppedOverall,
    bytes calldata aggrProof
) internal virtual {
    // [...]

    // Get the encryption key at the time of on-chain message queue index.
    bytes memory encryptionKey = totalL1MessagesPoppedOverall == 0
        ? _getEncryptionKey(0)
        : _getEncryptionKey(totalL1MessagesPoppedOverall - 1);

    // [...]
}

function _getEncryptionKey(uint256 _msgIndex)
    internal view returns (bytes memory) {
    // [...]

    while (_encryptionKey.msgIndex > _msgIndex) {
        if (_numKeys == 0) revert ErrorUnknownEncryptionKey();
        _encryptionKey = encryptionKeys[--_numKeys];
    }

    return _encryptionKey.key;
}
```

Recommendations

Consider deploying the messenger after registering the first encryption key to prevent others from sending cross-chain messages or registering the first encryption key during the contract initialization.

Remediation

This issue has been acknowledged by Scroll.

3.3. The function `commitBatch` can be executed before the function `importGenesisBatch`

Target	ScrollChainValidium		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The function `importGenesisBatch` can set the batch hash and state root for index 0 (i.e., it imports the genesis batch).

```
function importGenesisBatch(bytes calldata _batchHeader)
    external onlyRole(GENESIS_IMPORTER_ROLE) {
    // [...]

    committedBatches[0] = _batchHash;
    stateRoots[0] = _postStateRoot;

    emit CommitBatch(0, _batchHash);
    emit FinalizeBatch(0, _batchHash, _postStateRoot, bytes32(0));
}
```

When the sequencer commits a batch, the function `commitBatch` does not check if the genesis batch has already been imported. It only ensures that the value of the `parentBatchHash` parameter is equal to `committedBatches[cachedLastCommittedBatchIndex]`.

```
function commitBatch(
    uint8 version,
    bytes32 parentBatchHash,
    bytes32 postStateRoot,
    bytes32 withdrawRoot,
    bytes calldata commitment
) external onlyRole(SEQUENCER_ROLE) whenNotPaused {
    if (postStateRoot == bytes32(0)) revert ErrorStateRootIsZero();

    uint256 cachedLastCommittedBatchIndex = lastCommittedBatchIndex;
    if (parentBatchHash != committedBatches[cachedLastCommittedBatchIndex]) {
        revert ErrorIncorrectBatchHash();
    }
```

```
        cachedLastCommittedBatchIndex += 1;
        bytes memory batchHeader = BatchHeaderValidiumVOCodec.encode(
            // [...]
        );
        bytes32 batchHash
        = BatchHeaderValidiumVOCodec.computeBatchHash(batchHeader);

        lastCommittedBatchIndex = cachedLastCommittedBatchIndex;
        committedBatches[cachedLastCommittedBatchIndex] = batchHash;
        stateRoots[cachedLastCommittedBatchIndex] = postStateRoot;
        withdrawRoots[cachedLastCommittedBatchIndex] = withdrawRoot;

        emit CommitBatch(cachedLastCommittedBatchIndex, batchHash);
    }
```

Impact

If the function `commitBatch` is successfully called before the function `importGenesisBatch`, the value of `committedBatches[0]` actually used will be `0`. This will affect the hash-value calculation for subsequent batches.

Recommendations

Consider ensuring that `committedBatches[cachedLastCommittedBatchIndex]` is not zero in the function `commitBatch`.

Remediation

This issue has been acknowledged by Scroll.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Module: FastWithdrawVault.sol

Function: `claimFastWithdraw(address l1Token, address to, uint256 amount, byte[32] messageHash, bytes signature)`

A user submits a withdrawal request on Cloak, sets the contract FastWithdrawVault as the target address, and provides the actual recipient address in the payload. The Cloak sequencer checks if the transaction is valid. If the checks are passed, the user can fast-withdraw tokens through this function with the signature provided by the sequencer.

Inputs

- `l1Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal the `l1Token` encoded in the signed message.
  - **Impact**: The address of the token to be withdrawn.
- `to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal the `to` encoded in the signed message.
  - **Impact**: The address of the recipient.
- `amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal the `amount` encoded in the signed message.
  - **Impact**: The amount of tokens to withdraw.
- `messageHash`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal the `messageHash` encoded in the signed message.
  - **Impact**: The corresponding withdraw-message hash on Cloak.
- `signature`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid EIP-712 signature over `(l1Token, l2Token, to, amount, messageHash)`.

- **Impact**: The signature of the message from the sequencer.

Branches and code coverage

Intended branches

- This function emits the event `Withdraw`.
  - ☑ Test coverage
- The user receives the token as expected.
  - ☑ Test coverage
- The amount of the corresponding token held by the contract decreases.
  - ☑ Test coverage

Negative behavior

- Reverts if the signer does not have `SEQUENCER_ROLE`.
  - ☑ Negative test
- Reverts if the signature is invalid.
  - ☑ Negative test
- Reverts if the fast withdraw is already processed.
  - ☑ Negative test

4.2. Module: L1ERC20GatewayValidium.sol

Function: `depositERC20(address _token, bytes _to, uint256 _amount, uint256 _gasLimit, uint256 _keyId)`

This function deposits some token to a recipient's account on Cloak. The caller needs to attach ETH to pay for the relayer fee.

Inputs

- `_token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be an ERC-20 token with metadata.
  - **Impact**: The deposited token address on Scroll.
- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.

    - **Impact**: The encrypted address of the recipient on Cloak.
- `_amount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be greater than zero.
    - **Impact**: The amount of tokens to transfer.
- `_gasLimit`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be lower than the `maxGasLimit` specified in the contract SystemConfig.
    - **Impact**: The gas limit required to complete the deposit on Cloak.
- `_keyId`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be the latest key ID stored in the contract ScrollChainValidium.
    - **Impact**: The encryption key ID of the key used to encrypt the address of the recipient.

Branches and code coverage

Intended branches

- Emits a `QueueTransaction` event when the deposit is successful.
    - ☑ Test coverage
- The `_token` balance of the caller decreases by `_amount`.
    - ☐ Test coverage
- Adjusts, for fee-on-transfer tokens, the amount based on the actual received tokens.
    - ☑ Test coverage

Negative behavior

- Reverts if trying to reenter the function.
    - ☑ Negative test
- Reverts if the `_keyId` is not the latest.
    - ☑ Negative test
- Reverts if the amount of `_token` received by the contract is zero.
    - ☑ Negative test
- Reverts if there is insufficient value to pay for the relayer fee.
    - ☐ Negative test

- Reverts if the `_token` does not have metadata.
  - ☐ Negative test
- Reverts if the `symbol` and `name` functions of the `_token` return data of type `bytes32`.
  - ☐ Negative test

Function call analysis

- `this._deposit(_token, this._msgSender(), _to, _amount, new bytes[0], _gasLimit, _keyId) -> this._transferERC20In(this._msgSender(), _token, _amount) -> SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable(_token), _from, address(this), _amount)`
  - **What is controllable?** `_token` and `_amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no changes to the state variables before this transfer, so reentry does not matter.
- `this._deposit(_token, this._msgSender(), _to, _amount, new bytes[0], _gasLimit, _keyId) -> IL1ScrollMessenger(this.messenger).sendMessage{value: msg.value}(this.counterpart, 0, _message, _gasLimit, _from)`
  - **What is controllable?** `_message` and `_gasLimit`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `depositERC20(address _token, address _realSender, bytes _to, uint256 _amount, uint256 _gasLimit, uint256 _keyId)`

This function deposits some token to a recipient's account on Cloak on behalf of the `_realSender`. The caller pays the tokens and needs to attach ETH to pay for the relayer fee, while the refunded ETH is sent to the `_realSender`.

Inputs

- `_token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be an ERC-20 token with metadata.

    - **Impact**: The deposited token address on Scroll.
- `_realSender`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be able to receive ETH refunds.
    - **Impact**: The address of the real sender on Scroll.
- `_to`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: N/A.
    - **Impact**: The encrypted address of the recipient on Cloak.
- `_amount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be greater than zero.
    - **Impact**: The amount of tokens to transfer.
- `_gasLimit`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be lower than the `maxGasLimit` specified in the contract SystemConfig.
    - **Impact**: The gas limit required to complete the deposit on Cloak.
- `_keyId`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be the latest key ID stored in the contract ScrollChainValidium.
    - **Impact**: The encryption-key ID of the key used to encrypt the address of the recipient.

Branches and code coverage

Intended branches

- Emits a `QueueTransaction` event when the deposit is successful.
    - ☑ Test coverage
- The `_token` balance of the caller decreases by `_amount`.
    - ☐ Test coverage
- Adjusts, for fee-on-transfer tokens, the amount based on the actual received tokens.
    - ☑ Test coverage

Negative behavior

- Reverts if trying to reenter the function.

☑ Negative test

- Reverts if the `_keyId` is not the latest.

☑ Negative test

- Reverts if the amount of `_token` received by the contract is zero.

☑ Negative test

- Reverts if there is insufficient value to pay for the relayer fee.

☐ Negative test

- Reverts if the `_token` does not have metadata.

☐ Negative test

- Reverts if the `symbol` and `name` functions of the `_token` return data of type `bytes32`.

☐ Negative test

Function call analysis

- `this._deposit(_token, _realSender, _to, _amount, new bytes[0], _gasLimit, _keyId) -> this._transferERC20In(this._msgSender(), _token, _amount) -> SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable(_token), _from, address(this), _amount)`
  - **What is controllable?** `_token`, `_from`, and `_amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no changes to the state variables before this transfer, so reentry does not matter.
- `this._deposit(_token, _realSender, _to, _amount, new bytes[0], _gasLimit, _keyId) -> IL1ScrollMessenger(this.messenger).sendMessage{value: msg.value}(this.counterpart, 0, _message, _gasLimit, _from)`
  - **What is controllable?** `_message`, `_gasLimit`, and `_from`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `finalizeWithdrawERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, bytes _data)`

This function completes the ERC-20 withdrawal from Cloak to Scroll and sends funds to the recipient's account on Scroll. The caller must be the contract L1ScrollMessengerValidium, and the sender on Cloak must be the contract L2StandardERC20Gateway.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: The result of `getL2ERC20Address(_l1Token)` must be equal to `_l2Token`.
  - **Impact**: The address of the withdrawn token on Scroll.
- `_l2Token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: The value of `tokenMapping[_l2Token]` of the contract L2StandardERC20Gateway on Cloak must not be the zero address.
  - **Impact**: The address of the withdrawn token on Cloak.
- `_from`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The address of the sender on Cloak.
- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The address of the recipient on Scroll.
- `_amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be greater than zero.
  - **Impact**: The amount of tokens to withdraw.
- `_data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The optional data. Only used for event logging in this implementation.

Branches and code coverage

Intended branches

- The `_l1Token` balance of the contract decreases by the expected amount.
  - ☑ Test coverage
- The `_l1Token` balance of `_to` increases by the expected amount.
  - ☑ Test coverage

- The `tokenMapping[_l1Token]` is set to `_l2Token` if this is the first withdrawal of the token.

  - ☐ Test coverage

Negative behavior

- Reverts if the caller is not the contract L1ScrollMessengerValidium.

  - ☐ Negative test
- Reverts if the sender on Cloak is not the contract L2StandardERC20Gateway.

  - ☑ Negative test
- Reverts if `msg.value` is not zero.

  - ☑ Negative test
- Reverts if `_l2Token` does not match the expected L2 token address computed from `_l1Token`.

  - ☐ Negative test

Function call analysis

- `SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(_l1Token), _to, _amount)`
  - **What is controllable?** `_l1Token`, `_to`, and `_amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

4.3. Module: L1WETHGatewayValidium.sol

Function: `deposit(bytes _to, uint256 _amount, uint256 _keyId)`

This function allows a user to deposit ETH to Cloak. It converts ETH to WETH for users and deposits through the contract L1ERC20GatewayValidium.

Inputs

- `_to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The encrypted address of the recipient on Cloak.

- `_amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be less than `msg.value` and must be greater than zero.
  - **Impact**: The amount of ETH to deposit.
- `_keyId`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the latest key ID.
  - **Impact**: The encryption-key ID of the key used to encrypt the address of the recipient.

Branches and code coverage

Intended branches

- This function can be successfully executed without reverting.
  - ☑ Test coverage
- The WETH balance of the `gateway` increases by `_amount`.
  - ☑ Test coverage

Negative behavior

- Reverts if `msg.value` is less than `_amount`.
  - ☑ Negative test
- Reverts if `_amount` is zero.
  - ☑ Negative test

Function call analysis

- `IL1ERC20GatewayValidium(this.gateway).depositERC20{value: msg.value - _amount}(this.WETH, msg.sender, _to, _amount, L1WETHGatewayValidium.GAS_LIMIT, _keyId)`
  - **What is controllable?** `msg.value`, `_amount`, `_to`, and `_keyId`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

4.4. Module: ScrollChainValidium.sol

Function: `getEncryptionKey(uint256 _keyId)`

This function verifies `keyId` is the most recently registered `encryptionKeys` inserted via `registerNewEncryptionKey` by the `KEY_MANAGER_ROLE` and returns it. It is called in the deposit flow by `L1ERC20GatewayValidium::_deposit` for sanity checks on the key.

Inputs

- `_keyId`
  - **Control**: Arbitrary caller.
  - **Constraints**: Must be the last one inserted in `encryptionKeys`.
  - **Impact**: Return the last `encryptionKey` if `keyId` is the last one.

Branches and code coverage

Intended branches

- Verifies at least one key exists.
  - ☐ Test coverage
- Returns the last key value, if `keyId` represents the last key ID.
  - ☐ Test coverage

Function: `registerNewEncryptionKey(bytes _key)`

This is the key-manager entry point that records a freshly rotated encryption key so downstream deposits can reference the latest key.

Inputs

- `_key`
  - **Control**: Key manager (`KEY_MANAGER_ROLE`).
  - **Constraints**: Must be exactly 33 bytes (compressed EC public key) — otherwise, it reverts with `ErrorInvalidEncryptionKeyLength`.
  - **Impact**: Appended to `encryptionKeys` along with the current message queue index for later lookup.

Branches and code coverage

Intended branches

- Reverts when `_key.length` is not 33.
  - ☐ Test coverage
- Successful registration caches the next queue index.
  - ☑ Test coverage

Negative behavior

- A caller without `KEY_MANAGER_ROLE` reverts.
  - ☑ Negative test

Function call analysis

- `IL1MessageQueueV2(this.messageQueueV2).nextCrossDomainMessageIndex()`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `getLatestEncryptionKey()`

This function returns the most recently registered `encryptionKeys` inserted via `registerNewEncryptionKey` by the `KEY_MANAGER_ROLE`.

Branches and code coverage

Intended branches

- Verifies at least one key exists.
  - ☐ Test coverage
- Returns the last key ID and its value.
  - ☐ Test coverage

Function: `importGenesisBatch(bytes _batchHeader)`

This function stores the first batch header (genesis); this can only be called once.

Inputs

- `_batchHeader`
  - **Control**: Genesis role (`GENESIS_IMPORTER_ROLE`).
  - **Constraints**: The length must be <= 105 bytes.
  - **Impact**: Store the genesis batch header.

Branches and code coverage

Intended branches

- Reverts when the decoded batch index is not zero.
  - ☐ Test coverage
- Reverts when the parent batch hash is nonzero.
  - ☐ Test coverage
- Reverts when the withdraw root is nonzero.
  - ☐ Test coverage
- Reverts when the post state root is zero.
  - ☐ Test coverage
- Reverts when genesis is already imported.
  - ☐ Test coverage
- Successful import stores roots.
  - ☑ Test coverage

Negative behavior

- A caller without `GENESIS_IMPORTER_ROLE` reverts via `onlyRole`.
  - ☑ Negative test
- Duplicate invocation reverts.
  - ☐ Negative test

Function call analysis

- `BatchHeaderValidiumVOCodec.loadAndValidate(_batchHeader)`
  - **What is controllable?** `_batchHeader` is controlled by the `GENESIS_IMPORTER_ROLE`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Sanity check the payload length and copy data from calldata to memory.

    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.getBatchIndex(batchPtr)`
    - **What is controllable?** Derived from header input var.
    - **If the return value is controllable, how is it used and how can it go wrong?** Must equal zero.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.getParentBatchHash(batchPtr)`
    - **What is controllable?** Derived from header input var.
    - **If the return value is controllable, how is it used and how can it go wrong?** Must be zero.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.getWithdrawRoot(batchPtr)`
    - **What is controllable?** Derived from header input var.
    - **If the return value is controllable, how is it used and how can it go wrong?** Must be zero.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.getPostStateRoot(batchPtr)`
    - **What is controllable?** Derived from header input var.
    - **If the return value is controllable, how is it used and how can it go wrong?** Stored as the genesis state root — zero roots revert.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.computeBatchHash(batchPtr, _length)`
    - **What is controllable?** Derived from header input var.
    - **If the return value is controllable, how is it used and how can it go wrong?** Stored as the genesis committed batch.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `commitBatch(uint8 version, byte[32] parentBatchHash, byte[32] postStateRoot, byte[32] withdrawRoot, bytes commitment)`

This is a sequencer entry point used to append the next batch header, enforcing continuity with the precedent committed batch.

Inputs

- `version`
  - **Control**: Sequencer (`SEQUENCER_ROLE`).
  - **Constraints**: N/A.
  - **Impact**: Encoded into the batch header.
- `parentBatchHash`
  - **Control**: Sequencer (`SEQUENCER_ROLE`).
  - **Constraints**: Enforces batch continuity by matching the latest stored batch hash.
  - **Impact**: Any mismatch reverts — otherwise anchors the new batch to the prior one.
- `postStateRoot`
  - **Control**: Sequencer (`SEQUENCER_ROLE`).
  - **Constraints**: Must be nonzero.
  - **Impact**: Encoded in the batch header and stored as the state root for the newly committed batch.
- `withdrawRoot`
  - **Control**: Sequencer (`SEQUENCER_ROLE`).
  - **Constraints**: N/A.
  - **Impact**: Persisted to `withdrawRoots` for later withdrawal validation.
- `commitment`
  - **Control**: Sequencer (`SEQUENCER_ROLE`).
  - **Constraints**: N/A.
  - **Impact**: Encoded in the batch header.

Branches and code coverage

Intended branches

- Reverts when `postStateRoot` is zero (`ErrorStateRootIsZero`).
  - ☐ Test coverage
- Reverts when `parentBatchHash` mismatches the latest committed hash (`ErrorIncorrectBatchHash`).
  - ☐ Test coverage
- Successful commit stores the new batch.
  - ☑ Test coverage

Negative behavior

- A caller without `SEQUENCER_ROLE` reverts via `onlyRole`.
  - ☑ Negative test
- Paused contract path exercised (`whenNotPaused`).
  - ☐ Negative test

Function call analysis

- `BatchHeaderValidiumVOCodec.encode(version, uint64(cachedLastCommittedBatchIndex), parentBatchHash, postStateRoot, withdrawRoot, commitment)`
  - **What is controllable?** All arguments besides the cached index are sequencer-controlled.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BatchHeaderValidiumVOCodec.computeBatchHash(batchHeader)`
  - **What is controllable?** Input `batchHeader` is fully determined by the sequencer-provided fields (encoded).
  - **If the return value is controllable, how is it used and how can it go wrong?** Produces the hash stored in `committedBatches`.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `revertBatch(bytes batchHeader)`

This is an admin entry point that removes committed but unfinalized batches starting from the supplied header and rewinds `lastCommittedBatchIndex`.

Inputs

- `batchHeader`
  - **Control**: Admin (`DEFAULT_ADMIN_ROLE`).
  - **Constraints**: Must decode to a batch already present in `committedBatches` and with an index strictly greater than `lastFinalizedBatchIndex`.
  - **Impact**: When valid, deletes batch metadata from the target index through the tip and decrements `lastCommittedBatchIndex`.

Branches and code coverage

Intended branches

- Reverts when the provided batch header is not committed.
  - ☐ Test coverage
- Reverts when attempting to revert a finalized batch.
  - ☐ Test coverage
- Successful rollback clears storage entries.
  - ☑ Test coverage

Negative behavior

- A caller without `DEFAULT_ADMIN_ROLE` reverts via `onlyRole`.
  - ☑ Negative test
- Attempting to revert the genesis batch fails earlier because it is already finalized.
  - ☐ Negative test

Function call analysis

- `this._loadBatchHeader(batchHeader, lastBatchIndex) -> BatchHeaderValidiumVOCodec.loadAndValidate(_batchHeader)`
  - **What is controllable?** Entire header bytes supplied by the admin.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns pointer/length only when the header is well-formed — malformed data reverts before state changes.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._loadBatchHeader(batchHeader, lastBatchIndex) -> BatchHeaderValidiumVOCodec.getBatchIndex(batchPtr)`
  - **What is controllable?** Derived from the decoded header.
  - **If the return value is controllable, how is it used and how can it go wrong?** Must be ≤ `lastCommittedBatchIndex` — out-of-range values revert.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._loadBatchHeader(batchHeader, lastBatchIndex) -> BatchHeaderValidiumVOCodec.computeBatchHash(batchPtr, length)`
  - **What is controllable?** Derived from the decoded header.
  - **If the return value is controllable, how is it used and how can it go wrong?** The resulting hash must match storage.

    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._loadBatchHeader(batchHeader, lastBatchIndex)`
    - **What is controllable?** Header inputs validated above.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the batch index.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `finalizeBundle(bytes batchHeader, uint256 totalL1MessagesPoppedOverall, bytes aggrProof)`

This is the prover entry point that validates an aggregated proof for pending batches.

Inputs

- `batchHeader`
    - **Control**: Prover (`PROVER_ROLE`).
    - **Constraints**: Must represent the latest committed batch.
    - **Impact**: Determines which batch index/hash are finalized and emitted.
- `totalL1MessagesPoppedOverall`
    - **Control**: Prover (`PROVER_ROLE`).
    - **Constraints**: Should align with the queue state proven in `aggrProof` — used to `getMessageRollingHash` and `_getEncryptionKey` and must not exceed queue bounds.
    - **Impact**: Dictates which cross-domain messages are marked finalized and which encryption key is included in public inputs.
- `aggrProof`
    - **Control**: Prover (`PROVER_ROLE`).
    - **Constraints**: Must satisfy `IRollupVerifier.verifyBundleProof`.
    - **Impact**: Successful verification authorizes updating finalization state.

Branches and code coverage

Intended branches

- Reverts when the header does not correspond to a committed batch.
    - ☐ Test coverage
- Reverts when finalizing an already verified batch.

- ☐ Test coverage
- Reverts when queue/proof validation fails.
  - ☐ Test coverage
- Successful finalization updates indexes and finalizes queue entries.
  - ☑ Test coverage

Negative behavior

- A caller without `PROVER_ROLE` reverts via `onlyRole`.
  - ☑ Negative test
- Paused-contract scenario (`whenNotPaused`) is not covered.
  - ☐ Negative test

Function call analysis

- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._beforeFinalizeBatch(batchHeader) -> this._loadBatchHeader(batchHeader, this.lastCommittedBatchIndex) -> BatchHeaderValidiumVOCodec.loadAndValidate(_batchHeader)`
  - **What is controllable?** Entire header bytes supplied by the admin.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns pointer/length only when the header is well-formed — malformed data reverts before state changes.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._beforeFinalizeBatch(batchHeader) -> this._loadBatchHeader(batchHeader, this.lastCommittedBatchIndex) -> BatchHeaderValidiumVOCodec.getBatchIndex(batchPtr)`
  - **What is controllable?** Derived from the decoded header.
  - **If the return value is controllable, how is it used and how can it go wrong?** Must be ≤ `lastCommittedBatchIndex` — out-of-range values revert.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._beforeFinalizeBatch(batchHeader) -> this._loadBatchHeader(batchHeader, this.lastCommittedBatchIndex) -> BatchHeaderValidiumVOCodec.computeBatchHash(batchPtr, length)`
  - **What is controllable?** Derived from the decoded header.
  - **If the return value is controllable, how is it used and how can it go wrong?**

The resulting hash must match storage.

- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._beforeFinalizeBatch(batchHeader) -> this._loadBatchHeader(batchHeader, this.lastCommittedBatchIndex)`
  - **What is controllable?** Header inputs validated above.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the batch index.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._beforeFinalizeBatch(batchHeader) -> BatchHeaderValidiumVOCodec.getVersion(batchPtr)`
  - **What is controllable?** Version byte embedded in the header.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> IL1MessageQueueV2(this.messageQueueV2).getMessageRollingHash (totalL1MessagesPoppedOverall - 1)`
  - **What is controllable?** Index argument derived from `totalL1MessagesPoppedOverall`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Produces the queue hash included in public inputs.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> IRollupVerifier(this.verifier).verifyBundleProof(version, batchIndex, aggrProof, publicInputs)`
  - **What is controllable?** Proof bytes from the prover.
  - **If the return value is controllable, how is it used and how can it go wrong?** Verification must succeed.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._finalizeBundle(batchHeader, totalL1MessagesPoppedOverall, aggrProof) -> this._afterFinalizeBatch(batchIndex, batchHash, totalL1MessagesPoppedOverall, postStateRoot, withdrawRoot) -> IL1MessageQueueV2(this.messageQueueV2).finalizePoppedCrossDomainMessage( totalL1MessagesPoppedOverall)`

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** Completes cross-domain–message finalization.
- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

5. Assessment Results

During our assessment on the scoped Cloak V1 contracts, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.