Zellic

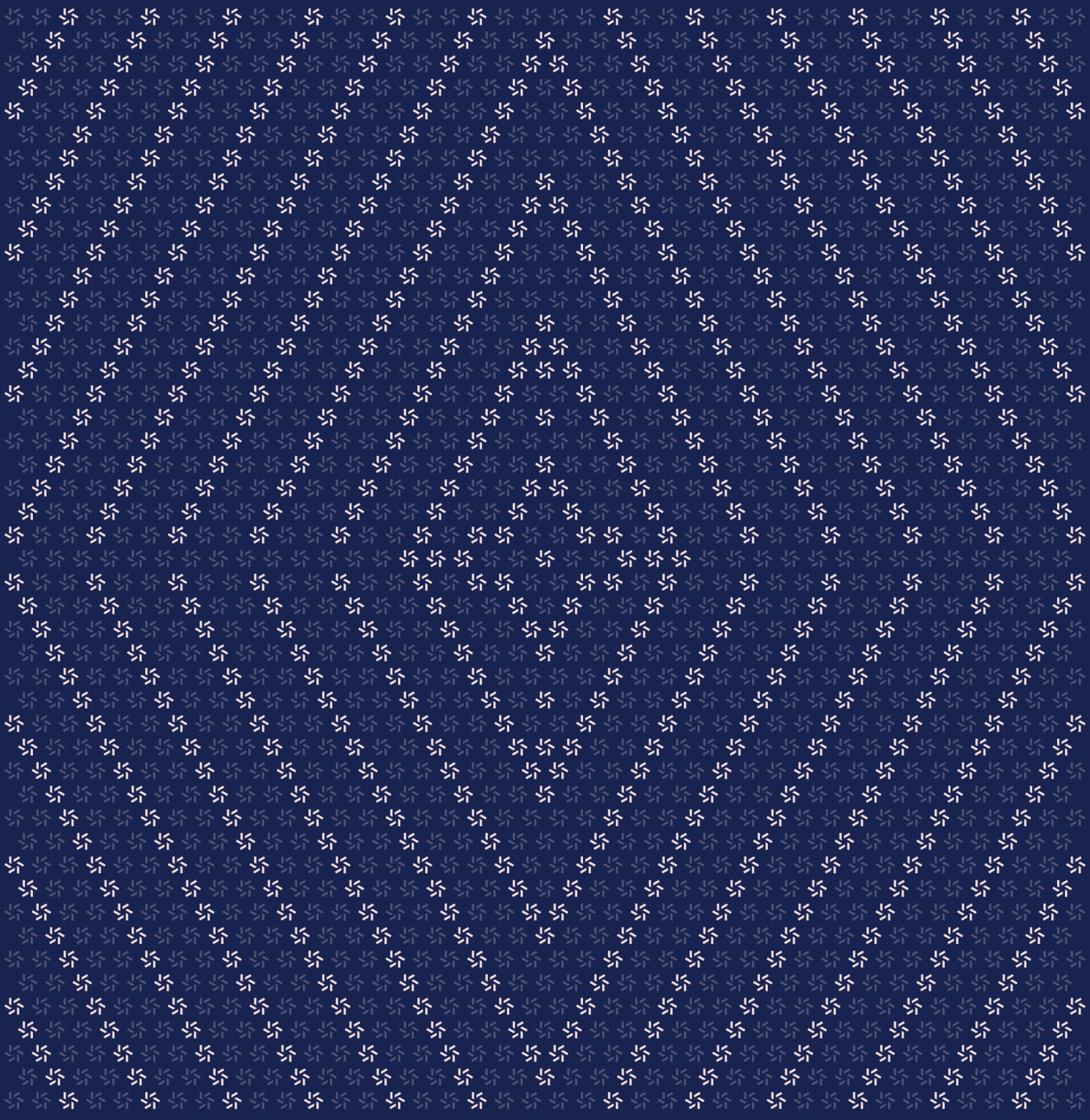
Prepared for
Aayush Gupta
ZK Email

Prepared by
Jade Han
Allen Roh
Zellic

August 30, 2024

ZK Email

Comprehensive 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 ZK Email from August 5th to August 26th, 2024. During this engagement, Zellic reviewed ZK Email'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:

- Can an attacker trigger malicious actions using inputs unrelated to the proof?
- Is it possible for an attacker to generate a manipulated proof by exploiting logical flaws in the circuit?
- Do the modules within the project comply with the ERC-7579 and Safe module extension format?
- Are there any scenarios where funds could be frozen or stolen?

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.

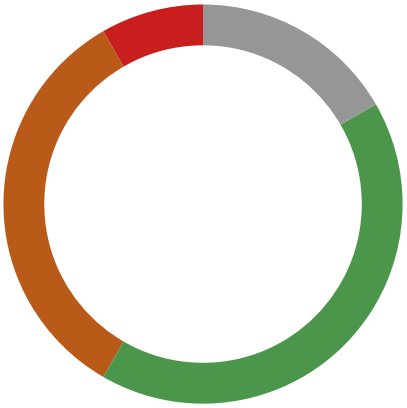
1.4. Results

During our assessment on the scoped ZK Email contracts, we discovered 12 findings. One critical issue was found. Four were of high impact, five were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for ZK Email's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	4
Medium	0
Low	5
Informational	2



2. Introduction

2.1. About ZK Email

ZK Email contributed the following description of ZK Email:

> One issue with existing applications on Ethereum is that all users who execute transactions on-chain must install Ethereum-specific tools such as wallets and manage their own private keys. The ZK Emailteam's ether email-auth SDK solves this issue: it allows users to execute any transaction on-chain simply by sending an email. As its concrete application, the ZK Email-team also provides email-based account recovery for account abstraction wallets such as Safe wallets. In social recovery, the account owner must appoint trusted persons as guardians who are authorized to update the private key for controlling the account. However, not all such persons are necessarily Ethereum users. The ZK Emailteam's solution mitigates this constraint by allowing guardians to complete the recovery process simply by sending an email. In other words, any trusted persons can work as guardians as long as they can send emails.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

ZK Email Contracts

Type	Solidity
Platform	EVM-compatible
Target	ether-email-auth
Repository	https://github.com/zkemail/ether-email-auth ↗
Version	`b5694a9e0e49d07a862232f665dc4d0886c5a15f`
Programs	`packages/contracts/*`
Target	email-recovery
Repository	https://github.com/zkemail/email-recovery ↗
Version	`85d3ff94677da3e9206d63815edcf8604a422245`
Programs	`src/*`

Target	zk-email-verify
Repository	https://github.com/zkemail/zk-email-verify ↗
Version	`13b41c2f3c4682ee16e6210999a649e8d6bfef18`
Programs	`blob/main/packages/contracts/DKIMRegistry.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-weeks. The assessment was conducted by two engineers over the course of 16 calendar days.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Jade Han
Engineer
hojung@zellic.io ↗

Allen Roh
Engineer
allen@kalos.xyz ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

August 5, 2024	Start of primary review period
August 7, 2024	Kick-off call
August 26, 2024	End of primary review period

3. Detailed Findings

3.1. Allowing bypass of proof verification via unrestricted `skipedSubjectPrefix`

Target	EmailAuth.sol		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The vulnerability lies in the `authEmail` function of the EmailAuth contract, where an attacker can exploit the way certain parameters are processed. When a victim executes their transaction, certain parameters such as `expectedSubject`, `proof.maskedSubject`, `trimmedMaskedSubject`, and `proof.skipedSubjectPrefix` are set.

For example, the `expectedSubject` might be "A B C D", while `proof.maskedSubject` could be "A B C D", and the corresponding `trimmedMaskedSubject` remains "A B C D" (`proof.skipedSubjectPrefix` is zero in this case). Also, the Verifier contract's `verifyEmailProof` function processes these values, packing the `stringFields` with "A B C D" and additional zero padding, resulting in a total length of 605 bytes.

The attacker, aiming to alter the `expectedSubject` to something like "E F G H", can craft their input such that length of `proof.maskedSubject` extends beyond the 605 length, appending "E F G H" after the zero padding. By setting `proof.skipedSubjectPrefix` to 605, the system trims the `stringFields` to "A B C D" + "\x00"*(605-len("A B C D")) in the attacker's context.

In summary, this means that a proof generated for a subject of "A B C D" can be used to execute a transaction where the subject is "E F G H".

Impact

An identified vulnerability in the EmailAuth contract allows attackers to perform a front-running attack by exploiting a victim's transaction. By copying and manipulating specific fields within the transaction, the attacker can change the ownership of a contract or asset to themselves. This poses a severe risk, potentially leading to unauthorized transfers of valuable assets or control over contracts.

Recommendations

To prevent the described attack, the contract should enforce the following constraint.

- **Constrain `maskedSubject` length**: Ensure that the `proof.maskedSubject` is less than or equal to 605 bytes. This will prevent any excess data from being included in the proof.

Remediation

This issue was fixed by the ZK Email Team in commit 1455cd22 ↗.

3.2. Regular expression flaw related to email in circuit

Target	email_auth_template.circom		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In email_auth_template.circom, when checking the email_domain_regex.circom that it references, the regular expression used to extract the email is as follows:

```
[A-Za-z0-9!#$%&'*+=?\^_`{|}~.]+@[A-Za-z0-9.-]+
```

The above regular expression can extract most email addresses. However, according to this section on Wikipedia ↗, there are examples of valid email addresses that this regular expression cannot handle.

- "victim@gmail.com"@spam.co.kr
- /victim@gmail.com/@spam.co.kr

Impact

By spoofing the email sender, an attacker can generate a false proof, which could be exploited to take ownership of the user's wallet or steal their assets.

Recommendations

To resolve this issue, the circuit code should be modified to use the following regular expression:

```
[A-Za-z0-9!#$%&'"/*+=?\^_`{|}~.]+@[A-Za-z0-9.-]+
```

Remediation

This issue was fixed by the ZK Email Team in commit f71b30bd ↗.

3.3. Incorrect public input range check

Target	Groth16Verifier.sol		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In a ZKP verifier, it is customary to add a range check on each public input — checking that they are less than `p`, the prime field of the arithmetization. This is due to the fact that `x` and `x + p` may be different as integers (or in `uint256`), but they are equal in `Fp`. To prevent using `x + p` in the place of `x` in critical values like timestamp or nullifiers, a range check is used. Indeed, timestamp and email nullifiers are public inputs in the circuit, and they have relevant logic on the contract side as well.

```
function authEmail(EmailAuthMsg memory emailAuthMsg) public onlyController {
    require(
        timestampCheckEnabled == false ||
            emailAuthMsg.proof.timestamp == 0 ||
            emailAuthMsg.proof.timestamp > lastTimestamp,
        "invalid timestamp"
    );

    // ....

    usedNullifiers[emailAuthMsg.proof.emailNullifier] = true;
    lastTimestamp = emailAuthMsg.proof.timestamp;
}
```

In the Groth16Verifier, an automatically generated verifier contract, the check is performed but implemented incorrectly.

```
// Scalar field size
uint256 constant r = 21888242871839275 ... 575808495617;
// Base field size
uint256 constant q = 21888242871839275 ... 645226208583;

// ...
function checkField(v) {
    if iszero(lt(v, q)) {
        mstore(0, 0)
        return(0, 0x20)
```

```
    }
}
```

Here, the field check is done against `q`, which is the base field of BN254. The correct check is against `r`, the scalar field of BN254. As `q` is larger than `r`, this allows an attacker to carry out an attack — for example, a nullifier can be used twice.

We note that this may be a result of using an older version of snarkjs, as suggested by this commit ↗ inside snarkjs.

Impact

Incorrect usage of the timestamp and nullifier can be carried out in the contract, such as double usage of a nullifier.

Recommendations

Update snarkjs to the recent versions to fix the incorrect prime field range check.

Remediation

This issue was fixed by the ZK Email Team in commit e84065db ↗, by updating the snarkjs version.

3.4. The zk-regex audit fixes are not incorporated

Target	invitation_code_regex.circom		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The regex circuits in ether-email-auth are derived from the ZK Regex library, which automatically generates a corresponding circom circuit. The regex library was previously audited by a different audit firm, and this audit assumes that the ZK Regex library is correct.

However, we note that the version of ZK Regex library used to generate the circom circuits in ether-email-auth is not up to date, especially with the previous audit fixes.

For example, in ether-email-auth, the `is_consecutive` signal is initialized to `1`.

```
signal is_consecutive[msg_bytes+1][3];
is_consecutive[msg_bytes][2] <== 1;
```

However, this is incorrect, and this was fixed in the ZK Regex audit, as shown in this commit ↗.

```
signal is_consecutive[msg_bytes+1][3];
is_consecutive[msg_bytes][2] <== 0;
```

Indeed, this does change the revealed regex results in the invitation code with prefix regex.

In general, we advise the team to use the final updated version of ZK Regex to generate the relevant circom circuits for ether-email-auth.

Impact

The revealed regex result may be incorrect, resulting in incorrect masking of the subject.

Recommendations

Update the ZK Regex library used to generate the relevant circom circuits.

Remediation

This issue was fixed by the ZK Email Team in commit 9deaf804 ↗.

3.5. Indexes in circuits are not checked to be valid

Target	email_auth_template.circom		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

Most regex functionalities are out of scope, but they basically work as follows. First, regex circuits take a string as their input and return the string with only the public parts of the regex match revealed and the remaining parts masked with null bytes. Then, the `SelectRegexReveal` function is used to only take the matched parts. We can see this pattern used inside the circuit as follows.

```
// FROM HEADER REGEX
signal from_regex_out, from_regex_reveal[max_header_bytes];
(from_regex_out, from_regex_reveal)
    <== FromAddrRegex(max_header_bytes)(padded_header);
from_regex_out === 1;
signal from_email_addr[email_max_bytes];
from_email_addr <== SelectRegexReveal(max_header_bytes,
    email_max_bytes)(from_regex_reveal, from_addr_idx);
```

First, the from header regex is matched to the padded header with the `FromAddrRegex`. The `from_regex_reveal` is the result. Then, `SelectRegexReveal` is run on `from_regex_reveal` to take the matched part, which is `from_email_addr`.

The `SelectRegexReveal` function takes the masked string and the starting index. It also takes the reveal length as a fixed parameter. It then works as follows.

- It constrains that the byte just before the starting index is a null byte.
- It constrains that the starting index byte is nonzero.
- It constrains that indexes at least `startIndex + maxRevealLen` are all null bytes.
- It then rotates the array to the left by `startIndex`, then takes the first `maxRevealLen` bytes.

However, all of the main checks can be bypassed by having a large `startIndex` — larger than the array length.

Indeed, the documentation of `SelectRegexReveal` mentions that it assumes a valid index was taken as input. However, there are no such corresponding checks inside email_auth_template.circom. Due to this, one can output incorrect results of `SelectRegexReveal`.

Impact

The results of `SelectRegexReveal` are underconstrained.

Recommendations

Add the constraint that each index is a valid index.

Remediation

This issue was fixed by the ZK Email Team in commit e4497357 ↗ and 0b21c2a4 ↗ by adding a range check. We note that the fix here is sufficient due to the additional range check on index inside `VarShiftLeft` in `zk-email-verify`.

3.6. Incorrect constraints in `HashSign`

Target	hash_sign.circom		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

To prevent a same set of signatures from being used twice, the email nullifier is computed as a double Poseidon hash of the signature. This email nullifier is one of the public values of the ZKP circuit, and in the contracts, it is verified that a nullifier will not be used twice.

To compute the Poseidon hash of the signature, two signature chunks are combined into a single field element, then the field elements are hashed via Poseidon. The circuit is as follows.

```
template HashSign(n,k) {
    signal input signature[k];

    signal output sign_hash;

    var k2_chunked_size = k >> 1;
    if(k % 2 == 1) {
        k2_chunked_size += 1;
    }
    signal output sign_ints[k2_chunked_size];

    for(var i = 0; i < k2_chunked_size; i++) {
        if(i==k2_chunked_size-1 && k2_chunked_size % 2 == 1) {
            sign_ints[i] <== signature[2*i];
        } else {
            sign_ints[i] <== signature[2*i] + (1<<n) * signature[2*i+1];
        }
    }
    sign_hash <== Poseidon(k2_chunked_size)(sign_ints);
}
```

Here, if `k`, the number of chunks, is odd, then one chunk will be left after we pair two chunks into a field element. This case is handled in

```
if(i==k2_chunked_size-1 && k2_chunked_size % 2 == 1) {
    sign_ints[i] <== signature[2*i];
```

```
}
```

However, the condition is incorrect, for values such as `k = 15` and `k2_chunked_size = 8`.

Thankfully, in the circuit `k = 17`, so `k2_chunked_size = 9`, which is also odd. This means that the circuit itself is currently safe. However, on different parameters, the circuit will not work correctly.

Impact

This issue can lead to a broken circuit on different parameters of `n` and `k`.

Recommendations

Fix the condition as follows.

```
if(i==k2_chunked_size-1 && k % 2 == 1) {
    sign_ints[i] <== signature[2*i];
}
```

Remediation

This issue was fixed by the ZK Email Team in commit 9deaf804 ↗.

3.7. The `lasttimestamp` update mechanism flaw in EmailAuth

Target	EmailAuth.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Based on the below code, if `emailAuthMsg.proof.timestamp` is zero, the `lastTimestamp` is overwritten with zero. This could potentially disrupt the logic that relies on the `emailAuthMsg.proof.timestamp` value, as the condition checking the timestamp may be affected by this overwrite.

```
function authEmail(EmailAuthMsg memory emailAuthMsg) public onlyController {
    //...
    require(
        timestampCheckEnabled == false ||
            emailAuthMsg.proof.timestamp == 0 ||
            emailAuthMsg.proof.timestamp > lastTimestamp,
        "invalid timestamp"
    );

    //...

    usedNullifiers[emailAuthMsg.proof.emailNullifier] = true;
    lastTimestamp = emailAuthMsg.proof.timestamp;
    emit EmailAuthed(
        emailAuthMsg.proof.emailNullifier,
        emailAuthMsg.proof.accountSalt,
        emailAuthMsg.proof.isCodeExist,
        emailAuthMsg.templateId
    );
}
```

Impact

The code intended to prevent the use of outdated proofs using `lastTimestamp` may not function correctly.

Recommendations

To resolve this issue, `lastTimestamp` should only be updated when `emailAuthMsg.proof.timestamp` is not zero.

Remediation

This issue was fixed by the ZK Email Team in commit 4e2f119a ↗.

3.8. Unnecessary complexity in `resetWhenDisabled` function implementation

Target	SafeEmailRecoveryModule.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `resetWhenDisabled` function in the smart contract currently uses the `account` parameter as follows:

```
function resetWhenDisabled(address account) external {
    if (account == address(0)) {
        revert InvalidAccount(account);
    }
    if (ISafe(account).isModuleEnabled(address(this)) == true) {
        revert ResetFailed(account);
    }
    deInitRecoveryModule();
}
```

The ZK Email team's intention was to allow any arbitrary relayer to call this function for a given `account` address in order to delete all configuration data associated with that `account` address. However, upon reviewing the code of the `deInitRecoveryModule` function, it becomes clear that it deletes the configuration data related to `msg.sender` rather than the specified `account` address. This behavior differs from what the ZK Email team intended.

```
function deInitRecoveryModule() internal onlyWhenNotRecovering {
    delete recoveryConfigs[msg.sender];
    delete recoveryRequests[msg.sender];

    removeAllGuardians(msg.sender);
    delete guardianConfigs[msg.sender];

    emit RecoveryDeInitialized(msg.sender);
}
```

Impact

The current implementation may introduce unnecessary complexity, but it does not pose a significant security risk.

Recommendations

It is recommended to create a new `deinitRecoveryModule` logic.

Remediation

This issue was fixed by the ZK Email Team in commit 003123cb ↗.

3.9. DKIM signatures before 2001 may break the timestamp logic

Target	email_auth_template.circom		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

In the circuits, the timestamp is derived by matching the `TimestampRegex` on the padded header, then converting the decimal digits to an integer. If the timestamp does not exist, then the timestamp is set to zero in the circuit. In the contracts, if `timestampCheckEnabled` is true, then it is guaranteed that the timestamp of each email proof is increasing, as shown below.

```
require(
    timestampCheckEnabled == false ||
        emailAuthMsg.proof.timestamp == 0 ||
        emailAuthMsg.proof.timestamp > lastTimestamp,
    "invalid timestamp"
);
```

To convert the digits to integers, the circuits take the 10 bytes from `TimestampRegex` match and `SelectRegexReveal`, then run the circuit `Digit2Int` on it. The circuit `Digit2Int` assumes that all inputs are within `ord('0')` and `ord('9')`, and it simply accumulates the digits to an integer.

```
/// @input in The input byte array; assumes elements are between 48 and 57
    (ASCII numbers)
/// @output out The output integer; assumes to fit in the field
template DigitBytesToInt(n) {
    signal input in[n];

    signal output out;

    signal sums[n+1];
    sums[0] <== 0;

    for(var i = 0; i < n; i++) {
        sums[i + 1] <== 10 * sums[i] + (in[i] - 48);
    }

    out <== sums[n];
}
```

However, this assumption does not become true if the matched timestamp is of length 9. In that case, the `timestamp_str`, the 10-byte result of `SelectRegexReveal`, would become `[9 byte timestamp]\x00`. This will be sent over to the `DigitBytesToInt`, which would accumulate this without further checks. For example, if the timestamp in the padded header was `10^9 - 1`, which is nine digits, then the actual timestamp result from the `DigitBytesToInt` would be `10 * (10^9 - 1) - 48`, which is a very large value, corresponding to some time in year 2286.

For this idea to work, one needs a DKIM signature with a timestamp of length 9. This means the signature should be generated before September 2001.

Impact

One could use an old signature and trick the verifier into thinking that it is a recent signature. Also, one could use an old signature to update the `lastTimestamp` in the contract into a very large value, which means that the contract will not be able to accept new email proofs for a very long time if `timestampCheckEnabled` is turned on. The exact impact is determined by the exact threat model of the team.

Recommendations

If the team decides that the attack idea is of notable severity, one can fix this by simply adding a digit check in `DigitBytesToInt`.

Note that this attack idea fails for the invitation code `Hex2Field` as it checks each byte is a correct hex.

Remediation

This issue was fixed by the ZK Email Team in commit b7fc2f6f ↗ by adding the check that each byte is a correct digit.

3.10. Invitation code and email address may overlap, leading to unexpected behavior

Target	email_auth_template.circom		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

To mask the subject bytes, one takes the full subject, matches `InvitationCodeWithPrefixRegex`, and matches `EmailAddrRegex`. Then, it subtracts the invitation code and email address from the subject bytes, then calls `Bytes2Ints` to make them field elements.

```
signal masked_subject_bytes[max_subject_bytes];
for(var i = 0; i < max_subject_bytes; i++) {
    masked_subject_bytes[i] <== subject_all[i] - removed_code[i] -
    removed_subject_email_addr[i];
}
masked_subject <== Bytes2Ints(max_subject_bytes)(masked_subject_bytes);
```

This means if the subject is of the form `[some bytes] [invitation code] [some bytes] [email address] [some bytes]`, the masked subject will be `[some bytes] [null bytes] [some bytes] [null bytes] [some bytes]`.

However, this idea fails if the invitation code and email address overlap. To be exact, if the email address contains a substring that matches the invitation code regex, we can see unexpected behavior. For example, if the email address is code1647@gmail.com, then "code1647" will also be matched in `InvitationCodeWithPrefixRegex`. Therefore, the `code1647` part will be subtracted twice.

Also, note that the `Bytes2Ints` circuit does not check that each input is a byte.

One could even try to use this idea to craft a subject that has the following properties.

- The subject itself looks normal.
- `Bytes2Ints(masked subject)` = `Bytes2Ints(dangerous masked subject)`.

However, this does not work, as the masked subject is still composed of values within `[-127, 127]` (+- printable ASCII). As we are in base 256, and each `Bytes2Ints` only accumulates at most 31 bytes, it can be shown that even in $\mathbb{F}_p$, no such “collisions" occur. Note that this means that if we subtract out more things (not just two regex matches) a phishing-like idea could work, as the masked subject can have values of a wider range.

Impact

The circuit does not work normally for email addresses with invitation code-like substrings.

Recommendations

Either concretely document this behavior, or add circuit logic to subtract values only once if matched both times.

Remediation

This issue was acknowledged, and additional documentation about this behavior was added in commit 0096e59b ↗.

3.11. Minor inaccuracies in documentation of `SelectRegexReveal`

Target	regex.circom (zk-email-verify)		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In `SelectRegexReveal`, there is a comment that the circuit checks that every byte that is before the reveal part is zero. This is not true.

```
// Assert value before startIndex is zero
// ZK-Regex circuit contstrains that every byte before the reveal part is zero
// This is assuming matched data doesn't contain 0 (null) byte
isStartIndex[i] * (1 - isPreviousZero[i]) === 0;
```

In reality, only the byte right before the reveal part is checked to be zero.

To be more exact, the main assumptions surrounding `SelectRegexReveal` are as follows, which should be documented.

- The regex is matched exactly once.
- The matched regex has a length of at most `maxRevealLen`.

Further commenting on behaviors of `SelectRegexReveal`, we note that technically there may be more than one regex match. In that case, more than one regex match may be returned as a result of `SelectRegexReveal`. Also, since the output of `SelectRegexReveal` is simply a shift of the original array by `startIndex`, we note that with inputs like `[match] [null bytes] [match]`, with `startIndex` being the first byte of the second match, it is possible that the return value of `startIndex` contains parts of the first match as well. Note that even with the circuit constraint and all values after `startIndex + maxRevealLen` being zero, this still works. This is because this check does not consider wraparounds inside the array when shifting it.

Impact

The incorrect circuit documentation may confuse users and developers.

Recommendations

Fix the incorrect documentation, and add explanation of `SelectRegexReveal`'s underlying assumptions.

Remediation

The ZK Email Team added further documentation in commit d718290d ↗.

3.12. Circuits could be further optimized

Target	email_auth_template.circom		
Category	Optimization	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

We observed some simple areas for optimization in the circuits.

```
(prefixed_code_regex_out, prefixed_code_regex_reveal)
    <== InvitationCodeWithPrefixRegex(max_subject_bytes)(subject_all);
is_code_exist <== IsZero()(prefixed_code_regex_out-1);
```

Since `prefixed_code_regex_out` is already known to be boolean, `is_code_exist` can simply be `prefixed_code_regex_out`.

This is also the case with `is_subject_email_addr_exist`.

```
signal code_consistency <== IsZero()(is_code_exist * (1 - code_regex_out));
code_consistency === 1;
```

One can replace this with `is_code_exist * (1 - code_regex_out) === 0`.

Impact

Circuit constraints could be improved further for performance.

Recommendations

We recommend to add these optimizations to the circuit to improve performance.

Remediation

The ZK Email Team added these optimizations in commit 9deaf804 ↗ and commit 8b63b004 ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Enhancement for selector filtering in UniversalEmailRecoveryModule and EmailRecoveryModule

A potential security concern has been discussed regarding the UniversalEmailRecoveryModule and EmailRecoveryModule contracts. These modules reference the `_execute` function from the ERC7579ExecutorBase contract, which internally invokes the `executeFromExecutor` function.

Initially, there was a concern that this function could allow the execution of functions related to `IERC7579Account.installModule`, `IERC7579Account.uninstallModule`, and `IERC7579Account.executeUserOp` due to the `onlyEntryPointOrSelf` modifier in the MSABasic contract. However, it was clarified that there is a mechanism in place that checks if the target is an installed validator within the recovery modules themselves, which should mitigate the risk of calling unintended functions in normal ERC-7579 accounts.

In the context of Safe7579, which is a set of contracts developed by the Safe and Rhinestone teams to make Safes compatible with 7579 modules, the Safe account itself can be classified as a validator. This situation may require additional consideration, as functions on the Safe/Safe7579 should be accessible under certain conditions.

To address the scenario where the Safe account is used as a validator, it may be prudent to implement additional checks in the selector-filtering mechanism. Specifically, when the contract to be recovered is the Safe (i.e., `validator == msg.sender`), only a specific subset of selectors (such as `addOwner` and `setThreshold`) should be whitelisted. This would help prevent unintended function executions while maintaining necessary functionality.

4.2. Test suite

While the overall test coverage quality was good, we found that tests for the revert conditions could be improved, particularly in the newly added functions after the commit hash change from 3ae87c9a ↗ to b5694a9e ↗.

These areas lack comprehensive testing for failure scenarios, which may lead to undetected issues in edge cases or improper error handling. We recommend increasing the depth of testing for revert conditions, especially in the newly introduced functions, to ensure robust handling of failure scenarios.

Building a robust test suite that includes thorough tests for both positive and negative outcomes has multiple benefits:

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).

- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

Although creating and maintaining tests can be time-consuming, the long-term benefit is undeniable. Tests give developers confidence in their own changes, helping ensure that refactors or even small fixes don't introduce unintended issues. This is especially useful for new developers or those returning to a project after some time. A well-designed test suite acts as a safety net, indicating that existing functionality is most likely unaffected by code changes.

4.3. Multichain replay issue

In a recent discussion, the ZK Email team and external reviewers debated the potential risk of cross-chain replay attacks when the EmailRecoveryModule is deployed with the same address across multiple blockchains. The concern raised was that a proof used on one chain (e.g., chain A) could be reused on another chain (e.g., chain B) due to the `account_salt` lacking chain-specific data like a chain ID. This could pose a security risk, particularly for applications using ether-email-auth for on-chain message authorization.

While the reviewers suggested that this could be a significant security vulnerability, the ZK Email team argued that, at least for account recovery, this does not represent a major attack vector. They emphasized that preventing such attacks should be handled at the application level, rather than by modifying the protocol itself. The team believes that including chain-specific data, such as a chain ID in the email subject, could degrade the user experience and should therefore remain optional.

This is the proposed approach:

- Application-level mitigation — The ZK Email team recommends that the responsibility for preventing cross-chain replay attacks should lie with application developers. By allowing developers to decide whether to include chain-specific data, the solution can be tailored to different security needs without sacrificing UX for all users.
- Enhanced documentation — To ensure that application developers are aware of the potential risks, the ZK Email team plans to improve the documentation. This will include detailed guidance on how to handle chain-specific data in emails to mitigate replay attacks. By doing so, they aim to ensure that developers are fully informed about the implications of their choices and can implement the necessary protections if needed.

By focusing on application-level controls and enhancing the documentation, the ZK Email team seeks to balance security with a positive user experience, while ensuring that developers are equipped to handle potential risks in their implementations.

5. 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.

5.1. Module: EmailAccountRecovery.sol

Function: `handleAcceptance(EmailAuthMsg emailAuthMsg, uint256 templateIdx)`

The function processes and validates an acceptance by a new guardian, deploying and initializing a new EmailAuth contract proxy if necessary, based on the provided email authentication message.

Inputs

- `emailAuthMsg`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: It must contain all necessary information for authentication and authorization, `emailAuthMsg.proof.isCodeExist` must be true, and it must pass verification in `guardianEmailAuth`, created using `emailAuthMsg.proof.accountSalt`.
  - **Impact**: The parameter represents the email auth message sent from the guardian.
- `templateIdx`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the same `emailAuthMsg.templateId` and `templateId` for the Accept request.
  - **Impact**: The parameter represents the index of the subject template for acceptance.

Branches and code coverage

Intended branches

- Extracts the recovered account from the email subject based on `emailAuthMsg.subjectParams` and `templateIdx`.
  - ☑ Test coverage
- Computes the guardian's address using the recovered account and `emailAuthMsg.proof.accountSalt`.
  - ☑ Test coverage
- Validates the template ID and whether `isCodeExist` is `true` in `emailAuthMsg.proof`.

- ☑ Test coverage
- Deploys a new EmailAuth contract as a proxy if the guardian's code length is zero and initializes it with necessary parameters.
  - ☑ Test coverage
- If the guardian already has an EmailAuth contract, ensures its controller is the current contract.
  - ☑ Test coverage
- Calls `authEmail` on the `guardianEmailAuth` contract to authenticate the email.
  - ☑ Test coverage
- Calls `acceptGuardian` with the guardian address, `templateIdx`, `emailAuthMsg.subjectParams`, and `emailAuthMsg.proof.emailNullifier`.
  - ☑ Test coverage

Negative behavior

- Revert if `recoveredAccount` is invalid or zero.
  - ☑ Negative test
- Revert if the current weight is larger than zero.
  - ☑ Negative test
- Revert if the computed template ID does not match `emailAuthMsg.templateId`.
  - ☐ Negative test
- Revert if `emailAuthMsg.proof.isCodeExist` is `false`.
  - ☐ Negative test
- Revert if the guardian's EmailAuth contract is not controlled by the current contract.
  - ☐ Negative test
- Revert if it fails to pass verification in the `authEmail` function of the `guardianEmailAuth` contract.
  - ☑ Negative test
- Revert if the guardian has already accepted and the setting is complete or if the guardian is in the process of recovery.
  - ☑ Negative test
- Revert if a recovery process is already in progress for the account.
  - ☑ Negative test
- Revert if the recovery process has not been activated for the account.
  - ☑ Negative test
- Revert if the guardian's status for the account is not `REQUESTED`.
  - ☑ Negative test

Function call analysis

- `computeEmailAuthAddress(recoveredAccount, emailAuthMsg.proof.accountSalt)`
  - **What is controllable?** `recoveredAccount` and `emailAuthMsg.proof.accountSalt`.

- **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?** The intended transaction may not be fully executed if a revert occurs in the transaction of an AA Wallet or Safe Wallet that uses this contract as a module.

- `guardianEmailAuth.initDKIMRegistry(dkim())`
  - **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?** If an arbitrary registry can be set, it would allow bypassing the checks for the domain and public key included in the proof.
- `guardianEmailAuth.initVerifier(verifier())`
  - **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?** If an arbitrary verifier can be set, it would allow completely bypassing the proof verification.
- `guardianEmailAuth.insertSubjectTemplate(computeAcceptanceTemplateId(idx), acceptanceSubjectTemplates()[idx])`
  - **What is controllable?** `idx`.
  - **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.
- `guardianEmailAuth.insertSubjectTemplate(computeRecoveryTemplateId(idx), recoverySubjectTemplates()[idx])`
  - **What is controllable?** `idx`.
  - **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.
- `guardianEmailAuth.authEmail(emailAuthMsg)`
  - **What is controllable?** `emailAuthMsg`.
  - **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?** If the verification can be unexpectedly bypassed, any input data could be used, putting the user's assets at risk.
- `acceptGuardian(guardian, templateIdx, emailAuthMsg.subjectParams, emailAuthMsg.proof.emailNullifier)`
  - **What is controllable?** `guardian`, `templateIdx`, `emailAuthMsg.subjectParams`, and `emailAuthMsg.proof.emailNullifier`.
  - **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: `handleRecovery(EmailAuthMsg emailAuthMsg, uint256 templateIdx)`

The function processes the recovery based on an email from the guardian, verifies the provided email auth message, and ensures the guardian's EmailAuth contract is appropriately authorized.

Inputs

- `emailAuthMsg`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must contain all necessary information for authentication, and authorization must contain all necessary information for authentication and authorization.
  - **Impact**: The parameter represents the email auth message sent from the guardian.
- `templateIdx`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the same `emailAuthMsg.templateId` and `templateId` for the Recovery request.
  - **Impact**: The parameter represents the index of the subject template for recovery.

Branches and code coverage

Intended branches

- Extracts the recovered account from the email subject parameters (`emailAuthMsg.subjectParams`) and template index (`templateIdx`).
  - ☑ Test coverage
- Computes the guardian's address using the recovered account and `emailAuthMsg.proof.accountSalt`.
  - ☑ Test coverage
- Computes the template ID using the `templateIdx` and specific encoding for the recovery process.
  - ☑ Test coverage
- Initializes the `guardianEmailAuth` contract with the guardian's address.
  - ☑ Test coverage
- Authenticates the email by successfully verifying the provided `emailAuthMsg` using `guardianEmailAuth.authEmail`.
  - ☑ Test coverage
- Calls `processRecovery` with the guardian address, `templateIdx`, `emailAu-`

thMsg.subjectParams, and emailAuthMsg.proof.emailNullifier.
- ☑ Test coverage

Negative behavior

- Revert if recoveredAccount is invalid or zero.
  - ☑ Negative test
- Revert if the guardian is not deployed (i.e., the code length of the guardian's address is zero).
  - ☑ Negative test
- Revert if the account is not activated.
  - ☑ Negative test
- Revert if the current weight does not meet the threshold.
  - ☑ Negative test
- Revert if the computed template ID does not match emailAuthMsg.templateId.
  - ☑ Negative test

Function call analysis

- guardianEmailAuth.authEmail(emailAuthMsg)
  - **What is controllable?** emailAuthMsg.
  - **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?** If the verification can be unexpectedly bypassed, any input data could be used, putting the user's assets at risk.
- processRecovery(guardian, templateIdx, emailAuthMsg.subjectParams, emailAuthMsg.proof.emailNullifier)
  - **What is controllable?** guardian, templateIdx, emailAuthMsg.subjectParams, and emailAuthMsg.proof.emailNullifier.
  - **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.

5.2. Module: EmailAuth.sol

Function: `authEmail(EmailAuthMsg emailAuthMsg)`

This function is responsible for verifying whether the EmailAuthMsg type parameter provided by the user contains valid data and proof.

Inputs

- emailAuthMsg

- **Control**: Fully controlled by the caller.
- **Constraints**: The `domainName` and `publicKeyHash` included in `emailAuthMsg.proof` must be preregistered in a separate registry, and the `emailNullifier` must not have been used previously. Additionally, the `accountSalt` should match the value predefined in the EmailAuth contract. If the `timestamp` is nonzero, it must be generated later than any previously used input data. Finally, these input data and `emailAuthMsg.proof.proof` must pass the verification code to confirm it as a valid ZK proof.
- **Impact**: This parameter required to verify the validity of the input data.

Branches and code coverage

Intended branches

- Marks the `emailAuthMsg.proof.emailNullifier` as used and updates the `lasttimestamp`.
  - ☑ Test coverage

Negative behavior

- Revert if the template specified by `emailAuthMsg.templateId` does not exist.
  - ☑ Negative test
- Revert if the DKIM public key hash is invalid for `emailAuthMsg.proof.domainName` and `emailAuthMsg.proof.publicKeyHash`.
  - ☑ Negative test
- Revert if `usedNullifiers[emailAuthMsg.proof.emailNullifier]` is already set to `true`.
  - ☑ Negative test
- Revert if `accountSalt` does not match `emailAuthMsg.proof.accountSalt`.
  - ☑ Negative test
- Revert if the timestamp specified by `emailAuthMsg.proof.timestamp` is invalid.
  - ☑ Negative test
- Revert if the constructed `expectedSubject` does not match `emailAuthMsg.proof.maskedSubject` after removing the prefix `emailAuthMsg.skipedSubjectPrefix`.
  - ☑ Negative test
- Revert if `verifier.verifyEmailProof(emailAuthMsg.proof)` returns `false`.
  - ☑ Negative test

Function call analysis

- `verifier.verifyEmailProof(emailAuthMsg.proof)`
  - **What is controllable?** `emailAuthMsg.proof`.
  - **If the return value is controllable, how is it used and how can it go wrong?** It is possible to pass verification using an input different from the one intended

by the user, potentially leading to the takeover of the user's wallet or the theft of their assets.

- **What happens if it reverts, reenters or does other unusual control flow?** If reentrancy were to occur, there is a risk that a previously used proof could be reused; however, there are no points where reentrancy can occur.

5.3. Module: EmailRecoveryFactory.sol

Function: `deployEmailRecoveryModule(bytes32 subjectHandlerSalt, bytes32 recoveryModuleSalt, bytes calldata subjectHandlerBytecode, address dkimRegistry, address validator, bytes4 functionSelector)` // Function prototype

The function deploys an EmailRecoveryModule along with its subject handler using provided deployment parameters.

Inputs

- `subjectHandlerSalt`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The parameter is used as salt for the deployment of the subject handler contract.
- `recoveryModuleSalt`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The parameter is used as salt for the deployment of the email recovery module contract.
- `subjectHandlerBytecode`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be valid bytecode for a contract.
  - **Impact**: The parameter contains the bytecode for the subject handler contract to be deployed.
- `dkimRegistry`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid address.
  - **Impact**: The parameter specifies the address of the DKIM registry.
- `validator`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid address.
  - **Impact**: The parameter specifies the address of the validator to be recovered.
- `functionSelector`

- **Control**: Fully controlled by the caller.
- **Constraints**: Must be a valid function selector.
- **Impact**: The parameter specifies the function selector for the recovery function to be called on the target validator.

Branches and code coverage (including function calls)

Intended branches

- Deploys the subject-handler contract using `subjectHandlerSalt` and `subjectHandlerBytecode` and initializes the `subjectHandler` address.
  - ☑ Test coverage
- Deploys the email-recovery–module contract using `recoveryModuleSalt` and initializes the `emailRecoveryModule` address with references to `verifier`, `dkimRegistry`, `emailAuthImpl`, `subjectHandler`, `validator`, and `functionSelector`.
  - ☑ Test coverage
- Emits the `EmailRecoveryModuleDeployed` event with `emailRecoveryModule`, `subjectHandler`, `validator`, and `functionSelector` as arguments.
  - ☑ Test coverage
- Returns the addresses of the deployed `emailRecoveryModule` and `subjectHandler`.
  - ☑ Test coverage

5.4. Module: EmailRecoveryManager.sol

Function: `cancelRecovery()`

The function cancels the recovery request for the caller's account by deleting the current recovery request associated with the caller's account.

Branches and code coverage

Intended branches

- Deletes the recovery request associated with the caller's account if a recovery process is ongoing.
  - ☑ Test coverage

Negative behavior

- Revert if there is no recovery process in progress for the caller's account.
  - ☑ Negative test

Function: `completeRecovery(address account, bytes calldata recoveryData)`

The function completes the recovery process for a given account by validating and executing the recovery request.

Inputs

- `account`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not be a zero address.
  - **Impact**: The parameter is the address of the account for which recovery is being completed.
- `recoveryData`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the case that the `keccak256` hash of `recoveryData` matches the prestored `recoveryRequest.recoveryDataHash` in storage.
  - **Impact**: The parameter includes the data required to recover the validator or account.

Branches and code coverage

Intended branches

- Retrieves the recovery request for the given `account`.
  - ☑ Test coverage
- Retrieves the guardian configuration and checks if a recovery is configured.
  - ☑ Test coverage
- Validates that the current approval weight meets the threshold.
  - ☑ Test coverage
- Checks that the recovery request is within the valid time range.
  - ☑ Test coverage
- Validates the integrity of the provided `recoveryData`.
  - ☑ Test coverage
- Deletes the recovery request for the account.
  - ☑ Test coverage
- Calls the `recover` function with the account and recovery data.
  - ☑ Test coverage

Negative behavior

- Revert if the account address is zero.
  - ☑ Negative test
- Revert if no recovery is configured.

- ☑ Negative test
- Revert if the current weight does not meet the threshold.
  - ☑ Negative test
- Revert if the account is not activated.
  - ☑ Negative test
- Revert if the necessary delay has not passed.
  - ☑ Negative test
- Revert if the recovery request has expired.
  - ☑ Negative test
- Revert if the recovery data hash is invalid.
  - ☑ Negative test

Function call analysis

- `recover(account, recoveryData)`
  - **What is controllable?** `account` and `recoveryData`.
  - **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?** If the user executes incorrect `recoveryData`, it could freeze the assets in the wallet or interfere with its proper functioning.

Function: `configureRecovery(address[] memory guardians, uint256[] memory weights, uint256 threshold, uint256 delay, uint256 expiry)`

The function configures recovery settings for the caller's account, including guardians, threshold, delay, and expiry.

Inputs

- `guardians`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must have the same length as `weights`. Each guardian address must not be zero or the caller's own address.
  - **Impact**: The parameter is an array of guardian addresses.
- `weights`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must have the same length as `guardians`, and each weight must be greater than zero.
  - **Impact**: The parameter is an array of weights corresponding to each guardian.
- `threshold`
  - **Control**: Fully controlled by the caller.

  - **Constraints**: Must be greater than zero and less than or equal to the total weight of the guardians.
  - **Impact**: The parameter sets the weight threshold required for recovery.
- `delay`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None specified.
  - **Impact**: The parameter defines the delay period before recovery can be executed.
- `expiry`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None specified.
  - **Impact**: The parameter sets the expiry time after which the recovery attempt becomes invalid.

Branches and code coverage

Intended branches

- Verifies that this is the initial setup by checking if the threshold is zero.
  - ☑ Test coverage
- Calls `setupGuardians()` to initialize the guardians, their weights, and the threshold. This includes setting up the guardians and verifying compliance with the constraints.
  - ☑ Test coverage
- Initializes the `RecoveryConfig` with delay and expiry and calls `updateRecoveryConfig()`.
  - ☑ Test coverage

Negative behavior

- Revert if the configuration has already been set (setup already called).
  - ☑ Negative test
- Revert if the lengths of `guardians` and `weights` arrays do not match while calling `setupGuardians`.
  - ☑ Negative test
- Revert if any guardian address is zero or the caller's own address while calling `_addGuardian` in `setupGuardians`.
  - ☑ Negative test
- Revert if any weight is zero while calling `_addGuardian` in `setupGuardians`.
  - ☑ Negative test
- Revert if the threshold is zero.
  - ☑ Negative test
- Revert if the threshold exceeds the total weight of guardians after initialization.
  - ☑ Negative test
- Revert if any guardian address is already a guardian while calling `_addGuardian` in `setupGuardians`.
  - ☑ Negative test

Function: `updateRecoveryConfig(RecoveryConfig memory recoveryConfig)`

The function updates and validates the recovery configuration for the caller's account.

Inputs

- `recoveryConfig`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: `delay` must not be greater than `expiry`, and `expiry - delay` must be no less than `MINIMUM_RECOVERY_WINDOW`.
  - **Impact**: The parameter sets the new recovery configuration parameters for the caller's account.

Branches and code coverage

Intended branches

- Obtains the caller's account address using `msg.sender`.
  - ☑ Test coverage
- Validates that the account has been previously configured by checking if the threshold is greater than zero.
  - ☑ Test coverage
- Validates that the delay is not greater than the expiry.
  - ☑ Test coverage
- Validates that the recovery window (`expiry - delay`) is at least `MINIMUM_RECOVERY_WINDOW`.
  - ☑ Test coverage
- Updates the recovery configuration for the caller's account in the `recoveryConfigs` mapping.
  - ☑ Test coverage

Negative behavior

- Revert if the account is not configured (threshold is zero).
  - ☑ Negative test
- Revert if the `delay` is greater than the `expiry` in `recoveryConfig`.
  - ☑ Negative test
- Revert if the recovery window (`expiry - delay`) is less than `MINIMUM_RECOVERY_WINDOW`.
  - ☑ Negative test
- Revert if a recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test

5.5. Module: EmailRecoveryModule.sol

Function: `onInstall(bytes calldata data)`

The function initializes the module with the provided threshold and guardians.

Inputs

- `data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The parameter is the encoded data required for recovery configuration.

Branches and code coverage

Intended branches

- Decodes the `data` parameter to extract `isInstalledContext`, `guardians`, `weights`, `threshold`, `delay`, and `expiry`.
  - ☑ Test coverage
- Verifies if the module is installed by checking the `validator` using the `isInstalledContext`.
  - ☑ Test coverage
- Calls `configureRecovery` with the decoded `guardians`, `weights`, `threshold`, `delay`, and `expiry`.
  - ☑ Test coverage

Negative behavior

- Revert if the `data` parameter is zero length.
  - ☑ Negative test
- Revert if the module's `validator` is invalid.
  - ☑ Negative test

Function call analysis

- `configureRecovery(guardians, weights, threshold, delay, expiry)`
  - **What is controllable?** `guardians`, `weights`, `threshold`, `delay`, and `expiry`.
  - **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: `onUninstall(bytes calldata /* data */)`

The function deinitializes the recovery configuration for the calling account.

Branches and code coverage

Intended branches

- Deletes the recovery configuration for the calling account.
  - ☑ Test coverage
- Deletes the recovery request for the calling account.
  - ☑ Test coverage
- Removes all guardians associated with the calling account.
  - ☑ Test coverage
- Deletes the guardian configuration for the calling account.
  - ☑ Test coverage

Negative behaviour

- Revert if recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test

5.6. Module: EmailRecoveryUniversalFactory.sol

Function: `deployUniversalEmailRecoveryModule(bytes32 subjectHandlerSalt, bytes32 recoveryModuleSalt, bytes calldata subjectHandlerBytecode, address dkimRegistry)`

The function deploys a universal email recovery module along with its subject handler.

Inputs

- `subjectHandlerSalt`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The parameter is used as the salt for the subject-handler deployment.
- `recoveryModuleSalt`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: N/A.
  - **Impact**: The parameter is used as the salt for the recovery-module deployment.
- `subjectHandlerBytecode`
  - **Control**: Fully controlled by the caller.

- **Constraints**: Must be valid bytecode for the subject-handler contract.
- **Impact**: The parameter contains the bytecode of the subject-handler contract.

- `dkimRegistry`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid address.
  - **Impact**: The parameter represents the address of the DKIM registry.

Branches and code coverage

Intended branches

- Deploys the subject handler using the provided `subjectHandlerSalt` and `subjectHandlerBytecode`.
  - ☑ Test coverage
- Deploys the email-recovery module using the provided `recoveryModuleSalt`, `verifier`, `dkimRegistry`, `emailAuthImpl`, and the deployed subject handler.
  - ☑ Test coverage
- Returns the addresses of the deployed email-recovery module and subject handler.
  - ☑ Test coverage

Negative behavior

- Revert if the subject-handler deployment fails.
  - ☐ Negative test
- Revert if the email-recovery module deployment fails.
  - ☐ Negative test

Function call analysis

- `Create2.deploy(0, subjectHandlerSalt, subjectHandlerBytecode)`
  - **What is controllable?** `subjectHandlerSalt` and `subjectHandlerBytecode`.
  - **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.
- `new UniversalEmailRecoveryModule{ salt: recoveryModuleSalt }(verifier, dkimRegistry, emailAuthImpl, subjectHandler)`
  - **What is controllable?** `recoveryModuleSalt` and `dkimRegistry`.
  - **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.

5.7. Module: GuardianManager.sol

Function: `addGuardian(address guardian, uint256 weight)`

The function adds a guardian for the caller's account with a specified weight.

Inputs

- `guardian`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not be a zero address or the address of the caller.
  - **Impact**: The parameter is the address of the guardian to be added.
- `weight`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be greater than zero.
  - **Impact**: The parameter represents the weight assigned to the guardian.

Branches and code coverage

Intended branches

- Checks if the initial setup has been called by verifying the threshold.
  - ☑ Test coverage
- Validates the guardian address and weight.
  - ☑ Test coverage
- Adds the guardian to the guardian's storage.
  - ☑ Test coverage
- Increments the guardian count and total weight in the guardian configuration.
  - ☑ Test coverage

Negative behavior

- Revert if recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test
- Revert if the threshold is zero, indicating setup has not been called.
  - ☑ Negative test
- Revert if the guardian address is zero or the caller's own address.
  - ☑ Negative test
- Revert if the weight is zero.
  - ☑ Negative test
- Revert if the address is already a guardian.
  - ☑ Negative test

Function: `changeThreshold(uint256 threshold)`

The function changes the threshold for guardian approvals for the caller's account.

Inputs

- `threshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be greater than zero and less than or equal to the total weight of the guardians.
  - **Impact**: The parameter sets the new threshold for guardian approvals.

Branches and code coverage

Intended branches

- Checks if the initial setup has been called by verifying the threshold.
  - ☑ Test coverage
- Validates that the new threshold is not greater than the total guardian weight.
  - ☑ Test coverage
- Validates that the new threshold is greater than zero.
  - ☑ Test coverage
- Updates the threshold in the guardian configuration for the caller's account.
  - ☑ Test coverage

Negative behavior

- Revert if the threshold is zero, indicating setup has not been called.
  - ☑ Negative test
- Revert if the new threshold is greater than the total guardian weight.
  - ☑ Negative test
- Revert if the new threshold is zero.
  - ☑ Negative test
- Revert if recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test

Function: `removeGuardian(address guardian)`

The function removes a guardian for the caller's account.

Inputs

- `guardian`

- **Control**: Fully controlled by the caller.
- **Constraints**: Must be an existing guardian address for the caller's account.
- **Impact**: The parameter is the address of the guardian to be removed.

Branches and code coverage

Intended branches

- Retrieves the current guardian configuration and storage for the caller's account.
  - ☑ Test coverage
- Removes the guardian from the guardian's storage.
  - ☑ Test coverage
- Validates that the new total weight of guardians does not fall below the threshold.
  - ☑ Test coverage
- Decrements the guardian count and total weight in the guardian configuration.
  - ☑ Test coverage
- Decreases the accepted weight if the guardian's status was `ACCEPTED`.
  - ☑ Test coverage

Negative behavior

- Revert if the guardian is not found in the storage.
  - ☑ Negative test
- Revert if the new total weight is less than the threshold.
  - ☑ Negative test
- Revert if recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test

5.8. Module: SafeEmailRecoveryModule.sol

Function: `resetWhenDisabled(address account)` // Function prototype

The function resets the guardian states for the account when the module is disabled.

Inputs

- `account`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a non-zero address and the module must not be enabled for the specified account.
  - **Impact**: The parameter specifies the account for which the guardian states will be reset.

Branches and code coverage

Intended branches

- Deletes the recovery configuration and pending recovery requests for the caller's account.
  - ☐ Test coverage
- Removes all guardian settings for the caller's account and deletes the guardian configurations.
  - ☐ Test coverage

Negative behavior

- Revert if the `account` is the zero address.
  - ☐ Negative test
- Revert if the module is currently enabled for the specified `account`.
  - ☐ Negative test

5.9. Module: UniversalEmailRecoveryModule.sol

Function: `allowValidatorRecovery(address validator, bytes memory isInstalledContext, bytes4 recoverySelector)` // Function prototype

The function allows a validator and function selector to be used for recovery.

Inputs

- `validator`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid module validator for the caller's account.
  - **Impact**: The parameter specifies the validator to be allowed for recovery.
- `isInstalledContext`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Additional context data required to verify if the module is installed.
  - **Impact**: The parameter provides additional context for module-installation validation.
- `recoverySelector`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not be an unsafe selector. Actual cases include `onInstall`, `onUninstall`, `execute`, `setFallbackHandler`, `setGuard`, or `bytes4(0)`.
  - **Impact**: The parameter specifies the function selector to allow for recovery.

Branches and code coverage

Intended branches

- Validates that the validator is indeed a module installed for the caller's account using `isInstalledContext`.
  - ☑ Test coverage
- Ensures that the total count of validators for the caller's account does not exceed `MAX_VALIDATORS`.
  - ☑ Test coverage
- Adds the validator to the `validators` list and increments the `validatorCount` for the caller's account.
  - ☑ Test coverage
- Maps the allowed recovery selector to the validator and caller's account in `allowedSelectors`.
  - ☑ Test coverage

Negative behavior

- Revert if the provided `validator` is not a valid module installed for the caller's account.
  - ☑ Negative test
- Revert if the total count of validators for the caller's account has already reached `MAX_VALIDATORS`.
  - ☑ Negative test
- Revert if the provided `recoverySelector` is unsafe. Actual cases include `onInstall`, `onUninstall`, `execute`, `setFallbackHandler`, `setGuard`, or `bytes4(0)`.
  - ☑ Negative test
- Revert if the recovery module for the caller's account is not initialized (due to the `onlyWhenInitialized` modifier).
  - ☑ Negative test

Function: `disallowValidatorRecovery(address validator, address prevValidator, bytes4 recoverySelector)` // Function prototype

The function disallows a validator and function selector that has been configured for recovery.

Inputs

- `validator`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be an existing validator in the caller's validator list.
  - **Impact**: The parameter specifies the validator to be disallowed for recovery.
- `prevValidator`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the previous validator in the validator's linked list.

- **Impact**: The parameter specifies the previous validator in the linked list for the one to be removed.

- `recoverySelector`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must match the selector previously allowed for the specified validator.
  - **Impact**: The parameter specifies the function selector to be disallowed for recovery.

Branches and code coverage

Intended branches

- Removes the validator from the `validators` list and decrements the `validatorCount` for the caller's account.
  - ☑ Test coverage
- Validates that the allowed selector for the specified validator matches the provided `recoverySelector`.
  - ☑ Test coverage
- Deletes the allowed selector mapping for the specified validator and the caller's account.
  - ☑ Test coverage

Negative behavior

- Revert if the allowed selector for the specified validator does not match the provided `recoverySelector`.
  - ☑ Negative test
- Revert if the recovery module for the caller's account is not initialized (due to the `onlyWhenInitialized` modifier).
  - ☑ Negative test

Function: `onInstall(bytes calldata data)`

The function handles the installation of the module with the provided configuration data.

Inputs

- `data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a nonzero length bytes array.
  - **Impact**: The parameter contains the configuration data for module installation.

Branches and code coverage

Intended branches

- Decodes the `data` to extract the configuration parameters and initializes the `validators` list for the caller's account.
  - ☑ Test coverage
- Calls `allowValidatorRecovery` to set up the initial validator and selector for recovery.
  - ☑ Test coverage
- Calls `configureRecovery` to set up the recovery configuration including `guardians`, `weights`, `threshold`, `delay`, and `expiry`.
  - ☑ Test coverage

Negative behavior

- Revert if the `data` is empty.
  - ☑ Negative test

Function: `onUninstall(bytes calldata data)`

The function handles the uninstallation of the module and clears the recovery configuration.

Inputs

- `data`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Unused parameter.
  - **Impact**: The parameter is ignored in this function.

Branches and code coverage

Intended branches

- Retrieves the list of allowed validators for the caller's account and clears their allowed selectors.
  - ☑ Test coverage
- Removes all validators from the `validators` list and resets the `validatorCount` for the caller's account.
  - ☑ Test coverage
- Deletes the recovery configuration and pending recovery requests for the caller's account.
  - ☑ Test coverage
- Removes all guardian settings for the caller's account.
  - ☑ Test coverage

Negative behavior

- Revert if recovery is in process for the account (due to the `onlyWhenNotRecovering` modifier).
  - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the mainnet.

During our assessment on the scoped ZK Email contracts, we discovered 12 findings. One critical issue was found. Four were of high impact, five were of low impact, and the remaining findings were informational in nature.

6.1. 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.