Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Mach34

zkemail.nr

Veridise Inc.

December 03, 2024

► **Prepared For:**

Mach34
https://mach34.space/

► **Prepared By:**

Tyler Diamond
Ian Neal
Benjamin Sepanski

► **Contact Us:**

contact@veridise.com

► **Version History:**

Dec. 03, 2024	V1
Nov. 26, 2024	Initial Draft

Contents

1 Executive Summary

From Nov. 11, 2024 to Nov. 25, 2024, Mach34 engaged Veridise to conduct a security assessment of their zkemail.nr. The security assessment covered the zero-knowledge circuits involved in validating the DKIM signature of an email. The circuit focuses on extracting the body hash of the email from the provided header and validating that a given RSA public key has signed the email. Veridise conducted the assessment over 6 person-weeks, with 3 security analysts reviewing the project over 2 weeks on commit `2f81196`. Due to the Noir zero knowledge language exposing many utilities to developers, Veridise engineers also investigated some of the utilities' implementations invoked by Mach34's circuit.

Project Summary. The security assessment covered a Noir implementation of zkemail*, as well as some associated TypeScript helper functions. The circuits use DKIM signatures as attestations to an email's validity, then extracts information about the email from the hashed headers (which also include a hash of the email body). This allows users to prove that they received an email from an entity associated with a certain DKIM key, then demonstrate application-specific properties hold over the email header or body without revealing sensitive information contained in the email (such as the sending/receiving parties or other sensitive information stored in the body of the email).

Code Assessment. The zkemail.nr developers provided the source code of the circuits for the code review. The source code appears to be mostly original code written by the zkemail.nr developers. It implements similar functionality to the original zkemail TypeScript library†. It contains some documentation in the form of READMEs and documentation comments on functions. To facilitate the Veridise security analysts understanding of the code, the zkemail.nr developers also wrote and shared documentation that provided more detail on the intentions of the project and design decisions they made.

The source code contained a test suite, which the Veridise security analysts noted provided both positive and negative tests for signature validation, and also tested the typescript library that generates inputs to the circuit.

Summary of Issues Detected. The security assessment uncovered 14 issues, 1 of which (V-ZEML-VUL-001) was assessed to be of high severity by the Veridise analysts. Specifically, a malicious user may be able to bypass the body hash parsing, allowing the user to insert a body hash under their control and thereby take control of the contents of the proven email message. The Veridise analysts also identified 1 medium-severity issue (V-ZEML-VUL-002), which allows attackers to manipulate the structure of the provided public key and potentially forge signatures, as well as 4 low-severity issues, 7 warnings, and 1 informational finding. The zkemail.nr developers have indicated an intent to fix these issues.

* `https://github.com/zkemail`

† `https://github.com/zkemail/zk-email-verify/tree/d0e6f7ff9bc5723dbb8f4607b9f9892177b6cafb/packages/helpers`

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve zkemail.nr.

Warnings and unconstrained functions. Resolve compiler warnings so that newly introduced issues can be identified easier. Additionally, wrap all `unconstrained` calls in `unsafe` blocks. Finally, take care to verify if a function called from the standard library is unsafe before using it (see related issue V-ZEML-VUL-002).

Avoiding type-casting. Casting to `u32` (or other sized types) can be a dangerous operation, as it may silently truncate bits. When a value is expected to be of a certain size, consider constraining it to be of that size rather than truncating it. This will decrease the attack surface of the protocol.

Full DKIM tag validation. As described in V-ZEML-VUL-013, several DKIM tags are ignored. This will allow validation of expired signatures, may cause issues for applications which receive simple-canonicalized headers, or could lead to compatibility problems if future versions of DKIM are released. While these are unlikely to be common errors, full spec compliance is important for trusted adoption of this library. All spec deviations should be carefully noted in the documentation. For example, a quoted-printable encoding is assumed, but not checked for.

Redacted publication over private computation. When writing applications using the zkemail library, the Veridise analysts recommend publishing redacted emails rather than proving statements about fully private emails. This allows for human intervention and review, without sacrificing the requirements privacy or authentication.

Using zkemail for financial applications requires a high degree of trust in the proper interpretation of an email. There are many potential threats to applications relying on this library. For example, DKIM keys are commonly associated with multiple entities from an organization. Automated accounts frequently use the same DKIM key as customer support accounts, meaning that great care must be taken to ensure the sender of an email is indeed the intended sender. Additionally, email is frequently unstructured, and rarely conforms to a specific structure that can be relied upon like an API. Emails sent from the same entity may vary based on a user account's settings/preferences conveyed to the entities (such as the account's time zone, language, or currency), or may change without notice at entity's discretion.

These various factors may lead to increased efforts in phishing to receive spurious emails signed by the correct DKIM keys, black-box testing configurations to find a vulnerable configuration, or other more advanced attacks such as compromising target email accounts. Keeping emails fully private, and only revealing application-specific information, will increase the attack surface and make attacks more difficult to identify.

Hard-coded constants. Consider defining constants for ASCII characters rather than using hard-coded hex-code. This will increase readability, and reduce the load of maintaining the repository.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
zkemail.nr	`2f81196`	Noir	Aztec

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Nov. 11–Nov. 25, 2024	Manual	3	6 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	0	0
Medium-Severity Issues	1	0	0
Low-Severity Issues	4	0	0
Warning-Severity Issues	7	0	0
Informational-Severity Issues	1	0	0
TOTAL	14	0	0

Table 2.4: Category Breakdown.

Name	Number
Data Validation	5
Maintainability	4
Logic Error	1
Under-Constrained Circuit	1
Usability Issue	1
Information Leakage	1
Constraint Optimization	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of zkemail.nr's circuits. During the assessment, the security analysts aimed to answer questions such as:

- Is the project compliant with common mail transfer protocols like RFCs 5321, 5322, and related superseded RFCs?
- Can valid emails be rejected by the protocol?
- Is the partial and interstitial SHA256 implemented correctly?
- Is the DKIM signature verification correctly implemented?
- Is each relevant tag from the DKIM signature properly parsed and validated?
- Are header fields properly validated and guaranteed to be what was signed?
- How are duplicate header fields handled?
- Are any common zero-knowledge vulnerabilities such as missing state checks, incorrect control-flow handling, or privacy leakage present?
- Is email address parsing correctly implemented? Can extra data be injected into the email?
- What types of attacks could usage of this system lead to (e.g. phishing, privacy leaks, etc.)?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a thorough review by human experts.

Scope. The scope of this security assessment is limited to the `lib/src` and `js/src` folders of the source code provided by the zkemail.nr developers, which contains the zero-knowledge circuit implementation of the zkemail.nr along with the typescript libraries to generate inputs for the circuits.

Specifically, the following files were in scope of the review:

1. `js/src/`
    a) `index.ts`
    b) `utils.ts`
2. `lib/src/`
    a) `headers/body_hash.nr`
    b) `headers/email_address.nr`
    c) `headers/mod.nr`
    d) `dkim.nr`
    e) `lib.nr`
    f) `macro.nr`

g) `masking.nr`
h) `partial_hash.nr`
i) `remove_soft_line_breaks.nr`

Methodology. Veridise security analysts inspected the provided tests and read the zkemail.nr documentation. They then began a review of the code.

During the security assessment, the Veridise security analysts regularly met with the zkemail.nr developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ZEML-VUL-001	bh= may occur in other DKIM tags	High	Open
V-ZEML-VUL-002	redc field is manipulable	Medium	Open
V-ZEML-VUL-003	Multiple nullifiers can be generated for a . . .	Low	Open
V-ZEML-VUL-004	Un-normalized signature/DKIM keys	Low	Open
V-ZEML-VUL-005	Non-standard Email parsing	Low	Open
V-ZEML-VUL-006	First header value chars not validated	Low	Open
V-ZEML-VUL-007	Duplicate code	Warning	Open
V-ZEML-VUL-008	Unused program constructs	Warning	Open
V-ZEML-VUL-009	Missing/incorrect documentation	Warning	Open
V-ZEML-VUL-010	Elements may silently be skipped in hashing	Warning	Open
V-ZEML-VUL-011	Wrong value is compared to dkim header . . .	Warning	Open
V-ZEML-VUL-012	Email nullifiers may leak information on . . .	Warning	Open
V-ZEML-VUL-013	Ignored DKIM Tags	Warning	Open
V-ZEML-VUL-014	Optimization opportunities	Info	Open

4.1 Detailed Description of Issues

4.1.1 V-ZEML-VUL-001: bh= may occur in other DKIM tags

Severity	High	**Commit**	2f81196
Type	Data Validation	**Status**	Open
File(s)	`lib/src/headers/body_hash.nr`		
Location(s)	get_body_hash()		
Confirmed Fix At	N/A		

The `get_body_hash()` function obtains a sequence of length `BODY_HASH_BASE64_LENGTH` prefixed by a `bh=`, constrained to be within the `dkim-signature` header field. However, no check is performed to ensure that this sequence does not occur as part of another tag-value within the `dkim-signature` header-field.

Other tags may be user-controlled, leading to extraction of a user-controlled body-hash.

Impact If users create strange-looking email header field values (e.g. bh=7xQMDuoVVU4m0W0WRVSrVXMeGSIASsnucK9dJsrc+vU=@domain.com) or include the body-hash in the subject line when the mail-server is including the subject in the `z=` tag, the user may successfully be able to receive a `dkim-signature` header-value which contains *two* `bh=...` sequences: the actual body-hash and a fake body-hash.

Once a user has done this, they have full control over the contents in the proven message.

Recommendation Check that the extracted `bh=` tag is either the first tag, or preceded by a semicolon. See Section 3.2 in RFC 6376.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.2 V-ZEML-VUL-002: redc field is manipulable

Severity	Medium	**Commit**	2f81196
Type	Data Validation	**Status**	Open
File(s)	lib/src/dkim.nr		
Location(s)	verify_dkim_signature()		
Confirmed Fix At	N/A		

The `verify_dkim_signature()` function is a method attached to an `RSAPubkey` parameter representing the RSA public key to verify the DKIM signature against. This struct has 2 fields: the `modulus` and the `redc`. The latter parameter should have a specific relation to the `modulus` as defined here. As an example, Z-imburse's `verify_linode_billing_receipt()` passes in the `RSAPubkey` as is:

```
pub fn verify_linode_billing_receipt(
    header: BoundedVec<u8, MAX_LINODE_EMAIL_HEADER_LENGTH>,
    pubkey: RSAPubkey<KEY_LIMBS_2048>,
    signature: [Field; KEY_LIMBS_2048],
    ...
) -> [Field; 3] {
    ...
    pubkey.verify_dkim_signature(header, signature);
    ...
}
```

Snippet 4.1: Snippet from `verify_linode_billing_receipt()`

The only verification of the `pubkey`'s legitimacy is in the `z_imburse_registry`, which confirms the `modulus` hash is registered. There is no verification that the `redc` parameter has the proper relation to `modulus`. Therefore, an attacker may be able to manipulate the parameter in order to find data that a forged signature can be created for.

Impact An attacker may be able to forge signatures against data that was not legitimately signed. Due to time constraints, Veridise analysts did not fully exploit this attack. However, it very clearly causes the implementation to diverge from RSA, and likely may be converted into full signature forgery given an additional few days.

Recommendation Verify that the `redc` = `floor((1 << (2 * Params::modulus_bits())) / modulus)`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.3 V-ZEML-VUL-003: Multiple nullifiers can be generated for a signature

Severity	Low	**Commit**	2f81196
Type	Logic Error	**Status**	Open
File(s)	`lib/src/lib.nr`		
Location(s)	standard_outputs()		
Confirmed Fix At	N/A		

The `standard_outputs()` function calculates an `email_nullifier` as one of its outputs. It does so by simply taking the pedersen hash of the passed signature, as seen below:

```
let email_nullifier = pedersen_hash(signature);
```

Snippet 4.2: Snippet from `lib/src/lib.nr:standard_outputs()`

The `signature` is composed of `KEY_BYTE_LENGTH` `Fields` which should be represented by 120-bit limbs. However, given that the `Field` type is much larger, one can simply add the RSA public key modulus to the signature to generate a signature that is distinct in its hash, but represents the same signed message. Therefore, one can generate a large amount of signatures from a single legitimately signed message. A proof-of-concept is provided below

```
#[test]
fn test_dkim_signature_add() {
    let mut sig = EmailLarge::SIGNATURE;
    let pubkey = EmailLarge::PUBKEY;
    for i in 0..KEY_LIMBS_2048 {
        sig[i] += pubkey.modulus[i];
    }
    pubkey.verify_dkim_signature(EmailLarge::HEADER, sig);
}
```

Snippet 4.3: A proof-of-concept of duplicating a signature, ran in `lib/src/tests/mod.nr`

Impact If the `email_nullifier` is used as a nullifier in consumers of this library, then one can generate a plethora of nullifiers for the same email.

Recommendation Reduce the signature by the public key's modulus before hashing it.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.4 V-ZEML-VUL-004: Un-normalized signature/DKIM keys

Severity	Low	**Commit**	2f81196
Type	Under-Constrained Ci	**Status**	Open
File(s)	lib/src/dkim.nr		
Location(s)	verify_dkim_signature()		
Confirmed Fix At	N/A		

As mentioned in V-ZEML-VUL-003, `signatures` are represented as an array of limbs. The `verify_dkim_signature()` (shown below) operates on the `signature` represented as a `BigNum`.

```
// instantiate BN for the signature
let signature: BN1024 = BigNum::from_array(signature);

// verify the DKIM signature over the header
let rsa: RSA1024 = RSA {};
assert(rsa.verify_sha256_pkcs1v15(pubkey, header_hash, signature, RSA_EXPONENT));
```

Snippet 4.4: Snippet from `verify_dkim_signature()`.

The internal implementation of `BigNum` assumes the limbs have been range-checked to 120-bits so that no intermediate field multiplications inside the `BigNum.mul` methods used in `verify_sha256_pkcs1v15` overflow. However, this is never validated by the developers.

This means that attackers may produce additional signatures by using un-normalized representations of a DKIM signature. The below proof of concept can be generalized to a large class of alternate signatures

```
#[test]
fn test_dkim_signature_unnormalized() {
    let mut sig = EmailLarge::SIGNATURE;
    let pubkey = EmailLarge::PUBKEY;
    let delta = 1;
    sig[0] += delta * 0x100000000000000000000000000000 ;
    sig[1] -= delta;
    pubkey.verify_dkim_signature(EmailLarge::HEADER, sig);
}
```

Snippet 4.5: Adding and subtracting between limbs still produces a valid signature. This test can be run from `lib/src/tests/mod.nr`.

Impact From a single signature, attackers may produce multiple fresh signatures which pass verification. Since pedersen hashes are used to commit to signatures in the repository, each of these malformed signatures will produce a different commitment, appearing to be a new valid signature in cases where the signature is kept as a private witness.

Additionally, attackers may attempt to leverage overflows in the native Noir field during RSA verification. While the Veridise analysts were unable to exploit this in the time allotted, a dedicated attacker may be able to leverage the field overflows to produce valid signatures without knowledge of the private key.

Recommendation Constrain the signature to have normalized limbs.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.5 V-ZEML-VUL-005: Non-standard Email parsing

Severity	Low	**Commit**	2f81196
Type	Data Validation	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Email addresses are frequently thought of as a sequence of alpha-numeric characters (with a few allowed exceptions like `-`, `.`, and `_`) followed by an `@`, and then a domain. However, there are well-established standards for accepted forms of email addresses, including

1. The Address Specification in Section 3.4 of IETF RFC 5322 (superseding RFCs 2822 and 822).
2. IETF RFC 6531, which defines an extension for internationalized email.

Common implementations of email address parsers (e.g. implementations in Rust or python) expressly conform to one or more of the above RFCs. However, zk-email diverges in some cases from these specifications.

1. `js/src/utils.ts`:
   a) `getAddressHeaderSequence()`: This function restricts emails to include any sequence of "valid" characters (`[a-zA-Z0-9._%+-]`) followed by an `@`, and a similarly defined domain specification. This will wrongly accept some names (e.g. those with a repeated "." character) and wrongly deny others (e.g. those including a `#`).
   b) `getHeaderSequence()`: Some headers may appear more than once. See the table just above Section 3.6.1 in RFC 5322.
2. `lib/src/lib.nr`:
   a) Some commonly used email characters, such as an under-score, are disallowed.
3. `lib/src/headers/email_address.nr`:
   a) Strings without an `@` may be accepted as valid email addresses. In general, there is no guarantee that the email addresses are valid.

Impact Some library users may be unable to prove desired properties about valid emails.

Other use cases may be able to choose invalid emails which pass verification but break important invariants of an application.

Recommendation In all JavaScript/TypeScript code, consider using an established and well-used email library.

Explicitly choose a specification to conform to, or clearly indicate the deviations from the specification in the documentation. This may be especially important for users who need to ensure their email addresses are valid.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.6 V-ZEML-VUL-006: First header value chars not validated

Severity	Low	**Commit**	2f81196
Type	Data Validation	**Status**	Open
File(s)	`lib/src/headers/mod.nr`		
Location(s)	constrain_header_field()		
Confirmed Fix At	N/A		

The `constrain_header_field()` function checks that no `CRLF` sequences occur within the header value to prevent a malicious prover from including subsequent header field name/value pairs as part of the header value.

Unfortunately, as shown in the below snippet, this check is performed starting from `start_index+i`, where `i` starts from `HEADER_FIELD_NAME_LENGTH+1`. This means that the check is not applied to the field value characters at indices `HEADER_FIELD_NAME_LENGTH + 1`, ..., `2 * HEADER_FIELD_NAME_LENGTH + 1`. As a consequence, a malicious prover may be able to include extra data in a header-value if the header field name is longer than the header field value.

```
let start_index = header_field_sequence.index + HEADER_FIELD_NAME_LENGTH + 1;
for i in (HEADER_FIELD_NAME_LENGTH + 1)..MAX_HEADER_FIELD_LENGTH {
    // is it safe enough to cut this constraint cost in half by not checking lf? i
    think so
    let index = start_index + i;
    if (index < header_field_sequence.index + header_field_sequence.length) {
        assert(
            header.get_unchecked(index) != CR,
            "Header field must not contain newlines"
        );
    }
}
```

Snippet 4.6: Snippet from `constrain_header_field()`

Impact This can be very important to ensure that user-controlled inputs (such as the subject line, which may be manipulable through phishing) are not included in a header for later exploitation.

For example, if an application is extracting subject from the below email,

```
subject:t
from:longstring
```

an attacker could bypass the newlines check. Since `HEADER_FIELD_NAME_LENGTH=7` for subject, the below malicious input

```
index = 0
length = 10
```

leads the application to compute

```
start_index = 0 + 7 + 1 = 8
index = start_index + i = 8 + 8 = 16
```

which is the index of the colon in `from:longstring`. If the user can control `longstring`, they may then lie to the application by proving facts about `longstring` instead of the actual subject (`t`).

Luckily, this only works if the header value is shorter than the header field-name, which is not commonly the case for any of `from`, `to`, or `dkim-signature`.

Recommendation Loop over the range starting from `0`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.7 V-ZEML-VUL-007: Duplicate code

Severity	Warning	**Commit**	2f81196
Type	Maintainability	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Duplicate code or constraints were identified, and should be de-duplicated:

1. `lib/src/headers/mod.nr`:
    a) The `end_index` variable is defined as the addition of `index` and `length` of the `Sequence` instead of using the `end_index()` method.
        i. Note that there are a few locations where this is done.
    b) The `header_field_sequence` `index+length` is checked to be less than `header.len()`, and this check is repeated after `end_index` is defined.
    c) The `constrain_header_field_detect_last_angle_bracket()` and `constrain_header_field()` functions share most of their implementation. Consider refactoring out the common logic into an internal function used by both implementations.
2. `lib/src/remove_soft_line_breaks.nr`:
    a) In `find_zeros()` the `crate::CR` and `crate::LF` constants should be used instead of hard-coding the `0x0D` and `0x0A` values.
3. `lib/src/lib.nr`:
    a) The `verify_email()` function asserts `header.len() <= MAX_EMAIL_HEADER_LENGTH`. This is already implicitly constrained due to `header` being a `BoundedVec`.
4. `lib/src/partial_hash.nr`:
    a) The `partial_sha256_var_start`, `partial_sha256_var_interstitial`, and `partial_sha256_var_end` functions all duplicate a large amount of code. However, since this code is primarily copied from the Aztec standard library and de-duplicating this code would require significant refactoring, we do not find that de-duplicating code in these functions is strictly necessary.

Impact Duplicate code can make maintaining the codebase more difficult, and bugs may be introduced in the future if the implementations were to diverge. Additionally, duplicate constraints unnecessarily increase the circuit size.

Recommendation Refactor to remove the duplicated code.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.8 V-ZEML-VUL-008: Unused program constructs

Severity	Warning	**Commit**	2f81196
Type	Maintainability	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description The following program constructs are unused:

1. `lib/src/lib.nr`:
   a) The `standard_outputs()` function does not appear to be used outside of the examples.
2. `lib/src/partial_hash.nr`:
   a) In `msg_u8_to_u32()` the `msg32` array is hard-coded to be of length 16. This depends on `BLOCK_SIZE` always being 64. Instead of leaving the `BLOCK_SIZE` unused, meta-programming should be utilized to generate the length as `BLOCK_SIZE/4`, and assert the `BLOCK_SIZE` is a multiple of `4`. Similar comments apply to the following functions:
      i. `partial_sha256_var_end()`.
      ii. `verify_msg_block()`.
   b) The global `ZERO` defined in this file is unused.
   c) The global `DATA` should be defined within `mod tests`.

Impact These constructs may become out of sync with the rest of the project, leading to errors if used in the future.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.9 V-ZEML-VUL-009: Missing/incorrect documentation

Severity	Warning	**Commit**	2f81196
Type	Maintainability	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

The following items have incorrect documentation or are lacking documentation:

1. `js/src/util.ts`
    a) `makeEmailAddressCharTable()`:
        i. This function contains an extraneous `console.log()`.
        ii. Consider adding documentation noting that this function is used to generate Noir code, and link to the code location.
        iii. The variable `procedingChars` contains a minor typo. It should be defined as `proceedingChars`.
    b) `u8tToU32()`: Consider including the endianness of the output `U32`s in the documentation.
    c) The end of the file contains commented out dead code.
2. `lib/src/headers/body_hash.nr`:
    a) `get_body_hash_unsafe()`: Consider linking to the implementation of `base64_encode` as explanation for the counterintuitive function naming. For example, this link may be a useful reference for readers.
3. `lib/src/headers/mod.nr`:
    a) `constrain_header_field_detect_last_angle_bracket()`: The doc comment of this function has a typo, writing `contrain` instead of `constrain`.
4. `lib/src/lib.nr`:
    a) `MAX_EMAIL_ADDRESS_LENGTH`: Consider linking to RFC 5321 as an explanation for how this maximum length was derived.
5. `lib/src/partial_hash.nr`:
    a) `partial_sha256_var_start()` details that `N` is the maximum length of the message to hash instead of the actual length. Additionally, `message_size` does not exist for this function.

Impact Missing or incorrect documentation may lead to misuse of the library down the road.

Recommendation Fix the documentation for the above functions.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.10 V-ZEML-VUL-010: Elements may silently be skipped in hashing

Severity	Warning	**Commit**	2f81196
Type	Usability Issue	**Status**	Open
File(s)	`lib/src/partial_hash.nr`		
Location(s)	partial_sha256_var_start()		
Confirmed Fix At	N/A		

In `partial_sha256_var_start`, the `num_blocks` to hash is defined as `N / BLOCK_SIZE`. When `N % BLOCK_SIZE != 0`, the leftover bytes at the end of the `msg` are not mixed into the internal hashing state, and therefore are simply ignored. There is no indication that this is occurring.

Impact The outputted intermediate hash state will not contain the leftover bytes, and the caller of this function may expect that these bytes were included in the calculation. Given the intended usage of this function, any such errors are likely to be found by testing, but it should still be fixed.

Recommendation Fail if `N % BLOCK_SIZE != 0`, or make it explicitly clear that the caller must make this check.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.11 V-ZEML-VUL-011: Wrong value is compared to dkim header length

Severity	Warning	**Commit**	2f81196
Type	Maintainability	**Status**	Open
File(s)	`src/headers/body_hash.nr`		
Location(s)	get_body_hash()		
Confirmed Fix At	N/A		

In `get_body_hash()` the `body_hash_index` represents the location of where the body hash field body starts in the header. The header will be read starting at this value for `BODY_HASH_BASE64_LENGTH` bytes. In order to ensure that the bytes read are within the bound of the body hash header field, the `body_hash_index` is compared to the the total field sequence as seen below:

```
1 assert(
2     body_hash_index > dkim_header_field_sequence.index
3     & body_hash_index < dkim_header_field_sequence.end_index(), "Body hash index
      accessed outside of DKIM header field"
4 );
```

Snippet 4.7: Snippet from `lib/src/headers/body_hash.nr:get_body_hash()`

However, the second comparison should instead compare with `body_hash_index + BODY_HASH_BASE64_LENGTH`. The current comparison only ensures the start of the read is within the bounds, instead of the entire read existing within the bounds.

Impact Although one could not feasibly exploit the DKIM verification given the nature of the body hash field, users could insert the `bh=` string at an arbitrary point in the body hash in order to read past the intended field body. This may be an issues for consumers of the zkemail.nr library that want to stipulate about the body hash field of a header.

Recommendation Change the second constraint to check `body_hash_index + BODY_HASH_BASE64_LENGTH < dkim_header_field_sequence.end_index()`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.12 V-ZEML-VUL-012: Email nullifiers may leak information on the message

Severity	Warning	**Commit**	2f81196
Type	Information Leakage	**Status**	Open
File(s)	`lib/src/lib.nr`		
Location(s)	standard_outputs()		
Confirmed Fix At	N/A		

The `standard_outputs()` function will output the hash of the `pubkey` and a hash of the `signature`, in which the latter is intended to be used as a nullifier:

```
pub fn standard_outputs<let KEY_BYTE_LENGTH: u32>(
    pubkey: [Field; KEY_BYTE_LENGTH],
    signature: [Field; KEY_BYTE_LENGTH]
) -> [Field; 2] {
    let pubkey_hash = pedersen_hash(pubkey);
    let email_nullifier = pedersen_hash(signature);
    [pubkey_hash, email_nullifier]
}
```

Snippet 4.8: Snippet from `lib/src/lib.nr:standard_outputs()` with comments removed

Given that RSA signatures are not hiding, an information leakage may occur if the signature is leaked. If an attacker is able to find which public key signed the message (which can be done by attempting to verify the signature against a database of DKIM public keys), then the note emitted is known to have been related to this signing entity. Additionally, RSA signatures are deterministic and dependent upon the contents being signed. Therefore, if the attacker is aware of the structure of a message (such as in `z-imburse`), and they know a list of potential recipients, they could theoretically brute-force the contents of the message by finding the matching signature.

Impact Information leakage on the email which is nullified may occur if the signature is leaked out of band. Note that this function is not currently used anywhere except in examples.

Additionally, emails may commonly be resent (e.g. requesting a new copy of a receipt), which will likely cause the signature to change due to the new email timestamp. Consequently, the nullifier may not be effective for its intended purpose of preventing email replays.

Recommendation Add a deterministic nonce to the `email_nullifier` which is not dependent upon the `pubkey` or `signature`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.13 V-ZEML-VUL-013: Ignored DKIM Tags

Severity	Warning	**Commit**	2f81196
Type	Data Validation	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Several DKIM tags (see Sec 3.5 of RFC 6376) are ignored, with varying consequences.

1. The canonicalization tag `c=` is ignored. It is assumed that a relaxed canonicalization is used for the header.
2. The version tag `v=` is not checked for validity.
3. The body length tag `l=` is not used.
4. The expiration tag `x=` is ignored.
5. The SDID tag `d=` is ignored. Note that RFC 6376 dictates that applications are required to check this tag to distinguish between multiple entities using the same DKIM key.

Impact Assuming relaxed canonicalization or ignoring the `l=` tag may lead to denial-of-service issues.

Accepting unexpected versions or expired DKIM signatures may cause applications to accept invalid emails.

Recommendation Add documentation describing the ignored tags, and what additional checks a library user may need to perform.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.14 V-ZEML-VUL-014: Optimization opportunities

Severity	Info	**Commit**	2f81196
Type	Constraint Optimization	**Status**	Open
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

In the following locations, the auditors identified missed opportunities for constraint optimizations:

- `lib/src/remove_soft_line_breaks.nr`:
  - `find_zeroes()`: Since `is_break[N - 2]` will always be zero, the expression `should_zero[N - 1] = is_break[N - 2] + is_break[N - 3];` can be simplified to `should_zero[N - 1] = is_break[N - 3];`.

Impact Circuits may perform poorly or gas may be wasted, costing users extra time and funds.

Recommendation Perform the optimizations.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

Glossary

DKIM DomainKeys Identified Mail (DKIM) Signatures are commonly used to authenticate e-mail senders. See IETF RFC 6376 to learn more (https://datatracker.ietf.org/doc/html/rfc6376). 1

Noir A DSL for writing zero-knowledge circuits in a high-level, rust-like syntax. See https://noir-lang.org to learn more. 1

RSA A digital signature scheme. See https://datatracker.ietf.org/doc/html/rfc3447 to learn more. 1

zero-knowledge circuit A cryptographic construct that allows a prover to demonstrate to a verifier that a certain statement is true, without revealing any specific information about the statement itself. See https://en.wikipedia.org/wiki/Zero-knowledge_proof for more. 1