Security Audit Report

Celestia Q1 2023: NMT

Authors: Ivan Gavran, Andrija Mitrovic

Last revised 18 August, 2023

Table of Contents

Audit overview

The Project

In March 2023, Informal Systems has conducted a security audit for Celestia of their NMT (Namespaced Merkle Tree) library. The focus of the audit was 1) checking the documentation of the library, 2) checking the implementation and the usage within the NMT wrapper, and 3) delivering the formal model of the NMT proofs.

The audited commit hash is 341006e5.

The audit took place from March 1, 2023 through March 30, 2023 by the following personnel:

- Ivan Gavran
- Andrija Mitrovic

Conclusions

The audit consisted of proof-reading the documentation to make sure it is sound and complete, inspecting the code, and capturing a part of the NMT functionality with a formal model.

Overall, we found the library to be well developed and documented. During the audit process, we reported 20 findings: 1 of high severity, 5 of the severity, and the rest of low or informational severity. The issues were promptly resolved.

An additional outcome of the audit was the Quint model (link) from which we derived (model based) tests.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an “as is” basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bug free status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client’s business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.

Overview of the audit results

The audit process was comprised of three parts:

1. **Specification overview**
    - NMT spec
    - Wrapper spec
2. **Code review**
    - NMT spec
    - Wrapper spec
3. **Generation of formal specification using Quint**

Specification overview

- NMT specification overview:
    - Checking the alignment between the specification, underlying documents (Lazy Ledger) and the code comments
    - Checking the clarity of the documentation
    - Issues: #IF-41, #IF-40, #NMT-122, #NMT-133
- Wrapper specification overview
    - Checking the alignment between the specification and the code comments
    - Checking the clarity of the documentation

Code review

- Manual code review (with the help of tools like semgrep and CodeQl):
    - Best coding practices
        - Issues: #NMT-130, #NMT-131, #NMT-132, #NMT-139, #NMT-147
    - Inspecting code logic and alignment with specification and comments
        - Issues: #NMT-148, #NMT-121, #NMT-123, #NMT-154
    - Analyzing edge cases and searching for flaws
        - NMT functions
            - Empty tree case
            - Tree with one namespace (special case with ParityNamespace)
            - Tree with a different namespace at the very beginning or the very end (special case with ParityNamespace)
        - Proof functions
            - Empty proof
                - Issue: #NMT-140
    - Invalid data input
        - Corrupted namespace id and data
            - Wrong data or namespace id size
                - Issues: #NMT-144, #NMT-157
            - Wrong ordering of namespace id
                - Issue: #NMT-129
        - Corrupted proofs
            - Issue: #NMT-164
    - Review of tests
        - Analyzing coverage by test cases
            - Issues: #NMT-146, #NMT-158
        - Debugging tests

Formal Model in Quint

- Generating a formal model with Quint for the following NMT functionalities (PR)

    - Generating Inclusion Proofs
    - Verifying Inclusion Proofs
- Using the generated specification to create simulation tests (resulting in the mentioned issue #NMT-164)
    - Happy path tests
    - Edge cases tests

Findings

Title	Type	Severity	Status
Check of the namespace ordering missing in HashNode	Implementation	3 High	RESOLVED
IgnoreMaxNS leads to false computing maxNs	Implementation	2 Medium	RESOLVED
The function verifyLeafHashes will panic if called from VerifyInclusion function over an empty proof	Implementation	2 Medium	RESOLVED
Wrapper Push function will panic if the data is invalid	Implementation	2 Medium	RESOLVED
ValidateInclusion function can panic if it is called with an invalid nid	Implementation	2 Medium	RESOLVED
VerifyNamespace panics if called on an empty range, but with non-empty nodes	Implementation	2 Medium	RESOLVED
Left and right child swapped in the documentation	Documentation	1 Low	RESOLVED
Description of calculateAbsenceIndex function in code comments incorrect	Documentation	1 Low	RESOLVED
The assumption that leaves are ordered is not used consistently	Implementation	0 Informational	RESOLVED
Typo in the spec: end instead of end-1	Documentation	0 Informational	RESOLVED
The code-comment about the return value for ProveNamespace function does not correspond the code	Implementation	0 Informational	RESOLVED
Duplication of code in the HashNode function	Implementation	0 Informational	RESOLVED
Hasher initialization and reset should be done after validity checks	Practice	0 Informational	RESOLVED

Title	Type	Severity	Status
Overly specific helper functions that can be replaced by one generic function	Practice	0 Informational	RESOLVED
Unclear explanation of namespace calculation when the option IgnoreMaxNamespace is set	Documentation	0 Informational	RESOLVED
Computing leaf hashes should be done after the range check	Implementation	0 Informational	RESOLVED
Duplicated test cases	Practice	0 Informational	RESOLVED
Sanity check should be done immediately after tree creation	Practice	0 Informational	RESOLVED
Proof struct is missing a function for range verification	Practice	0 Informational	ACKNOWLEDGED
An incomplete test	Implementation	0 Informational	RESOLVED

Check of the namespace ordering missing in HashNode

Title	Check of the namespace ordering missing in HashNode
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	3 HIGH
Impact	2 MEDIUM
Exploitability	3 HIGH
Issue	https://github.com/celestiaorg/nmt/issues/129

Description

Function validateSiblingsNamespaceOrder) only checks if the maximum namespace id of the left child is less or equal to the minimum of the right minimum namespace id. It does not check the of leftMinNs, leftMaxNs, rightMinNs and rightMaxNs among children nodes.

Evan if the ordering of namespaces will guaranteed by the Push function it would be wise to check the whole ordering for the sake of completeness.

There is a testCase that submits children nodes with invalid namespace ordering to HashNode function. The left child node has minNs larger than the maxNs of the same node, and in addition to that leftMinNs is even larger then the rightMinNs. The test, with invalid data, will pass because `validateSiblingsNamespaceOrder` does not check the complete ordering of namespaces.

Problem Scenarios

This leads to wrong ordering of namespace ids.

Status

Resolved.

IgnoreMaxNS leads to false computing maxNs

Description

Title	IgnoreMaxNS leads to false computing maxNs
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	2 MEDIUM
Exploitability	3 HIGH
Issue	https://github.com/informalsystems/audit-celestia/issues/39

Description

The logic for determining the `maxNs` with `n.ignoreMaxNS` set to true, is closely dependent on the fact that if the one namespace ID (`minNs` or `maxNs`) is equal to the `n.precomputedMaxNs` then both of them are. The documentation is not clear enough on this logic, and this behavior makes the Namespaced Merkle Tree library dependant on the data input.

If the nmt is used with the data that does not follow the previously mentioned fact, there is an issue with computing `maxNs` here. This issue can lead to setting `maxNs` to `n.precomputedMaxNs` even if it could be set to a namespace ID that is smaller than `n.precomputedMaxNs` .

If we assume the following:

- `n.ignoreMaxNS = true`
- `leftMinNs != n.precomputedMaxNs`
- `rightMaxNs = n.precomputedMaxNs`
- `rightMinNs < n.precomputedMaxNs`
- `rightMinNs>leftMaxNs` .

Than the `maxNs` will be set to `n.precomputedMaxNs` even if it should be set to `rightMinNs` because it is the largest namespaceId smaller than `n.precomputedMaxNs` .

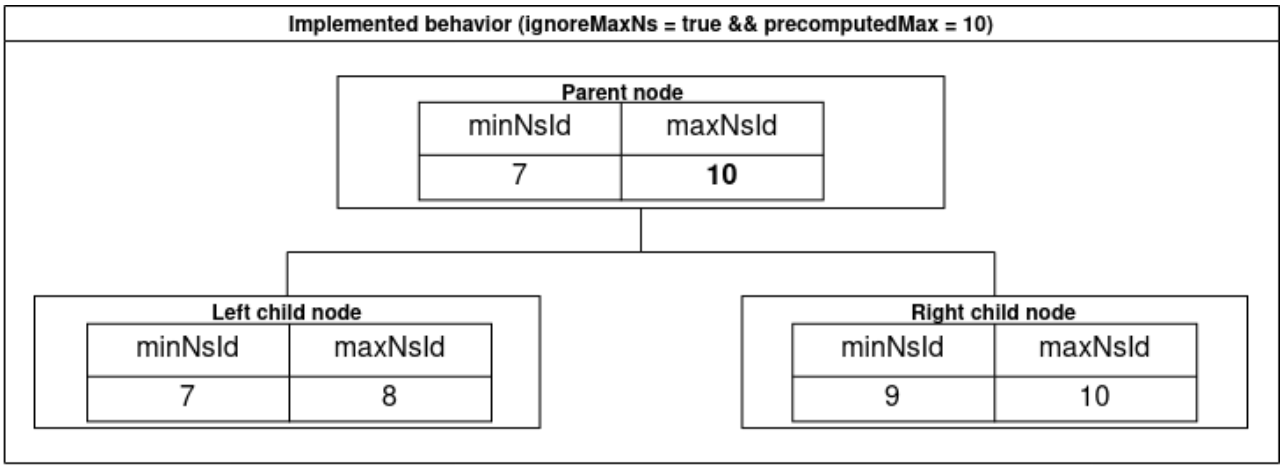
Example with numbers (presented also in the image at the end): Lets assume that following:

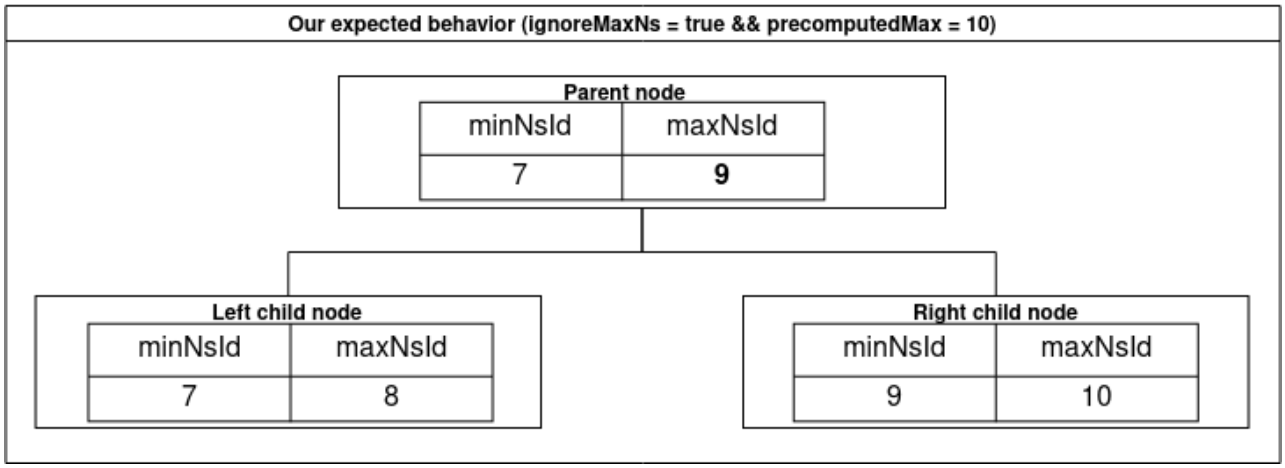
- precomputedMaxNs = 10
- ignoreMaxNS = true
- leftNode: left{minNs: 7; maxNs:8 }
- rightNode: right{minNs: 9; maxNs:10 }

Based on this logic in `HashNode` the `minNs` and `maxNs' for the node that is being calculated will take the following values:

- minNs = 7 (because it is the lowest value)
- maxNs = 10 or the `precomputedMaxNs` (because the `else` branch of the if statement will be executed and there 10 will obviously be chosen even if the `right.minNs` is larger then `left.maxNs` but not equal to `precomputedMaxNs`).

This numerical example with the current behavior and the behavior we expect is graphically presented in the following image.





Problem Scenarios

Wrong calculation of the value of `maxNS` .

Recommendation

We recommend to document well this behavior, and align the code with the documentation.

Status

Resolved.

The function verifyLeafHashes will panic if called from VerifyInclusion function over an empty proof

Title	The function verifyLeafHashes will panic if called from VerifyInclusion function over an empty proof
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Issue	https://github.com/celestiaorg/nmt/issues/140

Description

`verifyLeafHashes` function will panic when called over an empty proof. Panic will happen when `getSplitPoint(proof.end)` is called because an empty proof has proof.end = 0 that will lead to panic within `getSplitPoint` function here.

This can happen if the VerifyInclusion function is called over an empty proof. This function will not process the empty proof before calling `verifyLeafHashes`.

In contrast to when called from `VerifyInclusion`, `verifyLeafHashes` will not be called with an empty proof from the `VerifyNamespace` which handles empty proof before calling `verifyLeafHashes` (here and here).

Problem Scenarios

`verifyLeafHashes` function will panic when called over an empty proof.

Recommendation

`VerifyInclusion` should process the empty proof before calling `verifyLeafHashes`, but we suggest an additional protection in `verifyLeafHashes`.

Status

Resolved

Wrapper Push function will panic if the data is invalid

Title	Wrapper Push function will panic if the data is invalid
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Issue	https://github.com/celestiaorg/nmt/issues/144

Description

Push function from `ErasuredNamespacedMerkleTree` will panic if it is called with `data` whose size is smaller than the `NamespaceSize`. The panic will happen on this line. This is because the slice `data[:appconsts.NamespaceSize]` is out of bounds of the byte array.

Recommendation

Check `data` size before slicing.

Status

Resolved.

ValidateInclusion function can panic if it is called with an invalid nid

Title	ValidateInclusion function can panic if it is called with an invalid nid
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Issue	https://github.com/celestiaorg/nmt/issues/157

Description

If the ValidateInclusion function is called with an `nID` which size is not equal to the namespace size of the `proof.nodes`, the function could panic.

If this happened, the the nth hasher will be constructed with a different namespace size than the one that the proof was calculated with. This could lead to a panic in verifyLeafHashes function within the HashNode at this panic. Panic occurs because `validateSiblingsNamespaceOrder` could return an `ErrUnorderedSiblings` error because:

```
    leftMaxNs := namespace.ID(left[n.NamespaceLen:totalNamespaceLen])
    rightMinNs := namespace.ID(right[:n.NamespaceLen])
```

one of the `leftMaxNs` or `rightMinNs` is wrongly sliced due to different namespace sizes at their construction and thus parsed so that the following check is true and error is returned:

```
    if rightMinNs.Less(leftMaxNs) {
        return fmt.Errorf("%w: the maximum namespace of the left child %x is greater
than the min namespace of the right child %x", ErrUnorderedSiblings, leftMaxNs,
rightMinNs)
    }
```

Simple check like this one in the `VerifyNamespace` should be enough.

Problem Scenarios

Function `ValidateInclusion` could panic if called with invalid namespace id size.

Recommendation

Check if the passed `nid` parameter length is equal with the length on namespaces within the `root` parameter or the `proof.nodes` field.

Status

Resolved.

VerifyNamespace panics if called on an empty range, but with non-empty nodes

Title	VerifyNamespace panics if called on an empty range, but with non-empty nodes
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	2 MEDIUM
Impact	3 HIGH
Exploitability	1 LOW
Issue	https://github.com/celestiaorg/nmt/issues/164

Description

If `VerifyNamespace` is called with a proof that has an empty range (e.g., `start=end=0`), but non-empty `nodes`, it will panic. An example proof that would cause the panic is

```
customNode := []byte{7}
panickingProof := Proof{
    start:                  0,
    end:                    0,
    nodes:                  [][]byte{customNode},
    leafHash:               []byte{},
    isMaxNamespaceIDIgnored: true,
    }
```

`VerifyNamespace` does check for empty range, but only together with empty nodes. The panic happens in the getSplitPoint function.

Recommendation

`VerifyNamespace` should process the this edge case.

Status

Resolved.

Left and right child swapped in the documentation

Title	Left and right child swapped in the documentation
Project	Celestia Q1 2023: NMT
Type	DOCUMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/informalsystems/audit-celestia/issues/41

Description

In the nmt specification the names (`r` and `l`) of the children of the **Non-leaf Node** `n` have been switched when introducing them.

In the following code block:

```
**Non-leaf Nodes**: For an intermediary node `n` of the NMT with children

`r` = `l.minNs || l.maxNs || l.hash` and

`l` = `r.minNs || r.maxNs || r.hash`
```

`r` is used for the left node and the `l` is used for the right node, while it should be reversed.

Problem Scenarios

This issue affects the clarity of the specification.

Status

Resolved.

Description of calculateAbsenceIndex function in code comments incorrect

Title	Description of calculateAbsenceIndex function in code comments incorrect
Project	Celestia Q1 2023: NMT
Type	DOCUMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/informalsystems/audit-celestia/issues/40

Description

Description above the `calculateAbsenceIndex` function about index and leaf hash in case of absence proof is misaligned with the documentation and code.

Description states that the index of a leaf in the case of absence proof, should be the one with largest namespace ID smaller than the requested namespace, while the documentation states that the index should be the one with the smallest namespace ID that is larger than the requested namespace ID. The code is aligned with the documentation.

The same issue with the descriptions about index and leaf hash in case of absence proof was addressed trough this commit in two places to be aligned with the documentation, but was missed above before mentioned function.

Problem Scenarios

This issue affects the clarity of code.

Status

Resolved.

The assumption that leaves are ordered is not used consistently

Title	The assumption that leaves are ordered is not used consistently
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/121

Description

There is an assumption that leaves of NMTs are ordered by their namespace ids. However, this assumption is not used.

For instance in the specification:

```
NsH(n) = min(l.minNs, r.minNs) || max(l.maxNs, r.maxNs) || h(0x01, l, r)
```

, the specification takes min(l.minNs, r.minNs), where l.minNs would suffice.

Similarly, int the `HashNode` function same thing happens. The `minNs` is calculated like `minNs := min(leftMinNs, rightMinNs)` instead of `minNs := leftMinNs`.

But then on the other hand, the assumption on ordering namespace ids is used in the function CalculateAbsenceIndex, when finding the appropriate leaf for the absence proof.

Problem Scenarios

Differences between assumption, specifications and code can lead to harder understanding.

Recommendation

We recommend to use this assumption which would make the code and the spec clearer.

Status

Resolved.

Typo in the spec: end instead of end-1

Title	Typo in the spec: end instead of end-1
Project	Celestia Q1 2023: NMT
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/122

Description

In the Namespace Inclusion Proof section, two parts of the proof are mentioned. In part 2 it says:

> *In specific, the nodes include (...) 2) the namespaced hash of the right siblings of the Merkle inclusion proof of the* `end` *leaf*

but it should instead be

> *In specific, the nodes include (...) 2) the namespaced hash of the right siblings of the Merkle inclusion proof of the* `end-1` *leaf*

Status

Resolved

The code-comment about the return value for ProveNamespace function does not correspond the code

Title	The code-comment about the return value for ProveNamespace function does not correspond the code
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/123

Description

Comment above `ProveNamespace` function stated that an error is returned when the requested nID is not in the tree's range.

It appears that this is not aligned with the comment and the code in the `ProveNamespace` function for that specific case. These do not mention or return an error (nill is returned).

Problem Scenarios

This issue can cause wrong expectations about the signature of the function.

Status

Resolved.

Duplication of code in the HashNode function

Title	Duplication of code in the HashNode function
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/130

Description

There is a duplication of code in the `HashNode` function. Validity checks for internal node children are done within a function called ValidateNode. The same validity checks have been called individually here.So instead calling these validity checks one by one, `ValidateNode` should be called instead.

Status

Resolved.

Hasher initialization and reset should be done after validity checks

Title	Hasher initialization and reset should be done after validity checks
Project	Celestia Q1 2023: NMT
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/131

Description

This issue is a good coding practice. Hasher initialization and reset should be done after validity checks.

Problem Scenarios

Unnecessary resource allocation if the sanity check fails.

Status

Resolved.

Overly specific helper functions that can be replaced by one generic function

Title	Overly specific helper functions that can be replaced by one generic function
Project	Celestia Q1 2023: NMT
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/132

Description

One generic function could be made for generating nmt trees with different number of leaves and namespace id sizes. These two functions can be replaced by one generic function with a number of leaves as a function parameter, thus avoiding code duplication.

Problem Scenarios

Writing many functions that could be replaced by one generic function affects readability and maintainability.

Status

Resolved.

Unclear explanation of namespace calculation when the option IgnoreMaxNamespace is set

Title	Unclear explanation of namespace calculation when the option IgnoreMaxNamespace is set
Project	Celestia Q1 2023: NMT
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/133

Description

The section which explains how to calculate the namespace range when the parameterIgnoreMaxNamespace can be confusing. (here). In particular, the part

```
This is achieved by taking the maximum value among the namespace IDs available in the
range of node's left and right children (i.e., n.maxNs = max(l.minNs, l.maxNs ,
r.minNs, r.maxNs)), which is not equal to the maximum possible namespace ID value.
```

could be read as suggesting that the calculated n.maxNs is necessarily no equal to the maximum possible namespace ID value.

Status

Resolved.

Computing leaf hashes should be done after the range check

Title	Computing leaf hashes should be done after the range check
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/139

Description

computeLeafHashesIfNecessary should be done after the range check. This calculation could be time/resource consuming but unnecessarily called if the range is invalid.

Status

Resolved.

Duplicated test cases

Title	Duplicated test cases
Project	Celestia Q1 2023: NMT
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/146

Description

There are two testCases that only have different names (which are similar), but all the other parameters are the same.

Status

Resolved.

Sanity check should be done immediately after tree creation

Title	Sanity check should be done immediately after tree creation
Project	Celestia Q1 2023: NMT
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/147

Description

This sanity check comes too late. It should probably be done after the `tree` variable is created (here).

Problem Scenarios

Unnecessary resource allocation and computation if the sanity check fails.

Recommendation

First doing the sanity check, then if it passes allocate the needed resources.

Status Update

Resolved.

Proof struct is missing a function for range verification

Title	Proof struct is missing a function for range verification
Project	Celestia Q1 2023: NMT
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/154

Description

The `Proof` struct does not have a function for verifying a range of leaves that do not have to belong to the same namespace.

In the nmt.go there is a function ProveRange that does not depend on namespaceID and builds a proof for a given range. However, there is no its Verify- inverse function in proof.go. All verify functions (`VerifyNamespace` and `VerifyInclusion`) in proof.go depend on the namespaceID.

Function verifyLeafHashes could be used for proving a range of leaves that do not have to belong to the same namespace if its parameter `verifyCompleteness` is set to false (then its parameter `nID` would not be used, so it could be nil). This functions is used for internal purposes (called from `VerifyNamespace` and `VerifyInclusion`), so it could be as a called from the new function for verifying proof for a given range of leaves.

Problem Scenarios

Inability to verify a proof that does not depend on the namespace id.

Recommendation

Create a function for verification of ranged proofs.

Status Update

Acknowledged.

An incomplete test

Title	An incomplete test
Project	Celestia Q1 2023: NMT
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/nmt/issues/158

Description

TestNamespacedMerkleTree_calculateAbsenceIndex_Panic is incomplete. It does not push the data from test cases to the nmt. It passes (the tested function panics as expected) trivially because it tests an empty tree so this panic will be called in both cases. So this test does not test the given cases. Push function should be called for the given data.

One test case should added that tests the panic if the tree is empty, as it is done unintentionally in this test.

Status Update

Resolved.