



# **Security Audit Report**

Namada Multi-asset shielded pool

Authors: Manuel Bravo, Ivan Golubovic, Aleksandar Ljahovic

Last revised 26 February, 2024

# Table of Contents

<b>Audit Overview .....</b>	<b>1</b>
Scope .....	1
Conclusion .....	1
<b>Audit Dashboard .....</b>	<b>2</b>
Target Summary .....	2
Engagement Summary .....	2
Severity Summary .....	2
<b>System Overview .....</b>	<b>3</b>
Shielded transactions .....	3
Rewards and converts .....	4
On transaction validation and verification .....	4
<b>Threat Analysis .....</b>	<b>7</b>
Safety Properties .....	7
Liveness Properties .....	8
Other Threats .....	9
<b>Findings .....</b>	<b>10</b>
Fully transparent transaction can be marked valid in MASP validity predicate .....	11
Non-authorized users may render useless an arbitrary number of notes .....	13
Missing privacy checks in the masp validity predicate .....	15
Requiring that convert descriptions must point to the latest anchor is unnecessarily strict .....	16
The reward system is limited for NAM owners .....	17
Earlier rejection of transactions with invalid sapling value balance .....	19
Minor code improvements .....	21
<b>Appendix: Vulnerability Classification .....</b>	<b>22</b>
Impact Score .....	22
Exploitability Score .....	22
Severity Score .....	23
<b>Disclaimer .....</b>	<b>25</b>

# Audit Overview

## Scope

In January and February 2024, [Informal Systems](#) conducted a security audit for Heliix. The audit aimed at inspecting the correctness and security properties of Namada's multi-asset shielded pool (MASP) implementation, with a focus on the following components:

- The following files in <https://github.com/anoma/namada>
  - `crates/sdk/src/masp.rs`
  - `crates/core/src/types/masp.rs`
  - `crates/shielded_token/src/storage_keys.rs`
  - `crates/shielded_token/src/conversion.rs`
  - `crates/namada/src/ledger/native_vp/masp.rs`
  - `crates/apps/src/lib/node/ledger/shell/init_chain.rs`
  - `crates/apps/src/lib/node/ledger/shell/finalize_block.rs`
- Helper functions in <https://github.com/anoma/masp>

The audit was performed from January 18, 2024 to February 22, 2024 by the following personnel:

- Manuel Bravo
- Ivan Golubovic
- Aleksandar Ljahovic

## Relevant Code Commits

The audited code was from two repositories at the following commits:

- `anoma/namada` : hash `f7532c20072fd877046857f567735770389cd1d0`
- `anoma/masp` : hash `c3c6047a9c9da54058afc71219b913ac9f79e48b`

## Conclusion

We performed a thorough review of the project. We found some subtle problems - more on them in the section [Findings](#). Those problems, if left unattended, would violate both safety and liveness properties.

We are glad to report that the dev team has acknowledged our findings and is working towards fixing them.

# Audit Dashboard

## Target Summary

- **Type:** Protocol and Implementation
- **Platform:** Rust
- **Artifacts:** <https://github.com/anoma/namada>, <https://github.com/anoma/masp>

## Engagement Summary

- **Dates:** 18.01.2024 - 22.02.2024
- **Method:** Manual code review, protocol analysis

## Severity Summary

Finding Severity	#
Critical	2
High	0
Medium	3
Low	1
Informational	1
Total	7

## System Overview

Namada's multi-asset shielded pool (MASP) is a type of privacy-focused asset pool that allows for the shielding of multiple types of assets. It provides a way for users to pool their assets together in a way that hides the identities of the individuals involved and the amounts of the assets being traded. It is designed as an extension to the Sapling circuit which adds support for sending arbitrary assets.

Furthermore, MASP is built with mechanisms to align the incentives of users, rewarding those that contribute to shielding by adding assets to the pool.

The MASP system is built on top of Anoma's execution model, which implements a generic computational substrate with WASM-based transactions and validity predicate verification. A validity predicate (VP) is a boolean function that takes four inputs: (i) the transaction itself; (ii) the addresses that are involved with that specific VP; (iii) The storage state before a transaction execution; and (iv) the storage state after the transaction execution.

The execution system works as follows:

- When a block is decided by the consensus engine, transactions are executed tentatively. A transaction may modify any data in the accounts' dynamic storage sub-space.
- Upon transaction execution, the VPs associated with the accounts whose storage has been modified are invoked to verify the transaction.
- If any of them reject the transaction, all of its storage modifications are discarded.
- If all are accepted, the storage modifications are persisted.

## Shielded transactions

In Namada, the multi-asset shielded pool is a special Namada account (masp) with an associated validity predicate that handles the verification of shielded transactions, i.e., transactions that interact with the pool. There are three types of shielded transactions:

- A **fully shielded transaction** moves assets within the shielded pool.
- A **shielding transaction** moves assets from outside of the shielded pool to the shielded pool, i.e., shields assets.
- An **unshielding transaction** moves assets out of the shielded pool, i.e., unshields assets.

Throughout the report, we use transparent transactions to refer to those transactions that do not interact with the shielded pool: move assets from a transparent address to another transparent address.

Shielded transactions are implemented as an optional extension to transparent transfers. The transfer format is as follows:

```
pub struct Transfer {
  /// Source address will spend the tokens
  pub source: Address,
  /// Target address will receive the tokens
  pub target: Address,
  /// Token's address
  pub token: Address,
  /// The amount of tokens
  pub amount: DenominatedAmount,
  /// The unused storage location at which to place TxId
  pub key: Option<String>,
  /// Shielded transaction part
  pub shielded: Option<Hash>,
}
```

The optional `shielded` field in combination with the `source` and `target` field determines whether the transfer is fully shielded, shielding or unshielding.

- In a fully shielded transaction both the source and target addresses are the masp address.
- In a shielding transaction, only the target address is the masp address.
- In an unshielding transactions, only the source address is the masp address.

If it is a shielded transaction, through the `shielded` field one can retrieve a shielded transaction data that with the following format:

```
pub struct TransactionData<A: Authorization> {
    /// Transaction format version
    version: TxVersion,
    /// A globally-unique identifier for a set of
    /// consensus rules within the Zcash chain.
    consensus_branch_id: BranchId,
    lock_time: u32,
    /// Latest epoch at which the transaction can be executed
    expiry_height: BlockHeight,
    /// Transparent inputs and outputs
    transparent_bundle: Option<transparent::Bundle<A::TransparentAuth>>,
    /// Shielded inputs, converts (later introduced) and outputs
    sapling_bundle: Option<sapling::Bundle<A::SaplingAuth>>,
}
```

In more detail:

- `transparent_bundle` includes a set of transparent inputs descriptions ( `vin` ) and outputs descriptions ( `vout` ).
- `sapling_bundle` includes the set of shielded inputs (aka `shielded_spends` ), shielded outputs (aka `shielded_outputs` ), converts (aka `shielded_converts` ), which we explain later, and the `value_balance` , which informally is  $\text{sum}(\text{shielded\_inputs}) + \text{sum}(\text{converts}) - \text{sum}(\text{shielded\_outputs})$ .

## Rewards and converts

To incentivize the use of the multi-asset shielded pool, Namada rewards users who keep their assets shielded in the pool in Namada's native token (NAM). This is because the more volume of shielded transactions, the more serves its purpose: the more coverage for other users of the shielded pool; privacy is strengthened.

The set of assets that participate in the reward system is agreed upon via governance. Rewards are computed at the end of each epoch. This procedure consists of computing how many rewards an amount of a given asset type produces at a given epoch. This information is represented as conversions: an amount  $Y$  of an asset type  $A$  can be converted to the same amount of the same asset type plus  $X\text{NAM}$  ( $Y_A \rightarrow Y_A + X\text{NAM}$ ). For instance, if the protocol decides to give  $5\text{NAM}$  reward for each shielded BTC during epoch  $e$  , the protocol will create a conversion  $1\text{BTC}@e \rightarrow 1\text{BTC}@(e+1) + 5\text{NAM}(e+1)$ . Conversions are stored in a conversion Merkle Tree.

The set of conversions that the protocol creates at the end of each epoch defines the set of conversions that users can use to claim their rewards when executing transactions. To claim rewards, users must include convert descriptions in their `sapling_bundle` . Only convert descriptions that can be proved to exist in the conversion Merkle Tree are allowed by the protocol, i.e., a transaction that includes an invalid convert description is rejected.

## On transaction validation and verification

Shielded transactions are validated by the masp validity predicate. We now informally describe the conditions by which a shielded transaction must be accepted or rejected by the validity predicate.

## Shielding transactions

- The `source` address is transparent.
- The `target` address is the masp address.
- Shielded transactions well-formedness:
  - There are no `shielded_spends` in the transactions `sapling_bundle`.
  - There are no `shielded_converts` in the transactions `sapling_bundle`.
- Consistency between the `Transfer` record and the shielded transaction (from `Transfer.shielded`):
  - The address in each of the transparent inputs ( `transparent_bundle.vin` ) must match the `source` address in the transfer record.
  - The sum of all transparent inputs must match the amount in the transfer record.
  - The `sapling_bundle.value_balance` must match the amount in the transfer record.
  - The asset type of each transparent input must have the current epoch associated.
  - The token in the asset type of each transparent input must match the token in the transfer record.
- Changes to the masp state:
  - The `shielded_output` descriptions must be appended to the commitment tree.
  - No nullifiers must be revealed.
- It must pass the verification of `verify_shielded_tx`, which verifies among other things the integrity of value commitment, note commitment, and ephemeral public key of `shielded_outputs`, and that the `sapling_bundle.value_balance` is well computed.

## Unshielding transactions

- The `source` address is the masp address.
- The `target` address is transparent.
- Shielded transactions well-formedness:
  - There are no `shielded_outputs` in the transactions `sapling_bundle`.
  - The anchor of the `shielded_spends` must be valid.
  - The anchor of the `shielded_converts` must be valid.
- Consistency between the `Transfer` record and the shielded transaction (from `Transfer.shielded`):
  - The address in each of the transparent outputs ( `transparent_bundle.vout` ) must match the `target` address in the transfer record.
  - The sum of all transparent outputs must match the amount in the transfer record.
  - The `sapling_bundle.value_balance` must match the amount in the transfer record.
  - The asset type of each transparent output must have an epoch associated not greater than the current epoch.
  - The token in the asset type of each transparent output must match the token in the transfer record.
- Changes to the masp state:
  - Only the nullifiers associated with the `shielded_spends` must be revealed.
- It must pass the verification of `verify_shielded_tx`, which verifies among other things the integrity of value commitment, note commitment, and the nullifier of `shielded_spends`, and that the `sapling_bundle.value_balance` is well computed.

## Fully shielded transactions

- The `source` address is the masp address.
- The `target` address is the masp address.
- Shielded transaction well-formedness:
  - There are no transparent inputs or outputs.
  - The anchor of the `shielded_spends` must be valid.
  - The anchor of the `shielded_converts` must be valid.
- Changes to the masp state:
  - The `shielded_output` descriptions must be appended to the commitment tree.
  - Only the nullifiers associated with the `shielded_spends` must be revealed.
- Privacy:
  - The tokens in the transfer record must be the native one.
  - The transfer amount in the transfer record must be zero.
- It must pass the verification of `verify_shielded_tx`, which verifies among other things the integrity of value commitment, note commitment of `shielded_spends` and `shielded_outputs`, and that the `sapling_bundle.value_balance` is well computed.



## Threat Analysis

In our threat analysis, we start by defining a set of properties required for the correctness of MASP. We separate them into safety and liveness. We have then analyzed them individually to see if they can be violated. A property violation is a potential threat, as detailed for each property.

Additionally, we have inspected a set of threats related to integer-type conversions, integer overflow, and DoS attacks. We have listed them at the bottom of this section.

We have inspected the listed threats, resulting in the findings presented in the Findings section.

## Safety Properties

We list a total of 6 safety properties. Each of the properties has been verified assuming both malicious and well-behaving users.

### 1. Users cannot steal assets.

- Verify that a user can only spend notes if it is authorized.
- Verify that a user can only transfer assets to the shielded pool if it owns them.

**Threat:** Users can steal assets.

**Conclusion:** Under the assumption that a different validity predicate validates the ownership of assets for shielding transactions and that the `check_spend` function verifies that a user can only spend the notes it is authorized to, we conclude that the implementation prevents users from transferring assets or tokens they do not own.

### 2. Users cannot double spend: spend more than once the same set of assets.

- Verify that when a user spends a note, it reveals the corresponding nullifier.
- Verify that when a user attempts to spend a note, the validity predicate checks if its corresponding nullifier has already been revealed.

**Threat:** Users can double spend.

**Conclusion:** The implementation ensures the property.

### 3. Users cannot mint or burn assets: create or destroy assets.

- Verify that a shielded transaction achieves a net sapling value balance of 0.
- Verify that the sum of the net sapling value balance and transparent inputs of a shielding transaction is 0.
- Verify that the sum of the net sapling value balance and transparent outputs of an unshielding transaction is 0.
- Verify that a transaction only reveals the nullifiers of the notes it spends.

**Threat:** Users can create or destroy assets.

**Conclusion:** The implementation **does not** ensure the property. [This](#) finding describes a scenario in which users may reveal nullifiers of notes that they do not spend, effectively burning assets.

### 4. Users can only issue valid conversions.

- Verify that users can only issue conversions between assets in the allowed conversion set.

**Threat:** Users can issue non-allowed conversions by converting incompatible asset types or applying invalid ratios.

**Conclusion:** Under the assumption that `check_convert` verifies that the user can only apply allowed conversions, the implementation ensures the property.

## 5. The computation of allowed conversions at the end of each epoch is implemented as intended.

- Verify that the inflation computed is enough to guarantee that there are enough funds if all clients claim their rewards.
- Verify that the parallel computation of the conversions tree is correct.
- Verify that the computation of allowed conversions for the native token is as intended.
- Verify that the computation of allowed conversions for non-native tokens is as intended.
- Verify that the protocol does not create allowed conversions that allow users to mint or burn tokens.

**Threat(s):** Users can mint tokens by applying allowed conversions. The reward system is not working as expected.

**Conclusion:** The implementation ensures the property.

## 6. Validity: Only valid shielded transactions pass validation.

- Verify that an invalid shielded transaction fails validation.
- Verify that a transparent transaction either does not reach validation or fails it.

**Threat:** Invalid transactions pass validation.

**Conclusion:** The implementation **does not** ensure the property. [This](#) describes a scenario in which a transparent transaction passes validation.

## 7. Privacy: Fully shielded transactions are private.

- Verify that the validity predicate checks that fully shielded transactions are private: the transfer record uses the native token and the transfer amount is zero.

**Threat:** Users can observe the amount and tokens being transferred within the shielded pool.

**Conclusion:** The implementation **does not** ensure the property. [This](#) finding argues that privacy checks are missing in the validity predicate to reject fully shielded transactions that disclose either the token being transferred or the amount.

## Liveness Properties

### 8. No transaction makes validation crash the node.

- Verify that there is no way to crash a node during validation.

**Threat:** Users can crash the system by submitting a transaction that makes the masp validity predicate to panic.

**Conclusion:** The implementation ensures the property.

### 9. Assume that the chain is live. If a user submits a valid transaction, then the transaction passes validation and the chain state is eventually updated accordingly.

- Verify that if validation is called on a valid transaction, then the transaction passes validation.
- Verify that if a user submits a valid fully shielded or unshielding transaction in which it spends some unspent notes it owns, then the transaction passes validation.
- Verify that if a user submits a valid fully shielded or unshielding transaction with valid converts, then the transaction passes validation.

**Threat:** Users cannot interact with the shielded pool seamlessly, which may discourage them from using it.

**Conclusion:** The implementation **does not** ensure the property. [This](#) finding describes a scenario in which users may reveal nullifiers of notes that they do not spend. Thus, when a user attempts to spend its unspent notes, it may

be unable to. [This](#) other finding argues that the validity check for converts is too strict and may lead to fail valid transactions.

## Other Threats

**Threat:** Type conversions between different integer types are wrong

**Conclusion:** The implementation does not suffer from this issue.

**Threat:** Overflows and underflows due to inflation computations in the validator

**Conclusion:** The implementation does not suffer from this issue. We still highlight [here](#) a couple of computations in the code where we recommend checking for overflows - even if it is very unlikely.

**Threat:** Data structures in the shielded pool grow without bound by just executing valid shielded transactions. This may result in DoS attacks.

**Conclusion:** The implementation does not suffer from this issue.

**Threat:** Functions are non-deterministic due to iterations.

**Conclusion:** The implementation does not suffer from this issue.

**Threat:** Functions are non-deterministic due to reasons other than iterations.

**Conclusion:** The implementation does not suffer from this issue.

## Findings

Title	Project	Type	Severity	Impact	Exploitability	Status	Issue
Fully transparent transaction can be marked valid in MASP validity predicate	Anoma: MASP	IMPLEMENTATION	4 CRITICAL	3 HIGH	3 HIGH	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2594">https://github.com/anoma/namada/issues/2594</a>
Non-authorized users may render useless an arbitrary number of notes	Anoma: MASP	IMPLEMENTATION	4 CRITICAL	3 HIGH	3 HIGH	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2594">https://github.com/anoma/namada/issues/2594</a>
Missing privacy checks in the masp validity predicate	Anoma: MASP	IMPLEMENTATION	2 MEDIUM	3 HIGH	1 LOW	ACKNOWLEDGED	
Requiring that convert descriptions must point to the latest anchor is unnecessarily strict	Anoma: MASP	PROTOCOL IMPLEMENTATION	2 MEDIUM	3 HIGH	1 LOW	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2719">https://github.com/anoma/namada/issues/2719</a>
The reward system is limited for NAM owners	Anoma: MASP	PROTOCOL IMPLEMENTATION	2 MEDIUM	3 HIGH	1 LOW	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2720">https://github.com/anoma/namada/issues/2720</a>
Earlier rejection of transactions with invalid sapling value balance	Anoma: MASP	IMPLEMENTATION	1 LOW	1 LOW	1 LOW	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2721">https://github.com/anoma/namada/issues/2721</a>
Minor code improvements	Anoma: MASP	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	ACKNOWLEDGED	<a href="https://github.com/anoma/namada/issues/2722">https://github.com/anoma/namada/issues/2722</a>

## Fully transparent transaction can be marked valid in MASP validity predicate

<b>Title</b>	Fully transparent transaction can be marked valid in MASP validity predicate
<b>Project</b>	Anoma: MASP
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	4 CRITICAL
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	3 HIGH
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2594">https://github.com/anoma/namada/issues/2594</a>

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)

### Description

To assess the ability of the masp validity predicate to handle transparent transactions, we tried to simulate what checks would be triggered and whether they would prevent such a transaction from being marked valid.

We concluded that well-formed transparent transactions with at least one sapling output of value 0 will pass validation. This could be a security threat: transparent transactions that modify keys under the masp storage subspace and pass validation could change the masp internal state to a potentially wrong/inconsistent state. For instance, in the current implementation, the validity predicate does not guarantee that a transparent transaction does not reveal nullifiers. This is the only case we have found in the current implementation in which accepting as valid a transparent transaction would leave the masp internal state inconsistent. In [Non-authorized users may render useless an arbitrary number of notes](#) we discuss the negative consequences of allowing transactions to reveal an arbitrary number of nullifiers without authorization.

Note that even when a transaction does not write masp keys, it can explicitly request to be validated by the masp validity predicate. This means that someone can construct a transaction that does not modify the masp storage space but still requires the masp validity predicate to be executed. Nevertheless, it remains unclear whether allowing the execution of such transactions is harmful or not.

In any case, we recommend guaranteeing that the masp validity predicate rejects transparent transactions.

### Problem Scenarios

Consider the transaction with a non-empty sapling bundle consisting of a set of sapling outputs, each of value 0. Let  $x$  be the amount being transferred. The validity predicate will accept such a transaction:

- Since the source address isn't masp, the transaction enters the shielding transaction validation.
- It initializes `transparent_tx_pool` to 0 as the `sapling_value_balance` is 0.
- Since it has no spends or converts, it passes [this check](#).
- After validating all transparent inputs, `transparent_tx_pool=x`.
- It will now enter the validation for unshielding transactions.
- After validating all transparent outputs, `transparent_tx_pool=0`.
- It will then pass the [check for zero value balance](#), and enter verification.
- During verification, the transaction will pass the [non-empty sapling bundle check](#) because it has sapling outputs.
- It will also pass the `sapling_value_balance` check of the `final_check` function.

## Recommendation

We recommend adding a check to prevent transparent transactions from passing the masp validity predicate validation. It could consist of simply checking that the transaction should be rejected if both source and target addresses are transparent.

An alternative could be to completely decouple the masp state consistency checks from the type of shielded transaction being validated. This way, any transaction that passes the masp validity predicate validation is guaranteed to leave the masp state consistent.

## Non-authorized users may render useless an arbitrary number of notes

<b>Title</b>	Non-authorized users may render useless an arbitrary number of notes
<b>Project</b>	Anoma: MASP
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	4 CRITICAL
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	3 HIGH
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2594">https://github.com/anoma/namada/issues/2594</a>

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)

### Description

When a user spends a note, it must reveal the associated nullifier to avoid double-spending. In a fully shielded or unshielding transaction, the function `valid_nullifiers_reveal` checks that the nullifiers of every spent note are revealed and that the note has not been spent before. In addition, it checks that the transaction does not reveal any other nullifier for notes it has not spent. Nevertheless, the validity predicate does not do the latter check for shielding transactions. As a result, a shielding transaction is free to reveal an arbitrary number of nullifiers without being rejected.

This has multiple negative consequences, some of them being:

- A malicious user can burn assets within the pool: the maliciously nullified spends cannot be spent, but they have not been transferred or converted.
- A user loses assets.

### Problem Scenarios

- Assume that Alice has 5 BTC in the shielded pool that are registered as a single note with the associated nullifier `nul`.
- Assume that a second user Bob submits a valid shielding transaction that in addition reveals nullifier `nul`.
- Since it is a shielding transaction, the source address is not the masp. Thus, the `valid_nullifiers_reveal` function is not called [here](#).
- Thus, the transaction passes the validation and it is persisted.
- Assume further that Alice now wants to unshield her 5BTC by submitting an unshielding transaction.

- This time, the source address is not the masp and the `valid_nullifiers_reveal` function is called.
- The transaction would be rejected because the nullifier `nul` has already been revealed ([this](#) check)
- As a result, Alice has lost 5BTC and Bob has effectively burned 5BTC.

## Recommendation

We recommend adding a check similar to [this](#) for shielding transactions.



## Missing privacy checks in the masp validity predicate

<b>Title</b>	Missing privacy checks in the masp validity predicate
<b>Project</b>	Anoma: MASP
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	2 MEDIUM
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	1 LOW
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](https://github.com/Anoma/namada/blob/main/crates/namada/src/ledger/native_vp/masp.rs)

### Description

For privacy, fully shielded transactions generated via the `gen_shielded_tx` function don't disclose the amount or the token being transferred by setting the amount to 0 and the token to the native token in the associated transfer record. Nevertheless, there is no control over how users generate their fully shielded transactions. Thus, fully shielded transactions submitted by a malicious user may indeed disclose the amount or token being transferred in the transfer record. The masp validity predicate does not check that the transfer record associated with a fully shielded transaction always uses 0 as the amount and the native token as the token.

### Problem Scenarios

If malicious users that interact with the shielded pool coordinate to always reveal the amount and token being transferred, the pool will provide less coverage for other users; weakening privacy. This may discourage users from using the shielded pool.

### Recommendation

We recommend adding a check in the validity predicate to reject fully shielded transactions that disclose the amount or the token being transferred.

## Requiring that convert descriptions must point to the latest anchor is unnecessarily strict

<b>Title</b>	Requiring that convert descriptions must point to the latest anchor is unnecessarily strict
<b>Project</b>	Anoma: MASP
<b>Type</b>	PROTOCOL IMPLEMENTATION
<b>Severity</b>	2 MEDIUM
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	1 LOW
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2719">https://github.com/anoma/namada/issues/2719</a>

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)
- [crates/shielded\\_token/src/conversion.rs](#)

### Description

Shielded transactions that include convert descriptions pass validation only if all convert descriptions point to the latest convert anchor. We argue that this is too strict and could lead to the unnecessary rejection of valid shielded transactions. The risk is mitigated because the convert tree (and therefore the convert anchor) is only updated once per epoch.

### Problem Scenarios

An honest user may generate a shielded transaction with conversions at epoch  $e$ , such that all convert descriptions point to the convert anchor computed at the end of epoch  $e-1$ . Assume that the user submits the transaction. Assume further that the transaction is included in a block at epoch  $e+1$ . Then, the transaction fails validation ([here](#)) because the anchors of convert descriptions are not the current one.

### Recommendation

We recommend allowing convert descriptions to point to older anchors, similar to spend descriptions. This would require keeping in storage multiple convert anchors and possibly changing the way the protocol computes the convert tree.

## The reward system is limited for NAM owners

<b>Title</b>	The reward system is limited for NAM owners
<b>Project</b>	Anoma: MASP
<b>Type</b>	PROTOCOL IMPLEMENTATION
<b>Severity</b>	2 MEDIUM
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	1 LOW
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2720">https://github.com/anoma/namada/issues/2720</a>

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)
- [crates/shielded\\_token/src/conversion.rs](#)

### Description

Native token conversions have the following format  $-(X) \text{NAM}@e-1 + (X+Y) \text{NAM}@e$ . This means that users can convert  $X$  Namada tokens from epoch  $e-1$  into  $(X+Y)$  Namada tokens from epoch  $e$ . The protocol updates the set of allowed conversions at the end of each epoch as follows:

- Assume that we are at the end of epoch  $e$ .
- Assume further that a conversion exists for Namada tokens that allows users to convert  $(X) \text{NAM}$  from epoch  $e-1$  into  $(X+Y) \text{NAM}$  from epoch  $e$ .
- At the end of epoch  $e$ , the protocol will create a new conversion that allows users to convert  $(X+Y) \text{NAM}$  from epoch  $e$  into  $(X+Y+Z) \text{NAM}$  from epoch  $e+1$ .

This is done for efficiency: this way it is very efficient to compute the conversion for native tokens between non-consecutive epochs.

Conversions are a way of implementing a reward system to incentivize users to use the shielded pool. It is important to note that users can only claim conversions if they meet the minimum amount required. For instance, take the previous example in which we have a conversion that allows a user to convert  $X$  Namada tokens from epoch  $e-1$  into  $(X+Y)$  Namada tokens from epoch  $e$ . A user can only claim rewards for its Namada tokens from epoch  $e-1$  if it has at least  $X$  Namada tokens from epoch  $e$ .

Because of the way native token conversions are computed, a user is required to have greater amounts of native tokens as epochs increase to be able to claim rewards. We argue that eventually, the amount could be prohibitive

for normal users such that they are unable to claim rewards, which may disincentivize them from shielding their native tokens. We see this as a limitation of the reward system.

## Problem Scenarios

1. Assume that the conversion amount requirements for native tokens grow fast with epochs.
2. Assume that we are at epoch  $e \gg 0$ , and that the computed conversion for native tokens at the end of epoch  $e-1$  is  $-(X) \text{NAM}_{@e-1} + (X+Y) \text{NAM}_{@e}$ .
3. By (1),  $(X+Y)$  is a big number, such only a minority of users own that amount of Namada tokens.
4. Any user that shields its Namada tokens at any epoch  $e' \geq e$  won't be able to claim any rewards.
5. The above may disincentive them from shielding their native tokens.

## Recommendation

We recommend the team first assess how fast the conversion amount requirements for native tokens grow with epochs via simulation. If the experiment concludes that the conversion amount requirements for native tokens may grow fast, we recommend rethinking how conversions are computed for native tokens. An alternative could be to compute them similarly to how it is computed for non-native tokens, at the cost of making the computation of conversions between non-consecutive epochs more costly.

## Earlier rejection of transactions with invalid sapling value balance

<b>Title</b>	Earlier rejection of transactions with invalid sapling value balance
<b>Project</b>	Anoma: MASP
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	1 LOW
<b>Impact</b>	1 LOW
<b>Exploitability</b>	1 LOW
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2721">https://github.com/anoma/namada/issues/2721</a>

### Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)

### Description

The masp validity predicate only checks that the `sapling_value_balance` is well computed at the very end of the `verify_shielded_tx` function. This is unnecessary, one could early reject shielded transactions with an invalid `sapling_value_balance` by simply doing the computation at the beginning of the validation.

Note that similar checks could added to early reject transactions for other reasons, e.g., if a transaction is transparent (see [this other finding](#)).

### Problem Scenarios

For example, if the shielded transaction (no transparent inputs or outputs) is in place, `sapling_value_balance` is expected to be zero. A malicious user could use custom code to generate a shielded transaction with a spend, no shielded output, and 0 `sapling_value_balance`. This transaction would go through the validation with no alert until the very end where the `verify_shielded_tx` is called. This function includes such a check that would prevent this transaction for being accepted.

We argue that this is a waste of resources, even though the user will pay gas for the validation and verification of the transaction.

## Recommendation

We recommend implementing an explicit early check on `sapling_value_balance` against expected spends, converts, and sapling outputs. This measure will filter out transactions exploiting current validation processes, mitigating unnecessary computation, and preventing potential storage manipulations by invalid transactions.

## Minor code improvements

<b>Title</b>	Minor code improvements
<b>Project</b>	Anoma: MASP
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	0 INFORMATIONAL
<b>Impact</b>	0 NONE
<b>Exploitability</b>	0 NONE
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	<a href="https://github.com/anoma/namada/issues/2722">https://github.com/anoma/namada/issues/2722</a>

## Involved artifacts

- [crates/namada/src/ledger/native\\_vp/masp.rs](#)

## Description

1. This logging message is about the transfer source address and not the transfer target address.
2. We recommend strengthening this check: at the moment only check that the transparent bundle isn't empty, but we recommend checking that the transparent bundle has a non-empty set of transparent inputs.
3. This error message is misleading. It should be: "Expected transparent inputs in shielding transaction" if the above recommendation is accepted.
4. Potential overflows:
  - The calculation of native rewards could potentially lead to overflow. If the expression `(addr_bal / normed_inflation) * new_normed_inflation`, which transforms due to the custom multiplication implementation, produces a value over the limit of Uint, the panic will occur.
  - Potential overflow due to similar reasons detected [here](#) also
  - Checked arithmetics should be used in [this](#) and [this](#) expression also for multiplication.
5. Comment and assigned value do not match. The comment says that in case of overflow, `new_normed_inflation` will be assigned the value 0, however `* normed_inflation` is assigned to it.
6. Some of the code in validity predicate is almost the same, so there is a chance of optimizing readability and putting this code into separate function. The code [here](#), and the code [here](#) could probably be exported to single function since they're doing similar computations just for input/output respectively, or at least some parts of it could be. For example, asset type checks and total input/output values calculation as well as the public key check are similar in a way that only the value taken into account is from input or from output.





## Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of [Common Vulnerability Scoring System \(CVSS\) v3.1](#), which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the [Impact score](#), and the [Exploitability score](#). The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ [CVSS Qualitative Severity Rating Scale](#), and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

### Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	Examples
 <b>High</b>	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
 <b>Medium</b>	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
 <b>Low</b>	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
 <b>None</b>	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

### Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

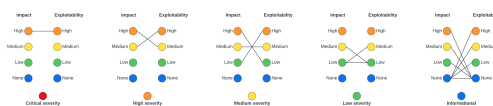


- *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
- *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	Examples
<span style="color: orange;">●</span> <b>High</b>	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
<span style="color: gold;">●</span> <b>Medium</b>	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
<span style="color: green;">●</span> <b>Low</b>	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
<span style="color: blue;">●</span> <b>None</b>	illegitimate actions taken in a coordinated fashion by all actors



## Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	Examples
<span style="color: red;">●</span> <b>Critical</b>	Halting of chain via a submission of a specially crafted transaction
<span style="color: orange;">●</span> <b>High</b>	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
<span style="color: gold;">●</span> <b>Medium</b>	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users

Severity Score	Examples
 <b>Low</b>	2x increase in node computational requirements via coordinated withdrawal of all user tokens
 <b>Informational</b>	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

## 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 bugfree 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.