Authors:

oleh oleh.blockchain@gmail.com

Created:

2024-08-10

Abstract

This RFC proposes eliminating the shield

, unshield

, and redeem_shield

methods from the token transfer system, replacing them with more intuitive methods: transfer_from_public_to_private

and transfer_from_private_to_public

. The changes aim to reduce redundancy, simplify the codebase, and make the token transfer process more understandable for developers.

Background

Currently, the token contract includes two distinct mechanisms for transferring tokens:

1. Direct Transfers:
2. transfer(from, to, amount)

— Transfers tokens privately, where both from

and to

are kept private.

- transfer_public(from, to, amount)

— Transfers tokens publicly, where both from

and to

are public.

1. transfer(from, to, amount)

— Transfers tokens privately, where both from

and to

are kept private.

1. transfer_public(from, to, amount)

— Transfers tokens publicly, where both from

and to

are public.

1. Shielded Transfers:
2. shield(from, amount, secret_hash)

— Transfers tokens from a public address from

to a private secret

(via secret_hash

).

- redeem_shield(to, amount, secret)

— Transfers tokens from a private secret

to a private address to

.

1. shield(from, amount, secret_hash)

— Transfers tokens from a public address from

to a private secret

(via secret_hash

).

1. redeem_shield(to, amount, secret)

— Transfers tokens from a private secret

to a private address to

.

While direct transfers are straightforward and essential, the shielded transfer mechanism introduces redundancy and potential confusion.

Problem Statement

The existence of two separate mechanisms for transferring tokens (transfer

/transfer_public

and shield

/redeem_shield

) is redundant. The redeem_shield

function, in particular, is nearly identical to the transfer

method, with the only difference being the use of a secret

instead of the from

address. This redundancy complicates the token transfer logic and could confuse developers.

Proposal

To simplify the token transfer system and remove redundancy, the following changes are proposed:

1. Rename shield

to transfer_from_public_to_private

:

- Instead of generating a secret

and secret_hash

, the user generates a burner sub-account.

- The shield

function will be replaced with transfer_from_public_to_private(from, burner_address, amount)

, where burner_address

is a temporary address created for privacy purposes.

1. Instead of generating a secret

and secret_hash

, the user generates a burner sub-account.

1. The shield

function will be replaced with transfer_from_public_to_private(from, burner_address, amount)

, where burner_address

is a temporary address created for privacy purposes.

1. Remove redeem_shield

:

- Tokens are transferred from the burner sub-account to the user’s main account using the existing transfer

method: transfer(burner_address, to, amount)

.

1. Tokens are transferred from the burner sub-account to the user’s main account using the existing transfer

method: transfer(burner_address, to, amount)

.

1. Rename unshield

to transfer_from_private_to_public

:

- This renaming will align with the new transfer method naming conventions, ensuring consistency in the contract’s functions.
- This renaming will align with the new transfer method naming conventions, ensuring consistency in the contract’s functions.

Implementation

contract Token { #[aztec(storage)] struct Storage { // ... - pending_shields: PrivateSet, }

- # [aztec(public)]

- fn shield(from: AztecAddress, amount: Field, secret_hash: Field, nonce: Field) {
- // removed

- }

- # [aztec(public)]

- fn transfer_from_public_to_private(from: AztecAddress, to: AztecAddress, amount: Field, nonce: Field) {
- storage.public_balances.get(from).sub(amount);
- storage.balances.add(to, amount);

- }

- # [aztec(private)]

- fn redeem_shield(to: AztecAddress, amount: Field, secret: Field) {
- // removed

- }

[aztec(private)]

- fn unshield(from: AztecAddress, to: AztecAddress, amount: Field, nonce: Field) {
- fn transfer_from_private_to_public(from: AztecAddress, to: AztecAddress, amount: Field, nonce: Field) { // implementation is the same } }

This refactor simplifies the token contract by removing the concept of shielding, reducing the number of methods developers need to learn, and consolidating the transfer logic.

Example Usage

Below is a comparison of how the token transfer functionality is used in JavaScript:

Before:

const [bobSecret, bobSecretHash] = await generateSecretAndHash(); await token.withWallet(alice).shield(alice, amount, bobSecretHash); await token.withWallet(bob).redeem_shield(bob, amount, bobSecret);

After the proposed changes:

const bobBurner = await bob.createBurnerAccount(); await token.withWallet(alice).transfer_from_public_to_private(alice, bobBurner, amount); await token.withWallet(bobBurner).createAuthWit({ caller: bob, action: token.transfer(bobBurner, bob, amount).request() }); await token.withWallet(bob).transfer(bobBurner, bob, amount);

The updated JavaScript code is more unified due to the consistent use of transfer methods. Instead of managing secrets and hashes, developers now work with burner accounts, using clear and consistent transfer terminology throughout the process.

Pros & Cons

Pros

- Simplicity:

The token transfer mechanism is streamlined, with fewer concepts and methods to understand.

- Elimination of Redundancy:

The secret

and secret_hash

logic, as well as the pending_shields

and TransparentNote

structures, are no longer necessary.

- Clarity:

Developers can more easily grasp the token transfer process, as there is only one transfer mechanism to learn.

Cons

- Privacy Risks:

Transferring tokens from a public to a private address could expose privacy risks if the destination is a main account rather than a burner account. The previous shielding process made it clear that tokens were being shielded for privacy, which may not be as obvious with the new approach.

- Needs an additional .createAuthWit(...)

call on JS side.

Open Questions

1. How does transfer(burner_address, to, amount)

compare to redeem_shield(to, amount, secret)

in terms of constraints used, assuming that the amount

is represented as a single note.

1. Can the call to .createAuthWit(...)

be avoided on JS side?

Conclusion

This RFC recommends simplifying the token contract by removing the shielding mechanism and replacing it with a more intuitive transfer method. By doing so, the token transfer process will become more straightforward, reducing the cognitive load on developers and improving overall system clarity.