While doing the Token contract tutorial I wonder what problems would be encountered if redeem_shield

would be simplified (I see why a pop_notes

example is wonderful for tutorial, so just asking my question here).

I mean pending_shields

are filled from the public execution hence it's not encrypted and available to anyone to see, so it makes sense to find the relevant note (with its header

) user wants to redeem, load it into the PXE and then just remove

from the pending_shields

set privately using the prepared note as the hint.

I still wonder if remove

even works this way. Intuition tells me it should just fail if the note isn't in the set, but I'm having trouble to understand https://github.com/AztecProtocol/aztec-packages/blob/48914ba71039f18d7cea9fca65997c2a6e263b25/noir-projects/aztec-nr/aztec/src/state_vars/private_set.nr#L70: if my snippet would pass this assertion while de-serializing or not. Also I'm sure I missed something else and would be happy to learn what I'm overlooking.

And all remarks are welcome! Like is it bad to insert

directly into a balance set, it feels like it shouldn't hurt it, since I feel it's just a trait for the token notes. How bad is it to have U128

inside a note instead of Field

(anyway it will be checked, and I wonder why not there).

# [private]

fn redeem_shield( secret: Field, to: AztecAddress, // note_the: TransparentNote note_content_serialized: [Field; 3] ) { let note_the = TransparentNote::deserialize_content(note_content_serialized); assert(note_the.secret_hashed == compute_secret_hash(secret)); let amount = note_the.amount; storage.pending_shields.remove(note_the); let to_keys = get_public_keys(to); storage.balances.at(to).set.insert(&mut TokenNote{ amount, npk_m_hashed: to_keys.npk_m.hash(), randomness: dep::aztec::oracle::unsafe_rand::unsafe_rand(), header: dep::aztec::prelude::NoteHeader::empty() }).emit(encode_and_encrypt_note_with_keys( &mut context, get_public_keys(context.msg_sender()).ovpk_m, to_keys.ivpk_m, to )); }