Question

Do we have either of these features implemented yet, and/or is there a plan to do so? Thenonce rework (recently finished) will make these features pretty important.

Notes and definitions

- nonce = hash(first_nullifier, noteIndexInTx)

where first_nullifier

is the tx-hash and noteIndexInTx

is the index of the noteHash

in the TX's newNoteHashes

array * Note: this array omits "transient" noteHashes that were nullified/squashed in the same TX. "Transient" notes are not assigned a nonce.

- Note: this array omits "transient" noteHashes that were nullified/squashed in the same TX. "Transient" notes are not assigned a nonce.
- noteHash

: direct hash of a note preimage

- innerNoteHash

: a noteHash siloed by storage slot

- siloedNoteHash

: an innerNoteHash siloed by contract address

- uniqueSiloedNoteHash

: a siloedNoteHash hashed with nonce

- NoteProcessor

: client-side component that listens for new L2 blocks and attempts to decrypt TX logs that target the current user. Loads any successfully decrypted notes into client's DB.

Why these features are important after nonce rework

Reference NonNativeToken

contract as an example.

I create a commitment/noteHash in public that I later want to claim in private. I know the note preimage (amount, secretHash), so I can compute the noteHash easily. But today all that ends up in L1 calldata for my note is the final uniqueSiloedNoteHash. The nonce used to make this noteHash (created in a public function) unique can only be known by the Sequencer/Prover since they will execute the public section of the transaction. If at some later time I want to claim my funds in private, I'll need to know the uniqueSiloedNoteHash for my note in order to nullify it.

So how do we handle this?

Well I could record the tx-hash alongside the secret and then when I want to claim in private I can find the TX's L1 calldata, but I still won't know which

final noteHash (unique / hashed with nonce) corresponds to my note. So, the public function could emit an unencrypted log that includes the my noteHash (non-unique) such that I can determine the nonce assigned to my noteHash along with the uniqueSiloedNoteHash. I can subscribe to unencrypted logs (for that contract) that include my noteHash and can then use that information to claim/redeem the note privately when I need to nullify the uniqueSiloedNoteHash.

Also relevant for noteHashes created in private!

These challenges can also be relevant to noteHashes/commitments created in private functions (see ZkTokenContract's

createClaims

and claim

functions). Basically if a note is not encrypted in calldata for decryption by the receiver, it will not be successfully decrypted by the receiver's note-processor and therefore will not end up in their DB. So, the note creator will need to communicate the commitment nonce to the receiver (which, in the ZkTokenContract example, can be communicated alongside the "secret"/note-preimage).

More info on nonce rework

in this PR.

Why not use private data tree leaf index as nonce?

"We originally did this in Aztec Connect, but then we encountered problems when trying to chain transactions (i.e. squash nullifiers and commitments between multiple txs in the same rollup). If I want to create tx1 and tx2 in the same rollup block, and tx2 spends a note which came from tx1, then I need to know the nonce for the note in order to nullify it. I don't want to wait for the sequencer to tell me the leafIndex before I can start generating a proof for tx2. So we can't use the leafIndex, because I don't know it, because I don't know where the sequencer will sequence my tx1 in the block. Also, tx1 might get put in one rollup block, and tx2 might not get included until some later rollup block, in which case the nullifier for the note has to be a proper nullifier (with some kind of nonce - not a half-baked transient nullifier), because it could end up in the nullifier tree forever." - @Mike

I am particularly interested in hearing from@leila, @PhilWindle and @spalladino and whether your teams would be able to pick these features up.