With the TokenBridge example, suppose I do something like this:

l2Recipient = 0x1234 amount = 0x9999 secretHash = computeSecretHash("foo")

tx1 = l1Bridge.depositToAztecPublic(l2Recipient, amount, secretHash) // send another message with the exact same content tx2 = l1Bridge.depositToAztecPublic(l2Recipient, amount, secretHash)

Only one of the messages will be consumable because the message nullifier's end up being equivalent, so after consuming one, attempting to consume the second one fails with:

Assertion failed: L1-to-L2 message is already nullified '!self.nullifier_exists(nullifier, self.this_address())'

It seems L1->L2 messaging is relying on the secretHash

being different each time, but it doesn't seem reasonable to rely on the end user to always provide a unique value.

This seems like a potentially painful gotcha, unless I'm missing something obvious.

Assuming this is actually a problem:

The simplest solution would be to add a unique value to the message in the TokenBridge contract, but that relies on all bridge contract developers knowing about this pitfall and planning for it. Considering it was missed in the example contract, that seems likely to cause problems for someone in the future.

I think a more appropriate solution would be to have the L1 Inbox contract add something unique to the message that can help ensure the uniqueness of the nullifier. The most straight forward method would probably be to use the totalMessagesInserted counter value that already exists, doing something like:

// combine totalMessagesInserted with the original contentHash contentHash = sha256ToField(abi.encode(totalMessagesInserted, contentHash)); // add totalMessagesInserted to the event log emit MessageSent(, totalMessagesInserted);

Does this seem reasonable? I can open a github issue & potentially send a PR if folks agree.