Given the different ways we have for sharing notes (provable on-chain, on-chain, and off-chain), the different types of encryption keys we envision (incoming, outgoing, internal incoming), and the different use cases (direct transfer, swap, etc), what should a token contract do in the event of a transfer?

Provable encryption and broadcasting

When creating a note, we have three different options for encrypting and sharing it:

1. Create the note without constraining its encryption, and then share the encrypted note with the recipient via an offchain channel.
2. This is the cheapest option as it does not require proving encryption or tagging, and does not consume L1 space for broadcasting the cyphertext.
3. Useful eg on direct transfers, when the sender wants

the receiver to get their funds, since if the receiver doesn't get their funds, they won't give the sender the coffee they just paid for!

1. This is the cheapest option as it does not require proving encryption or tagging, and does not consume L1 space for broadcasting the cyphertext.
2. Useful eg on direct transfers, when the sender wants

the receiver to get their funds, since if the receiver doesn't get their funds, they won't give the sender the coffee they just paid for!

1. Encrypt the note without constraining it, but share it onchain by broadcasting an event.
2. This keeps proving times low, but requires costly L1 space for broadcasting the cyphertext.
3. Similar to the above, requires the sender to be cooperative since they could grief the recipient by incorrectly encrypting the note. Useful when the sender cannot or doesn't want to establish an offchain communication channel with the recipient, or when the recipient wants to be able to reconstruct their private state from L1.
4. This keeps proving times low, but requires costly L1 space for broadcasting the cyphertext.
5. Similar to the above, requires the sender to be cooperative since they could grief the recipient by incorrectly encrypting the note. Useful when the sender cannot or doesn't want to establish an offchain communication channel with the recipient, or when the recipient wants to be able to reconstruct their private state from L1.
6. Provably encrypt and tag the note, and broadcast it onchain.
7. Most expensive option, in terms of both cost and proving time.
8. Useful in adversarial settings, eg a swap, where the sender could grief the recipient by not sending them the preimage of their share of the swap, while running away with their cut.
9. Note that there are workarounds that can prevent the sender from griefing, but they are usually specific to each use case and often require more complex interactions.
10. Most expensive option, in terms of both cost and proving time.
11. Useful in adversarial settings, eg a swap, where the sender could grief the recipient by not sending them the preimage of their share of the swap, while running away with their cut.
12. Note that there are workarounds that can prevent the sender from griefing, but they are usually specific to each use case and often require more complex interactions.

Encryption keys and a transfer example

We envision three different flavors of encryption keys. From the yellow paper, by@Mike:

Definitions (from the point of view of a user ("yourself")):

- Incoming data: Data which has been created by someone else, and sent to yourself.
- Outgoing data: Data which has been sent to somebody else, from you.
- Internal Incoming data: Data which has been created by you, and has been sent to yourself.

- Note: this was an important observation by ZCash. Before this distinction, whenever a ‘change’ note was being created, it was being broadcast as incoming data, but that allowed a 3rd party who was only meant to have been granted access to view “incoming” data (and not “outgoing” data), was also able to learn that an “outgoing” transaction had taken place (including information about the notes which were spent). The addition of “internal incoming” keys enables a user to keep interactions with themselves private and separate from interactions with others.
- Note: this was an important observation by ZCash. Before this distinction, whenever a ‘change’ note was being created, it was being broadcast as incoming data, but that allowed a 3rd party who was only meant to have been granted access to view “incoming” data (and not “outgoing” data), was also able to learn that an “outgoing” transaction had taken place (including information about the notes which were spent). The addition of “internal incoming” keys enables a user to keep interactions with themselves private and separate from interactions with others.

Let’s apply these keys in a token transfer example:

- Sender consumes one or more notes that sum up to at least the amount to transfer
- A note is created for the recipient with the amount transferred, and is encrypted with their incoming viewing key
- That same note is also encrypted for the sender themselves, using their outgoing viewing key
- A change note is created for the sender and encrypted with their internal incoming viewing key

So, what should we do in terms of provable encryption and broadcasting in each scenario…?

Encrypting for self

When encrypting for self, the outgoing and incoming internal scenarios, there is no need to constrain correct encryption

. But whether to broadcast the note on chain depends exclusively on whether the user wants to be able to recover their state or transaction history from on-chain data. This is a tradeoff between tx cost and recoverability, and may depend on each user, each app, and even the value of the note.

The easiest way out for us

most flexible option is to let the wallet software decide

. The software could decide for the user based on their preferences, or even let them choose on a note-by-note basis if they are advanced enough.

Implementation-wise, this means that encrypting and tagging with an incoming internal or outgoing key should not be constrained, and broadcasting should depend on the result of an oracle call.

Encrypting for someone else

Let’s now go into the note sent to the recipient. As we discussed in the very beginning of this post, there are valid scenarios for provable encryption, for onchain broadcast, and for cheap offchain broadcast. And these depend on the context of the transfer, not on the token implementation itself.

- If the transfer

is being executed by another app, like a swap contract, it’ll need to constrain encryption and broadcasting, to prevent a malicious sender from griefing the recipient.

- If the transfer

is direct from one user to another, we most likely don’t need to constrain encryption and tagging, but it’s unclear whether to broadcast it on-chain: * If the sender wants to broadcast in order to avoid manually sending the note off-chain, all good.

- But if the receiver wants the note broadcasted so they can reconstruct their state later, they are putting that burden on the sender, who needs to pay for the extra gas cost. Also, we don’t have the receiver’s wallet software around to aks it whether the receiver wants to have this specific note broadcasted or not (as we did in the “encrypt for self” scenario).
- If the sender wants to broadcast in order to avoid manually sending the note off-chain, all good.
- But if the receiver wants the note broadcasted so they can reconstruct their state later, they are putting that burden on the sender, who needs to pay for the extra gas cost. Also, we don’t have the receiver’s wallet software around to aks it whether the receiver wants to have this specific note broadcasted or not (as we did in the “encrypt for self” scenario).

From the above it seems that we need to add a flag to the transfer

method (or have two different flavors of transfer

) such that the msg.sender can choose whether to provably encrypt or not.

As for broadcasting, it seems we have a few options:

1. We can have the receiver record their preference of “I want my noted broadcasted” in the registry, but this does not provide flexibility per-app (much less per-note).
2. We can make it so the sender always

broadcasts in a token transfer, for the sake of simplicity and security, at the expense of more expensive transfers.

1. We can add a way for the receiver to send a tx themselves where they broadcast the note they received off-chain - if they so much want the onchain backup, have them pay for it themselves (though the extra cost of an additional tx probably makes the whole thing more expensive overall).

Encrypting for sender by someone else

We should stress that “encrypting for self” and “encrypting for the sender” is not the same. Thanks to authwits and randomized nullifiers, an authorized third party could transfer funds on behalf of the sender, usually in the context of a broader tx. Think of a transferFrom

in Ethereum executed from a contract, like a swap.

These transferFrom

operations should follow the same rules as when “encrypting for someone else” (described above), since what matters is who is locally executing the tx then and not the owner of the funds. Still, here we have the option of the sender encoding their broadcasting preferences in the authwit message.

Note that, if the incoming viewing key is used here, this will lead to records that describe value sent will be sometimes encrypted with an outgoing key (when the executor of the tx was the owner) and sometimes with an incoming key (when the executor was an authorized third party). We could avoid this issue by always using the outgoing key, regardless of who encrypts, but this means we cannot use the outgoing secret key for encryption as originally planned.

Encryption and compliance

Note that any of the scenarios described above where encryption is unconstrained or broadcasting is optional potentially break audit compliance for the app. If an application requires all its interactions to be auditable by having the user share an app-siloed key with a 3rd party auditor, then unconstrained encryption or optional broadcasting can be used for bypassing it. Apps that have audit compliance requirements should always provably encrypt and broadcast, even if it’s more expensive.

Wrapping up

We have the following decisions to make:

- Encrypting data for self should be conditional on an oracle call, so the wallet controls whether to generate an outgoing trail or backup change notes.
- Token transfer

method should accept a flag for whether to prove encryption of the recipient note or not.

- Token transfer

method should always broadcast the recipient note, for the sake of simplicity.

- Outgoing encryption can only be done by the holder of the key, so even for notes that “semantically” mean outgoing data (eg transferFrom

), if they are produced by a 3rd party, we use the incoming key.

information on this forum for any purpose - the development, release, and timing of any products, features or functionality remains subject to change and is currently entirely hypothetical. Nothing on this forum should be treated as an offer to sell any security or any other asset by Aztec Labs or its affiliates, and you should not rely on any forum posts or content for advice of any kind, including legal, investment, financial, tax or other professional advice.”