Noteworthy thoughts on sharing Notes on Aztec

| pun intended

Introduction

While building the Aztec Private Oracle MVP we quickly found out that sharing Notes among different participants of the Oracle would be a key mechanism of the design.

Aztec private oracle provides an oracle solution for those who want to keep their privacy. When asking something to a particular address (called divinity), the question and the answer are encrypted and only the requester and the divinity can read it, no other party has access to it. This allows for users to be able to ask anything they want and for the data to be priced more fairly as current oracle systems publicly share their data feeds so everyone can read it for free instead of paying for every usage.

Thanks to @spalladino and the Aztec team for all their help and for providing feedback in reviewing this document!

Why do we need to share a Note?

Sharing private information might sound counter-intuitive, but it allows us to coordinate different parties, which don't necessarily need to know each-other.

For example, when a User creates a new Question Note

, the contract emits 2 logs, one for this User, and another one for assigned party to answer that Question Note

.

In this example. only 1 note is created, but since it has been shared to both participants each can do one specific flow with that Question Note

before it get's nullified/used by the other party.

The User can "cancel" the request and get it's payment back, only as long as the other party has not yet nullified/used the Question Note

for creating a response and claiming the payment.

Sharing a note in this way is quite easy, since it only requires the User to know the public_encryption_key

of the other party.

This has "somewhat"

the same effect as creating completely separate Notes, which share the same nullifier.

Sharing a nullifier is useful when you want a note to hold specific data for each participant. i.e. we might want the UserNote to enforce a specific expiration time, without having the divinity know about this.

This is useful when coordination is required between two different Note types.

But for same Note types, emitting logs is the optimal way to go.

It's not so easy tho, if we'd like to add more participants we quickly run into a few issues.

Common issues and limitation when sharing keys though public encryption keys

Let's say we want to create and share one single new Note with 10 other participants.

We would need to provide and loop over all participants public_encryption_keys

, Aztec has a few limits for this,see here, that quickly makes this a no-go solution if we keep on adding participants.

It also has the downside that the User creating these Notes, needs to know all participants, we might not want that.

Another approach would be to obfuscate the list of participants from the User. But it's not so easy to do also.

Currently there is research being done on diversified

and stealth

addresses that could help with this issue.

Creating a private quorum

We'd like to have the following properties on our Quorum:

Participants should not know the size of the quorum.

Participants should not know each-other, this protects against coordination attacks.

If required, it's also easy to add functionality to allow participants to prove that they are part of this quorum.

Users/Requesters should not know the size nor the participants of the quorum.

The Creator of the quorum should not know the Participants. It should only set the rules of the quorum. (This is yet not a property we have on the design below)

The Quorum should be dynamic, in participants and size. (Not yet a property of the current design)

We are currently working on the dynamic quorum design

For a pre-defined list of N participants that would all share the same encryption_key

we can use some tricks.

Let's go though some Figma flows now!

[

1104×727 86.9 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/a39cde296fdf6e657ed973ae75c24dc113dad0ac.png)

This is our first PoC of a private quorum, so take anything you see here with a grain of salt. (also, any and all suggestions are more than welcomed!!)

The first action is to deploy a new Quorum, ideally for this there would be an easy way to validate the deployed contract source. This actions can be done by anyone.

Then, the contract deployer or Creator would initialize the Private Quorum by seeding it the required parameters.

We are using a merkle root as a way to store all participants, but any other method is also valid. (i.e. it can be a simple hash of the array of participants)

Since this is an immutable singleton note, a specific nullifier is created under the hood for us.

[

1042×344 36.3 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/d70df6ef0ebf06b8905e63f7f99a553447be5b17.png)

Divinities will get the secret_key

, that will let them read the Question Notes sent to the Private Quorum, one by one. The Creator needs to generate all the parameters off-chain. Repeating this process for all Divinities in the quorum.

It should also, not included in this flow, keep track using a private Note of the total amount of divinities it already shared the secret key with. So only when this matches the total divinities quantity, can the Private Quorum be enabled to receive Notes.

For this to work Secret Key Notes need to behave as Immutable Singleton notes too, to force a revert if the Creator sends a Secret Key Note to the same Divinity twice.

Note: we first wanted to use Diffie-Hellman to share the secret_key

but it proved to add unnecessary complexity, also, to avoid leaking the privacy of the public keys of all other participants, the Creator had to send for Participant1 the product of all of the other participants public_keys

, repeating for each of the other participants. This had to be done on the contract to avoid having the Creator input invalid off-chain generated products, which also increased code complexity and costs.

So, for the MVP we decided on just encrypting the secret_key

using the receiving divinity public key.

As you might have noticed, the Creator of the Private Quorum will know ALL participants, and also, chooses who would be on it. So there is an absolute trust assumption that the Creator will behave honestly when creating the Quorum. This is not a desired scenario, and we have a solution for a Private Dynamic Divinities Quorum that preserves the identity of the participants, while also allowing them to somewhat-freely join and leave.

If solving these problems peaks your interest, please reach out! we'd love chat

Day to day operations after complex setup:

[

1167×500 44.9 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/db3b4eb8ea1a798e29098cbdf366a3dd69e77461.png)

Here we can see why is so important to abstract all this complexity away from the Private Oracle, who treats the Private Quorum as a random singular Divinity.

Protocols requesting Private Answer Notes for their computation, can whitelist one or multiple trusted Private Quorums. And validate those against the divinity field on the Answer Note provided by their User.

[

1023×576 51.4 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/c5967c62f018903a1a7d533a633d2f68669fdd52.png)

Internally on the Private Quorum, the flow is a bit more complex and requires input from all (or almost-all) parties.

Divinities can send their answers without any pre-requisite or coordination.

They are tho, forced by a Required/Unique nullifier, similar to how ImmutableSingletonNotes

work, to submit the same answer to the same question if they already previously answered it.

If they try to create a new answer A2 to question Q, by hiding that they already previously answered A1, the contract will generate the unique divinity-question-nullifier

which will revert as that nullifier was created with A1.

If the Divinity indeed provides answer A1 for question Q, the unique nullifier is not re-created, as question Q for that divinity is treated as a "repeated" question.

[

1344×668 77.6 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/b2585a97f2b2fc4148884812668ed147550b17e8.png)

The Requester, which is the one who asks/creates the Question, can start matching the Answer Notes as they get created and broadcast to them.

We are using these Matching Answer Notes to merge Answer Notes async, so we can circumvent some of Aztec's currently enforced limitations.

[

909×678 42.5 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/603f83832bc481984faa7722d85ee67109527fae.png)

The Requester is the only one that can finalize the Request flow on the Private Quorum.

We can also add an expiration date, that would enable any Divinity to forcefully finalize release the payment. But it's out of scope for now.

The Requester will only provide the “winning” Matching Answer Note.

This will in turn, on the Private Oracle, create 2 Answer Notes, one for the Requester, and another one for the Private Quorum.

The latter is very important since it will be used for two things, claiming the respective payment from each Divinity who proves they answered the same finalized Answer.

And to enforce the Private Quorum to reply to exactly the same Answer if asked the same question.

This is critical to avoid having to go though the previous process for all subsequent queries to the same question.

We currently don’t have a Figma flow for this, but it would just be a separate function any single Quorum Divinity could call to atomically finalize the Request, if they provide the question-matching Answer Note.

That’s all!

We’ll try to keep this up to date with more interesting designs, such as the Private Dynamic Divinities Quorum so stay tuned!