Calling all visionaries, innovators, and builders! Aztec's testnet launch is imminent and we are inviting you to take on one of the most critical and rewarding challenges left for the Aztec ecosystem: the Wallet. Building this wallet isn't just a software development project; it's a chance to redefine how users interact onchain. We are looking for brilliant minds who will have the chance to build an elegant, secure, and user-friendly gateway that will empower millions to harness the full potential of privacy-preserving blockchains.

Aztec Labs is allocating $200,000-$300,000 to support 2-3 extraordinary teams in their quest to build the ultimate testnet wallet. We're scouting for visionaries who are in it for the long haul, ready to push beyond testnet.

Wallets serve as the primary user interface for blockchain networks. Yet, they have been clunky, confusing and sometimes downright frustrating. Aztec's combination of public and private execution, native account abstraction and other advanced features present an opportunity to redefine the wallet landscape. On Aztec, wallets could become a safer space for users, where they can view their private or public balances, generate client-side proofs for private functions, and effortlessly sync with the network, without having to navigate the complexities of cryptography. Please read on to the Requirements section to learn more about Aztec and the wallet design space.

With the Aztec Sandbox live (a local development playground) and the Aztec Devnet (our gated persistent environment with an advanced proving system), there isn't a better time to start building! As we gear up for the decentralized testnet launch, the wallet developed could act as the heartbeat of Aztec's ecosystem.

We look forward to see how you shall shape Aztec's future. Proposals must be submitted before 4 October 2024 to be considered

. Please also refer to Appendix 1 on Potential Areas of Interest if building wallets themselves don't interest you.

EDIT

This deadline has now passed. Winners of this grant will be announced by 18th October 2024

Grant Details and Submission Format

We are excited to support independent developers and small companies in their journey to build on Aztec. We will provide stipends to up to three selected teams to help offset the costs of onboarding and development. We are particularly interested in teams with a long-term vision for the Aztec ecosystem and a commitment to contributing to the developer community.

To ensure consistency and facilitate the review process, kindly adhere to the following submission format:

Title

: Team/Product name

Contact Details

: where Aztec Labs can contact you - Signal, email etc.

Summary

: A brief, easy-to-understand summary of your proposal (about 300 words)

Estimated Start and End Date

: recommended to start in October (ASAP), with a functional version out for testnet in December and final version out by mid Q1 2025.

About You

: Short paragraph with details about your team and their expertise or relevant experience etc to show us you have what it takes.

Details

: Any designs (technical system designs), what wallet will you build (browser, iFrame, etc.), features to support, etc. Make this section unique and try to stand out from other proposals. You could even write an example account contract or show a wallet UI to stand out!

Grant Milestones and roadmap

: 3-5 milestones on how you expect to get to a testnet ready wallet, with at least 2-3 items dedicated to future work (post-testnet improvements, feature additions, and mainnet launch). Importantly please indicate your longer-term roadmap and how you want to keep building in the long run as an independent company. Please also prioritize our requirements to these

milestones such that you have a functional wallet by testnet launch in December and a roadmap on improving it through Q1 2025.

Grant amount requested

: Any grant requests over US $100,000 will be immediately rejected. Successful projects may get further grant support post launch, although please note that future funding or support is not guaranteed.

Grant budget rationale

: explain comprehensively, why you are requesting the amount of funds, how it relates to milestones and timelines.

Questions

: Any outstanding questions. Any confidential questions can also be sent to rahul[at]aztecprotocol[dot]com

Submissions should be created as a new post on this forum, tagged wallet

and rfgp

. Once the new post is created, please refer back to this RFGP and post the link to your proposal as a comment.

Please submit proposals by 4 October 2024

.

Okay, onto the fun stuff!

Requirements

About Aztec

Note that Aztec is under active development and breaking changes may occur. We will communicate these in advanced where possible. Please expect some breaking changes and note that functionality is subject to change. We also welcome any constructive feedback on Aztec's devex throughout the process.

Aztec is a privacy-preserving Layer 2 solution on Ethereum. Its unique execution environment offers features like private and public state, seamless composability between private and public transaction execution and native account abstraction.

Aztec has native account abstraction. This means

- There are no EOAs on Aztec. Everything is a smart contract
- Any function on any contract can act as an "entry point" of a transaction
- An entry point must be a private function
- It receives the actions to be carried out (app payload), fee payload and an authentication payload
- An entry point must be a private function
- It receives the actions to be carried out (app payload), fee payload and an authentication payload
- This allows account contracts to handle signature verification and payload execution as they please, as long as you can prove it, you can include any arbitrary logic.

Aztec supports both private and public transactions. All transactions begin as private. Private functions can enqueue calls to public functions. Users generate proofs of private computations client-side and submit them along with transaction hashes, note hashes, nullifiers, etc. The sequencer processes the remaining enqueued public functions, constructs the final transaction proof, and assembles an L2 block.

All of this is possible due to our custom VM (not evm compatible).

[

1600×436 80.7 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/5/51b11b427559e0291050b3fdf04b62ed8e1b7214.png)

PXE (Private eXecution Environment; pronounced "pixie") is the client side node that simulates and generates proof of private transactions. It also persistently stores the keys to discover new notes sent to them ("note discovery"), keep track of

nullified notes, look for encrypted logs etc.

The wallet can always assume that the PXE is running at a particular port. Note that currently, PXE doesn't work in the browser (it is a docker container running on the user's device) but by the testnet launch, we expect it to work in the browser. So for now a wallet can make local calls to the PXE port, but after that, a wallet should be able to use browser based PXE with minimal work (just changing port number).

Note that, we are still researching into how privacy preserving RPC providers could work. For testnet, a wallet should run an aztec node in a server, send all transaction calldata to that to forward to the mempool. In this way the server acts like a RPC proxy. This is because it may not be viable to expect users to run their own L2 Node

Requirements:

A wallet should implement the following requirements.

The wallet should be either browser-based (an iframe, embedded or browser based) or a PC app. It can also be a metamask snap based extension (although we'd like to see detailed description on how it is possible and some past experience with snaps)

The wallet should also integrate with Wallet Connect (similar to our ecosystem builder Oleh's wallet).

Onboarding

The onboarding process should aim to be extremely streamlined and user friendly with minimal steps and hassle for the user.

- Behind-the-scenes, generate the signing keypair (a.k.a. tx auth key), nullifier keypair, incoming viewing keypair and outgoing viewing keypair. These could all come from one master seed phrase with different derivation paths or from different seed phrases altogether.
- We will encourage wallet builders to move away from asking users to save 12 word seed phrases. Using passkeys is also similarly encouraged!
- Pay for the deployment of account contracts and the registering of the keys.
- Contract deployments incur gas fees so there is a chicken-and-egg situation here. However:
- contracts don't need to be deployed in order to receive funds (like with CREATE2 on EVM, addresses are deterministically derived from the contract bytecode and a user's signing public key)
- Contracts also don't need to be deployed if the user is just interacting with private functions (as there is no public code involved!). So account contracts only need to be deployed if using public authwits (more on this later).
- Furthermore, deployment can be delayed until the user makes their first transaction, similar to StarkNet where you need to "activate account".
- contracts don't need to be deployed in order to receive funds (like with CREATE2 on EVM, addresses are deterministically derived from the contract bytecode and a user's signing public key)
- Contracts also don't need to be deployed if the user is just interacting with private functions (as there is no public code involved!). So account contracts only need to be deployed if using public authwits (more on this later).
- Furthermore, deployment can be delayed until the user makes their first transaction, similar to StarkNet where you need to "activate account".
- At deployment, you must also register keys in the registry so anyone can look up your public keys to send you notes.
- Contract deployments incur gas fees so there is a chicken-and-egg situation here. However:
- contracts don't need to be deployed in order to receive funds (like with CREATE2 on EVM, addresses are deterministically derived from the contract bytecode and a user's signing public key)
- Contracts also don't need to be deployed if the user is just interacting with private functions (as there is no public code involved!). So account contracts only need to be deployed if using public authwits (more on this later).
- Furthermore, deployment can be delayed until the user makes their first transaction, similar to StarkNet where you need to "activate account".
- contracts don't need to be deployed in order to receive funds (like with CREATE2 on EVM, addresses are deterministically derived from the contract bytecode and a user's signing public key)

- Contracts also don't need to be deployed if the user is just interacting with private functions (as there is no public code involved!). So account contracts only need to be deployed if using public authwits (more on this later).
- Furthermore, deployment can be delayed until the user makes their first transaction, similar to StarkNet where you need to "activate account".
- At deployment, you must also register keys in the registry so anyone can look up your public keys to send you notes.
- Support first x transactions fee-wise
- Integrate with a token bridge (aka asset portal in aztec speak) for easy in-wallet bridging tokens
- An opinionated way shown to the user to pay fees (e.g. a dedicated FPC to facilitate paying fees in a user's tokens)
- Integrate an explorer for viewing transactions
- Provide any relevant educational material on Aztec for new users
- Show popular ecosystem tokens and their private and public token balance + fee juice balance

Account Contract

Written in Aztec.nr (our smart contract language)

- Choose appropriate signing key scheme (secp256r1 key most preferred)
- Authwit support -
- Authwits (or authorization witnesses) are most similar to approvalpermits on ethereum. Using authwits you can privately or publicly authorize certain addresses to perform actions on your behalf.
- Support adding, spending, cancelling authwits and checking validity of the authwit (is_valid)
- Authwits (or authorization witnesses) are most similar to approvalpermits on ethereum. Using authwits you can privately or publicly authorize certain addresses to perform actions on your behalf.
- Support adding, spending, cancelling authwits and checking validity of the authwit (is_valid)
- Set tx.max_block_number (addresses privacy issues when setting delay for shared mutables)
- Help create standards for account contracts, so various wallets can plug in with each other.
- Make account contracts upgradeable to enable migrating to a new different signature scheme, patch bugs or even allow users to switch wallets while retaining their address.
- Nonce abstraction and replay protection
- Support canceling pending transactions
- Support canceling pending transactions

Account Management - Syncing, Backups and Migrating

- Support managing multiple accounts in the same wallet
- Work on snapshots/quick syncing to enable exporting PXE information when migrating to the same wallet on a different device. Quick syncing/snapshots would inform the PXE quickly of all of the user's notes and also helps with backups.
- Similar to backups of a user's notes, is how to backup or recover a user's keys. While the signing key would reside with the wallet, the other keys will persist in the PXE.
- Ability to export the account to same wallet of another device (the keys, the latest synced notes)Zashi could be an inspiration for this.
- Ability to import account with the same classID/bytecode as the wallet's account contract.

Privacy and PXE data leaks

- Establishing a secure connection to the PXE to prevent dapps from just directly accessing PXE state
- Working with the ecosystem on a good secure standard for "connecting to an app"

- Informing of any PXE feedback, specifically related to authorization scopes to reduce potential data leaks around what an app could pull from a PXE once connected
- Create and manage secrets useful for L1->L2 messages or shielding
- Don't collect any logs/analytics that might leak personal or sensitive data.
- Showing currency conversation typically requires calling an API from the wallet which may leak IP address. All such leaks should have a disclaimer. Ideally they are turned off by default and user can switch them off. Zashi does this nicely.
- When routing any requests to the wallet's node, no privacy-leaking data should be preserved. Note that a wallet may have to talk to a node not just to send transactions but also to query the node for any information (such as merkle paths, note discovery etc.). Ensuring any personal data is deleted is similar to how Aleo's Leo Wallet delegates responsibility to their server too.

Contract Interactions

- Tab to select how user wants to pay fees
- Closer to testnet, we will provide more guidance but fees can be paid either via a potential protocol enshrined token ("fee juice") or via paymasters. A wallet should integrate with both.
- It could show an opinionated paymaster and/or show multiple different ones(or redirect to a website with sufficient information) to allow a sophisticated user to select one of their choice.
- Closer to testnet, we will provide more guidance but fees can be paid either via a potential protocol enshrined token ("fee juice") or via paymasters. A wallet should integrate with both.
- It could show an opinionated paymaster and/or show multiple different ones(or redirect to a website with sufficient information) to allow a sophisticated user to select one of their choice.
- In-wallet tab for private or public token transfers
- Work with the ecosystem to develop a QR Code like protocol to facilitate easy private transferring of tokens from one address to another.
- Since a user might not have their account contract deployed, just having a recipient's address may not be enough. The sender might need to have their public keys already to encrypt data to them. The QR code must include these too.
- Since a user might not have their account contract deployed, just having a recipient's address may not be enough. The sender might need to have their public keys already to encrypt data to them. The QR code must include these too.
- Authwit support tab
- Bonus: Making authwits human readable. Authwits are hashes. Work with Aztec Labs on an EIP712-like mechanism or something more suitable/better
- Bonus: Making authwits human readable. Authwits are hashes. Work with Aztec Labs on an EIP712-like mechanism or something more suitable/better
- Ability to seamlessly register contracts and recipients
- Ability to let a user batch several transactions (the batch is submitted as a app_payload to the account contract's entrypoint() method).
- Show
- block currently synced to (for showing which notes have been discovered)
- My public keys
- My notes and public state
- Any other relevant data in the PXE database (could be hidden under some advanced section) e.g. current authwits, capsules, notes, nullifiers, registered contracts and recipients
- block currently synced to (for showing which notes have been discovered)
- My public keys
- My notes and public state

- Any other relevant data in the PXE database (could be hidden under some advanced section) e.g. current authwits, capsules, notes, nullifiers, registered contracts and recipients

Please note that, Aztec is rapidly under development and we are redesigning some of our keys and note discovery specifications. While this may have some changes for wallet builders, we are aiming for minimal breaking changes.

Reference Links:

Accounts on Aztec: Overview

AuthWits

Keys on Aztec

Understanding Fees

Understanding Contract Deployments

Writing Account contract

Example account contracts in our monorepo:

- ECDSA - scep256r1 Account Contract
- Schnorr Account Contract

Wallet References:

- Cli wallet code
- CLI Wallet docs
- Old Browser based wallet built by in-house Aztec engineer @alexghr
- Shoutout also to @oleh for his passkey based wallet integrated with wallet connect:https://wallet.shieldswap.org/

FAQ

Q: Is Aztec EVM compatible?

A: No! We have a custom VM and our smart contract framework - aztec.nr that allow us to transform the privacy blockchain experience.

Q: If the testnet is not launched yet, where will I test and build?

Sandbox and devnet!

Q: I saw you have a devnet live but it is gated. Will I get access to Devnet?

A: Yes! Successful applicants will get access to Devnet and be connected with the rest of the Devnet ecosystem group. That said, for day-to-day development, we recommend trying out the Aztec Sandbox.

Q: Do I need to have ALL the requirements done by the Testnet deadline?

A: We expect you to have a minimum working wallet by testnet (so users can use it!), but you don't have to complete all the above requirements for it. We'd like to see a roadmap in your proposal dictating how you will prioritize the features and estimated timelines. We expect at least one milestone to be after the testnet/december deadline. You will also be in constant touch with Aztec Labs so that we can be sure we unblock you and that you move fast, deploy a wallet and iterate.

Q: If selected, can we meet the team IRL?

A: We have an office in London (our HQ) and are always open to host people to work from here!

Appendix 1 - Other Potential Areas of Interest

This section explores other things tangential to awallet, if you are still reading this document and don't want to build a wallet itself but think about the peripheries around it. In order of importance to Aztec Network:

- Contract verification tooling

: Aztec supports both private and public functions. For public functions, the bytecode is stored on a merkle tree (like in

ethereum) and this tool could help the code could be distributed via a block explorer. For private functions, only a commitment to the bytecode is stored onchain as opposed to the entire bytecode. While our circuits check the commitment matches the actual bytecode the app is serving, there is room for a similar UI based tool even for private contracts.

- Account Contract 2FA

: Account Countracts let you create custom security features like requiring a 2FA before sending your transaction. You could create a library or demo app to integrate zkEmail or another similar 2FA for this service. ZkEmail in Noir can be found here and a demo app using it is here, built by Oleh

- Education material on Aztec for onboarding

: Unlike Ethereum, privacy networks like Aztec are quite unique with several custom features. A new user of Aztec might need to know some (but not all) of these to best use Aztec. This could be education on our several public-private key pairs or understanding about fees on Aztec.

- Key Management best practises

: Given we have several keys and a PXE, best practises might be more involved. An in-depth community driven research would be extremely helpful for Aztec users and wallet builders. We have a dedicated signing key, a nullifier key (to spend your notes), an incoming viewing key (to view any notes or logs that were sent to you), an outgoing viewing key (to view any logs or notes you sent to another entity) etc.

DISCLAIMER