



Aztec Whitepaper

AZTEC

Aztec Team
aztec.network

Abstract

We introduce the Aztec Network, the privacy-first Layer 2 on Ethereum. By utilizing state-of-the-art zero-knowledge cryptographic protocols, the Aztec Network enables a fully programmable smart contract environment with an encrypted state ledger, without sacrificing any of the traditional security properties of distributed ledgers.

The forced transparency of existing blockchains is a fundamental barrier to real-world adoption. Programmable privacy gives developers control over the visibility of data and code; without this, blockchains are simply not feature complete. By introducing private state and private computation, smart contracts on the Aztec Network can incorporate identity and compliance procedures, and utilise attestations generated from existing products and infrastructure in Web2. Through cryptography, a new design space is opened up where sensitive data never leaves a user's device and instead zero-knowledge attestation proofs are shared and utilised. Not only can blockchains be a source of digital cash, they can now be canonical sources of credentials. The Aztec Network, designed from the ground up around a universal programmable cryptography language called Noir, utilises a hybrid state model that enables applications that manage both public and encrypted data. Fast client-side proving ensures users always maintain full control over who has access to their private information. These abstraction layers, along with a privacy-first protocol design, provide an idiomatic way to build and deploy composable, privacy-preserving blockchain applications. With Aztec Network, developers have the tools to finally fulfill the promise of blockchain.

[1 Introduction](#)

[1.1 History](#)

[2 Client-Side Proving](#)

[2.1 Client-Side Proving in Aztec](#)

[2.1.1 Private EXecution Environment](#)

[2.1.2 Limitations and future work](#)

[3 Public and Private State](#)

[3.1 The Hybrid State Model](#)

[3.2 Public State](#)

[3.3 Private State](#)

[3.4 Private Logs](#)

[3.4.1 Log Discovery](#)

[3.4.1.1 Brute-Force Trial Decryption](#)

[3.4.1.2 Out-of-Protocol Message Passing](#)

3.4.1.3	Tagging
3.5	Partial Notes
3.6	Abstracting Notes
4	Contracts and Programmability
4.1	Execution Flow
4.1.1	Private Functions
4.1.2	Public Functions
4.1.3	Kernel Circuits
4.1.4	AVM
4.1.5	Rollup Circuits
4.2	Contract Classes and Instances
4.2.1	Classes
4.2.2	Instances
4.3	Fees
4.3.1	Fee Payment Contracts
4.3.2	Node operator costs
4.3.3	Mana
4.3.4	Base fee per mana
4.3.5	Fee distribution
4.3.6	Block rewards
4.3.7	Multi-proofs and reward boosts
4.4	Accounts
4.4.1	Nonces and replay protection
4.4.2	Addresses
4.4.3	Non-user-owned accounts
4.4.4	Account initialization
4.4.5	Denial of Service (DOS) attack prevention
4.5	Noir
4.6	Aztec.nr
5	Cryptography
6	Cross-chain messaging
6.1	Portals
7	The Aztec Token
7.1	Distribution
7.1.1	Token Sale (21.96%)
7.1.2	Ecosystem Grants (10.73%)
7.1.3	Future Incentives (4.88%)
7.1.4	Foundation (11.71%)
7.1.5	Insiders (48.31%)
7.1.5.1	Team (21.06%)
7.1.5.2	Investors (27.25%)
7.1.6	Year 1 Network Rewards (2.41%)
7.2	Tokenomics

[8 Decentralization](#)

[8.1 Network Roles](#)

[8.1.1 L1 Contracts](#)

[8.1.1.1 Rollup contracts](#)

[8.1.1.2 Governance contracts](#)

[8.1.1.3 Token and Reward contracts](#)

[8.1.2 Sequencers](#)

[8.1.3 Provers and Prover Marketplaces](#)

[8.2 Governance](#)

[8.2.1 Portals](#)

[8.3 Data Availability](#)

[8.4 Staking](#)

[8.4.1 The Governance Staking Escrow Contract \(GSE\)](#)

[8.5 Block Production](#)

[8.5.1 Pending Chain](#)

[8.5.2 Proven Chain](#)

[8.5.3 Escape Hatch](#)

[8.6 Slashing and Security](#)

[8.6.1 What is a vote?](#)

[8.6.2 The Slashing Vetoer](#)

[9 Launch and Timelines](#)

[9.1 Testnet](#)

[9.1.1 Public testnet](#)

[9.1.2 Adversarial Testnets](#)

[9.2 Ignition](#)

[9.3 Alpha](#)

[9.4 Mainnet \(Beta\)](#)

[9.5 Mainnet 1.0](#)

[10 The Foundation](#)

1 Introduction

Most blockchains are fully transparent global ledgers. For Bitcoin (2008) and Ethereum (2014), this was seen as a feature, not a bug. Both launched after the financial crisis (2008) had irrevocably shaken people's trust in supposedly expertly run systems, and maximizing transparency hit a cultural nerve.

For all the idealism, funding, and effort of the last decade, the impact of blockchain on global finance, commerce, and computation has remained muted. In 2025, it is still a technology with transformational potential, but too little real and meaningful adoption. We believe blockchains have failed to achieve product market fit because they are not

feature complete. What was assumed to be their defining feature – enforced transparency – is in fact a poison pill, preventing widespread use. The importance of privacy was already obvious to Satoshi, who was well aware of the limitations in relying solely on pseudoanonymity, but in 2010 practical solutions were out of reach.

The Internet had to go through a similar rude awakening in 1990's. Early adopters flocked to places like public message boards, and saw the future. Adoption by enthusiasts was growing, but 'e-commerce', which was understood at the time to be the largest commercial opportunity, was stalled. Users were unwilling to put their credit card information into unencrypted web forms. Businesses, understanding this reluctance, were waiting on the sidelines. A permissionless, public network lacked the features needed to evolve into a platform for finance and commerce. That early version of the internet - full of potential but not yet feature complete - is where blockchain sits today.

Before Amazon or eBay, Gmail or online banking, the internet needed a feature upgrade - functionality that would make it safe for consumers to transact and attractive for businesses to migrate services online. The unlock for the early Internet was the release of SSL (Secure Sockets Layer) 2.0 by Netscape in 1995. HTTPS followed soon after. These two cryptographic infrastructure innovations - SSL and HTTPS - unleashed the commercialization era of the Internet.

It wasn't fast, and it wasn't inevitable. SSL 1.0 (1994) was so flawed that Netscape decided not to release it, taking another year to perfect and ship 2.0. It took another 15 years for SSL/HTTPS to become the expected default, but along the way it transformed the internet and unlocked trillions of dollars of value.

Aztec's programmable privacy is SSL for blockchains, finally making blockchains feature complete.

1.1 History

We started Aztec to bring real world assets to blockchains. In 2017, we pitched private debt funds on a product to issue and seamlessly trade corporate debt on Ethereum. We encountered a predictable roadblock: no traditional financial institution wanted their entire portfolio, every trade, and every commercial term immutably published on a global and transparent ledger. Without privacy, we simply could not go to market.

Since this realization, we have been entirely focused on producing the tools the ecosystem lacked. This has forced us to repeatedly push the state-of-the-art across both fundamental research and in-production engineering. With every iteration, we have aimed to increase the practical usefulness of the software and cryptographic protocols we've built, with a focus on generality, speed, and security.

In November 2018, we released the first proof of concept for private token transfers on Ethereum Mainnet¹. A year later, we released it to production. In order to launch, we successfully coordinated Ignition², at the time the largest ever MPC trusted setup ceremony. After an initial contribution from Vitalik Buterin, another 175 participants joined the computation chain. We released the result as a public good.

In 2019, we released PLONK³, a SNARK-based proving system that is computationally inexpensive, and only requires one universal trusted setup. At the time, PLONK was the most performant universal SNARK, and is still the bedrock for the majority of SNARK-based protocols in Web3 today. PLONK helped us solve two problems which had plagued our first protocol: though cheap, it was still much more expensive than transparent transactions, and its cheapness came from a very limited ability for developers to shape it to their needs. PLONK allowed us to build recursive SNARKs (our first open-source implementation released in April of 2020), which unlocked arbitrary zero-knowledge proofs computed on user's machines to be aggregated and posted onchain cheaply, as a batch.

We introduced the first two products taking advantage of these new tools in October 2020: Aztec 2.0, a private payment Layer 2 (L2) enabling zcash-style privacy, and Noir, our zkSNARK DSL. 6 months later, in March 2021, we released zk.money, an easy web interface to Aztec 2.0, to Ethereum Mainnet. Finally, in June 2022, we released Aztec Connect, which helped hundreds of thousands of Ethereum users interact with DeFi without inadvertently revealing private personal data.

Each release required an incredible amount of effort from a highly skilled team of cryptographers and engineers. In pursuit of smooth, fast, and easy-to-use products, we had to hand craft zero-knowledge circuits, and assembly routines. Despite our attempts to provide abstractions and higher level APIs, external developers wanting to extend our platforms often had these very same demands placed on them, unacceptable given our goals.

In 2023, we decided that building a fully general and easy to use privacy platform required our undivided attention. This meant sunseting all other products, and aggressively pursuing a goal orders of magnitude more ambitious than those which came before it.

The Aztec Network represents the culmination of all of our work - a zk-SNARK based privacy and scaling L2 built on Ethereum, with general programmability powered by Noir and Aztec.nr, our open-source programming language and smart-contract framework, respectively. Together, they give developers access to the platform we wished we had access to in 2017. Programmable privacy, without having to become a

¹ medium.com

² <https://aztec.network/blog/aztec-crs-the-biggest-mpc-setup-in-history-has-successfully-finished>

³ <https://eprint.iacr.org/2019/953>

cryptographer. Composable smart contracts, without having to reveal your secrets to the world.

It gives regulated financial institutions the ability to gate access to any of their onchain products based on sensitive, private user information (like name, age, nationality, net-worth); permissionless and user-owned games which make use of asymmetric information, as needed by Poker or Bridge; activist organizations the ability to facilitate legitimate shadow elections in oppressive regimes, without risking the safety of voters; users the ability to use onchain finance without putting themselves at risk of targeted scams or violence based on fully visible sensitive personal information.

The Aztec Network is built to be used by a wide variety of stakeholders, who will naturally have diverging expectations and needs. As a result, we and the broader Aztec community have designed and built it to be maximally neutral and transparent at the protocol level, while giving users strong control and privacy. The network will be decentralized from day 1.

With the Aztec Network, we aim to finally fulfill the promise of blockchains.

2 Client-Side Proving

Blockchains are *consensus networks*. Their purpose is to ensure the latest state of the network is recognized as such by all participants, no matter how little trust exists between them.

Critical to this goal is ensuring that the rules of the network have been correctly and uniformly enforced. On almost all blockchains today, this requires delegating all ability to modify the state of the network to the set of nodes which make it up. A user who wants to update some state that they own (in order to transfer funds to another user, or vote in a DAO, for example) will need to submit an authenticated transaction to the network detailing the set of functions they want to call. It is then up to nodes in the network (usually all of them, to prevent malicious nodes from lying) to run these functions correctly, and keep track of any resulting valid state changes.

This model creates a set of constraints which ultimately compromise the usefulness of blockchains: first, it creates a need for extreme transparency, since any and all nodes need to be able to read and act on your data on your behalf. Second, to avoid the risk of Denial of Service (DOS) attacks and because all nodes in the network will need to re-run every function call, all computation must be metered and extremely limited in complexity.

Programmable cryptography helps us move past this local equilibrium. Zero-knowledge validity proofs give users the ability to run computations over their data locally, and subsequently prove to the network that the resulting state changes are valid. Nodes

gain the same guarantees as in traditional blockchains, but they do not need visibility into the data, or even which functions specifically were executed. This is the foundational building block on which the Aztec Network is built: zero-knowledge validity proofs ensure users can benefit from consensus over the latest valid state of the network, without forcing them to reveal their data or which smart contracts they are using to the world.

2.1 Client-Side Proving in Aztec

Aztec makes use of zero-knowledge validity proofs in two contexts. Like most validity proof rollups, nodes on the Aztec network use them to batch user transactions and post them to Ethereum Layer 1 (L1). What sets Aztec apart, however, is their use by end-users, directly on their laptop or phone.

While the former is an important and common tool to make L1 settlement cheaper, the latter makes it possible for users to privately execute smart contract code, locally. The network only receives a proof of correct execution, revealing nothing of what code was executed by whom, over what data.

Unlike applications built on public blockchains, which permanently force every user interaction into the public record, Aztec is built to help users regain control over their digital footprint. Client-side proving and native protocol support for encrypted state enables composable, flexible, and programmable privacy.

2.1.1 Private EXecution Environment

Local computation over encrypted data must happen in a context where that data can first be decrypted, i.e., in an environment with ready access to relevant secrets. At minimum, this means that the computation should happen on a user's own device.

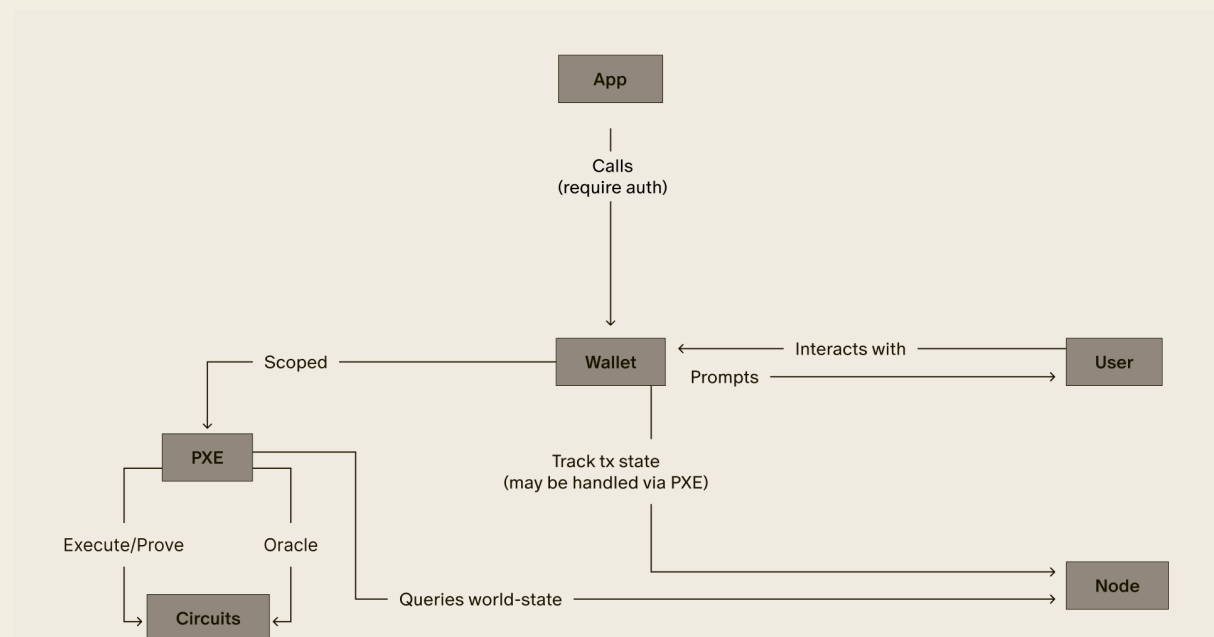
Given the permissionless and composable nature of Aztec, any app should be able to ask the user to make use of this potentially sensitive data, without requiring the user to independently investigate if their private information is going to be misused or insufficiently protected. This is why we have developed PXE (pronounced 'pixie'), a.k.a. **Private EXecution Environment**.

The PXE is an open source, client-side library for the execution of private operations. It is intended to be run inside wallet software, and requires no special hardware.

The PXE maintains a database of the user's master secret keys (nullifier and viewing keys) and private state. For security, it does not divulge master secrets to any smart contract. If a smart contract requires access to a secret key (for example, to compute certain kinds of nullifier), the PXE will "silo" the key – creating a secret key specifically for use only within that smart contract – using a scheme inspired by BIP-32 hierarchical

keys. Similarly, the PXE does not divulge the private state of one smart contract to another smart contract without explicit authorisation from the user. This prevents malicious contracts from leaking a user’s data. All private state that ‘belongs’ to a particular contract is “siloe” by the protocol and is stored in siloes within the PXE’s database.

The PXE (leveraging Kernel Circuits ([section 4.1.3](#))) generates proofs of private function execution, and sends these proofs along with public function requests to the Sequencer ([section 8.1.2](#)). The details of a user’s transaction (their decrypted inputs and outputs, which private functions were called) never leave the client-side PXE.



2.1.2 Limitations and future work

In SoK: Programmable Privacy in Distributed Systems⁴, Benarroch, Gillespie, Lai, and Miller introduce a useful framework to speak about the potential and limitations of various approaches to programmable privacy. They classify techniques into three computational contexts: independent, mediated, and global.

Zero-knowledge validity proofs are most useful in an “independent” context. In other words, they are most useful for situations where the computation is over some combination of public data (i.e., data accessible by everyone) and personal private data (data that the user has full access to). If a computation requires as an input private data of other users which cannot be shared, zero-knowledge validity proofs are not sufficient.

Thankfully, it is possible to extend systems like Aztec using other programmable privacy techniques in order to gain the ability to compute over shared private state. Trusted

⁴ <https://eprint.iacr.org/2024/982.pdf>

Execution Environments, Multi Party Computation, or Fully Homomorphic Encryption can be leveraged out of the protocol.

We are particularly excited about the progress made in coSnarks i.e., the collaborative computation of zero-knowledge validity proofs. Notably, teams like TACEO have already extended Noir ([section 4.5](#)) with support for collaborative proving, and we expect adding support within Aztec during the course of Alpha, subject to technical readiness ([section 9.3](#)).

3 Public and Private State

Aztec implements a hybrid state model that combines public (transparent) state with private (encrypted) state within a unified system. This architecture enables developers to choose which data should be public and which should remain private, while preserving the core properties that make blockchains valuable: consensus, composability, permissionlessness, and programmatic verification. Aztec is designed to provide privacy for the user, while enabling transparency for the protocol.

3.1 The Hybrid State Model

Aztec's state model reflects that different applications and users have different privacy requirements. The architecture allows developers to specify which data should be public or private, along with the rules governing how this data can be accessed and modified.

Smart contracts can have both public and private states, and a user can read and write from both within a single transaction.

3.2 Public State

Public state in Aztec follows established patterns from Ethereum's account-based model, providing developers with familiar semantics. In particular, specific data lives in a consistent storage slot over time, and that slot can be mutable.

All public state is stored within a single [indexed Merkle tree](#). To ensure data is appropriately scoped to a contract, storage slots are computed as:

$$real_storage_slot = H(contract_address, logical_storage_slot)$$

3.3 Private State

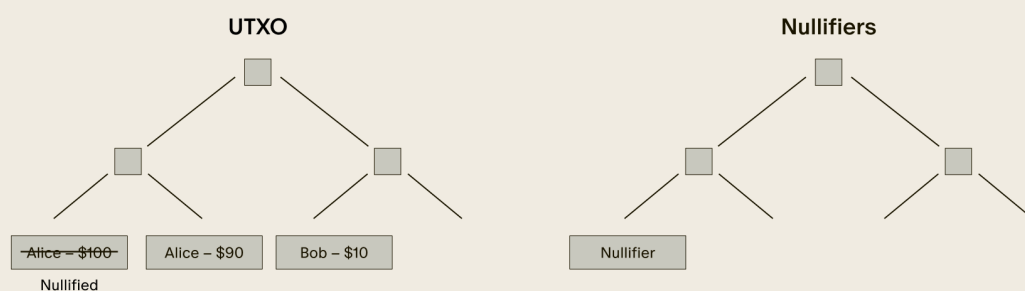
Unfortunately, simply adding a layer of encryption on top of an architecture designed for public state does not give developers or users useful private state.

Developers need to be able to explicitly define precise rules over who has access to what data, and users need strong guarantees that these rules can not unexpectedly change or weaken over time. By design, account-based models create trails of public metadata about who modified what data and when. This flow of information is impractical to appropriately manage, and the expectations of both developers and users will inevitably get broken as a result.

Concretely: since it is possible to observe which storage slot was read or modified by a transaction in account-based models, encrypting the contents of the slot provides only weak and temporary privacy. An unauthorised observer can still easily and without detection accumulate enough information over time to come to statistically-informed conclusions about what an encrypted transaction relates to, and who the transacting entities are. This requires only basic statistical tools, which are becoming ever easier to use by powerful and widely accessible AI tools.

These worst-of-both-worlds systems create complexity for users and developers while leaking enough information that bad actors can prevail. Stalkers can track victims through their payment patterns, companies can map their competitors' relationships via onchain contract execution, and hostile regimes can expose activists who might use the chain for privacy preserving voting. These risks ultimately prevent real world adoption.

As a result, the network has been designed to minimize the amount of information inadvertently leaked by onchain interactions with private state. Users are still free to reveal data they control to whoever they see fit (through zero-knowledge attestations, or by sharing viewing keys), but we take the view that the network should minimize the information shared implicitly or accidentally.



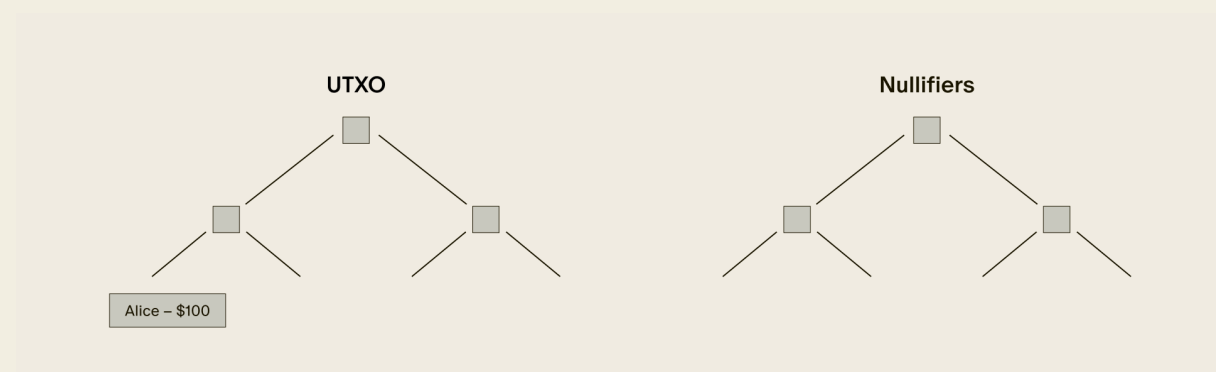
In practice, Aztec implements private state as two append-only databases, in a model similar to the one pioneered by Zerocash⁵ and first implemented at scale by Zcash. State is represented via single-use state objects that are a generalisation of the UTXO (Unspent Transaction Output) model used in Bitcoin. In Aztec, these objects are called **Notes** and, unlike Bitcoin or Zcash, where UTXOs represent cryptocurrency values, Aztec Notes can represent arbitrary, smart-contract-defined data:

⁵ <https://ieeexplore.ieee.org/document/6547123>

1. **Note Hash Tree:** Stores Notes. Each Note is created by hashing the relevant data into a single 254-bit prime field element (along with some randomness). Notes are the fundamental primitive for private state, enabling arbitrary encrypted data to be stored and referenced.
2. **Nullifier Tree:** An indexed Merkle tree storing cryptographic commitments that correspond to destroyed Notes. Each nullifier is a commitment that marks a Note as consumed, but crucially, only the Note owner is aware of the correspondence between Notes and nullifiers. For everyone else, it's mathematically intractable to determine which nullifier corresponds to which Note. Only the owner can nullify a note using their nullifying key.

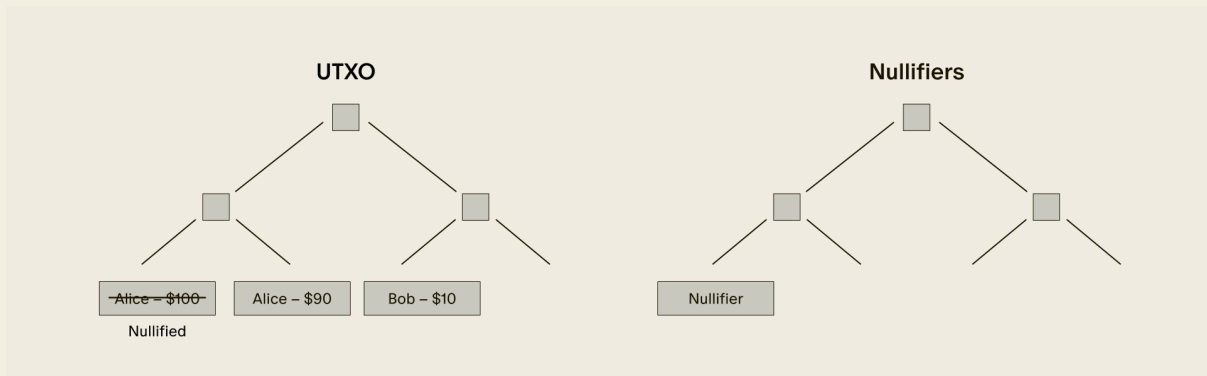
Similar to Zcash's UTXO model, private state updates happen through the creation and nullification of notes⁶, unlike account-based systems where state can be directly modified. The goal of this design is to make all state reads and writes look indistinguishable from the perspective of an outside observer.

By way of example, let's work through a simple transfer of a hypothetical token deployed on Aztec. Alice has \$100 stored in a Note, and wants to transfer \$10 to Bob.



When Alice makes a transfer of \$10, she destroys (or “nullifies”) her \$100 note. In other words, she creates a nullifier corresponding to her particular \$100 note. Only she will know that this nullifier corresponds to the \$100 note. She also creates a \$10 note for Bob and a note for herself reflecting her updated balance of \$90. The original \$100 note can never be used again.

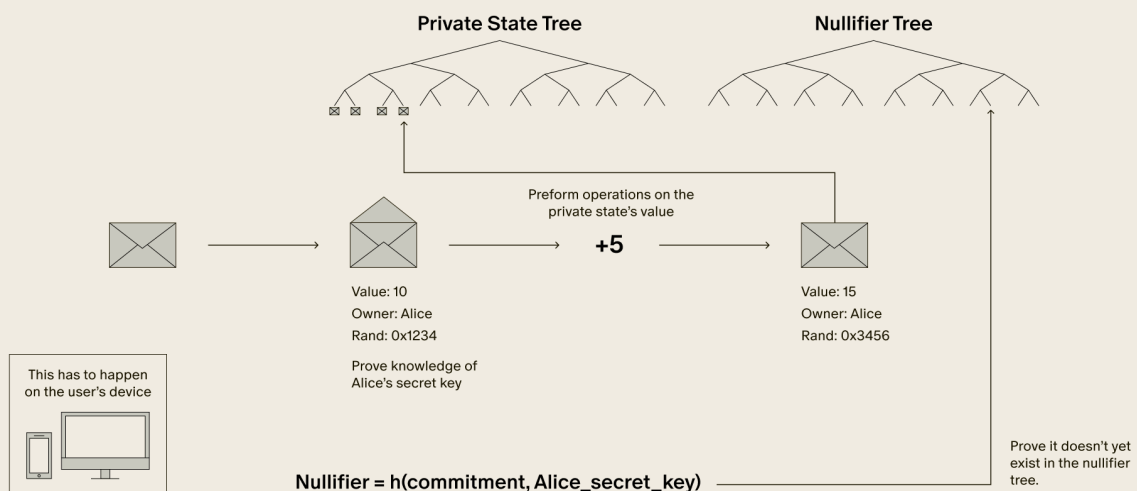
⁶ Unlike Zcash and Bitcoin, Notes on Aztec can store arbitrary data.



Since all notes are encrypted, Bob only sees *his* Note's value. Notes created and nullified by Alice, and notes created and nullified by other transactions on the network look indistinguishable to him (and to everyone else).

STATE CHANGES WITHIN THE PRIVATE REALM

Prove it exists somewhere in the tree



A nullifier is a cryptographic commitment corresponding to a destroyed Note, computed using deterministic secret values known only to authorized parties. While the network can verify that each nullifier is valid and has never been used before, the relationship between a nullifier and its Note remains private. Even when observing that a nullifier has been published (indicating some Note was destroyed) and new Notes have been created, it's computationally intractable to determine *which* specific Notes were involved in the transaction.

This pattern extends beyond simple value transfers. Any state can be represented with Notes.

Moreover, nullifiers extend beyond simple “double-spend prevention”: a nullifier can be more-generally considered as evidence that a one-time action has taken place, without

revealing *which* action. For example, nullifiers can be used as evidence that a contract has been deployed, or that some state has been initialized, or to signal an action in Semaphore⁷-like applications built on Aztec.

Unlike public state, only hashes of private state notes are stored onchain. The user's PXE is responsible for maintaining a database of the underlying private data. In most cases, a user can reconstruct their private state from Private Logs emitted by relevant smart contracts.

3.4 Private Logs

Any private function of any smart contract on Aztec can emit private logs. Their purpose is to broadcast encrypted data. A private log cannot be read by everyone; it can only be decrypted and read by the intended recipient. To the rest of the world, a private log will just look like a random number.

These logs are a source of truth that a user's PXE can use if it needs to reconstruct private state, and they are by default posted to Ethereum L1 blobs.

Aztec smart contract developers can define the layout and content of the message, the encryption scheme, and even how the shared secrets used to decrypt the message are derived. Importantly, all of these decisions are enforced by the smart contract, so developers know that all rules they set will be followed by all users. This makes it possible for asset issuers to build in flexible visibility controls, when relevant.

Each transaction can include around 16KB of private logs, though this may change during future upgrades.

Two common uses for Private Logs are privately communicating information about a Note to its ultimate owner, and privately communicating that a function has been called. [Aztec.nr](https://aztec.nr) provides useful but optional abstractions in the form of Note Logs and Event Logs for these.

Private logs must always be created with a recipient (or set of recipients) in mind, since they must be encrypted to a specific cryptographic key. Private logs should also not leak information about *who* the intended recipient is. This creates an issue: by their nature, private logs must look indistinguishable from one another, but a recipient must be able to find logs relevant to them in particular.

3.4.1 Log Discovery

Aztec does not enshrine a particular private log discovery mechanism into the protocol. Instead, developers are free to adopt any scheme of their design or choosing.

⁷ <https://semaphore.pse.dev/>

3.4.1.1 Brute-Force Trial Decryption

The simplest solution, used for example by Zcash, is for users to simply brute force their way through every log emitted and attempt to decrypt each with their key. Unfortunately, this approach doesn't scale well. After many years of blockchain use, users may simply be unable to download and process all private logs on consumer devices in a reasonable time frame. A minimal private transaction emits 570 bytes of encrypted data. At 10 transactions per second, a node would have to consume and trial-decrypt over 150GB of data to sync 1 years' worth of transactions, which is not practical.

3.4.1.2 Out-of-Protocol Message Passing

A user can be informed that a log is relevant to them out-of-protocol. This can be facilitated in a peer-to-peer fashion, for users who have existing open communication channels, or it can be facilitated by an app or service provider (though this can mean a third party potentially gaining enough information to statistically de-anonymise a user).

3.4.1.3 Tagging

Our goal with the Aztec Network is to provide neutral, permissionless, and reliable infrastructure. In that context, brute-force trial decryption risks making the network unusable in practice, while out-of-protocol message passing is burdensome at best, and at worst will create reliance on potentially censorious or unreliable third parties.

This is why we designed a third option for developers, built-in to Aztec.nr ([section 4.5](#)): Tagging.

In short, Tags are 254-bit field elements which can be prepended to any private log. They enable the sender to efficiently convey in-protocol to the intended recipient that a log is relevant, without outside observers gaining any information.

Before Tags can be computed and used, the sender and recipient must agree on a shared secret. In practice, this can be done at any point as long as both know each other's Aztec address. We call this step the "handshake", and it only needs to happen once.

With this shared secret, both the sender and the recipient can deterministically compute a sequence of unique Tags. Each Tag should only be used once, and the users' PXEs ([section 2.1.1](#)) can always look for private logs tagged with the next N unused Tags.

In other words: compared to out-of-protocol message passing ([section 3.4.1.2](#)), which would require each log to be separately shared out-of-protocol, Tagging gives users the ability to communicate only in-protocol as long as they have been able to establish each other's addresses at least once.

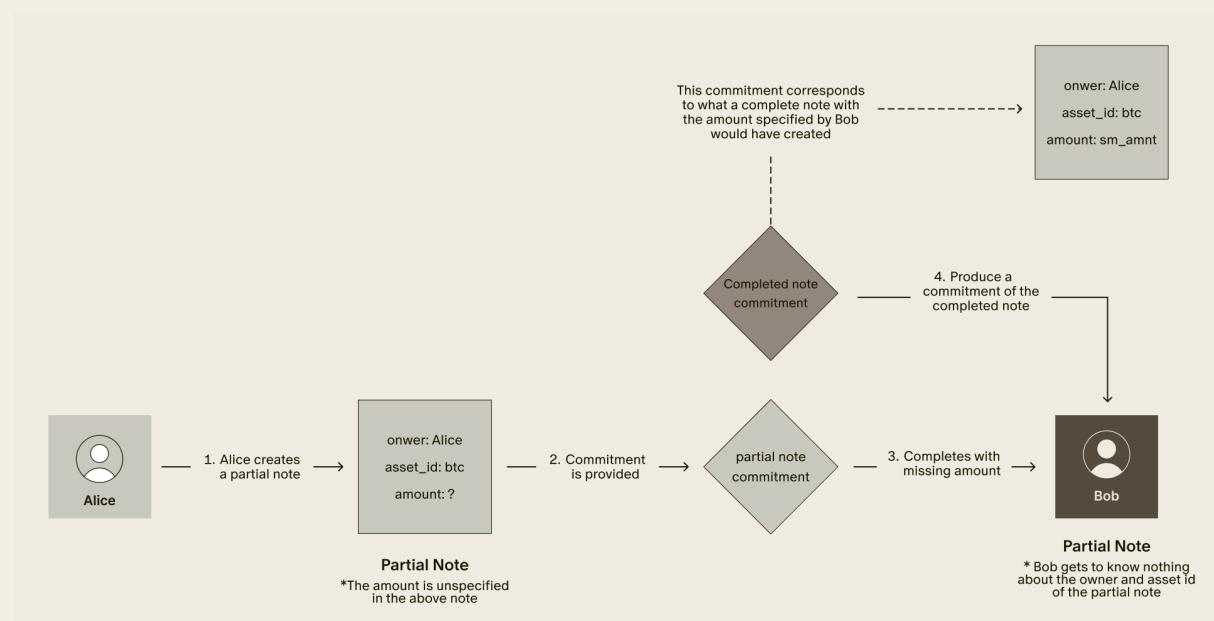
Importantly, users can always fall back to brute-force trial decryption.

One important downside of Tagging is the information it reveals to full nodes. A user's request to a full node for a particular set of tags, when combined with other metadata like request timing and IP address, could be used to statistically deanonymize onchain data. This can be resolved out-of-protocol, including through the use of Private Information Retrieval techniques or mixnets.

3.5 Partial Notes

It is common for blockchain transactions to depend on network state that will only be known when the transaction is included in a block, such as gas prices, token exchange rates, or oracle data. In traditional transparent systems, nodes can read and incorporate this information at execution time, enabling simple and efficient state updates. However, privacy-preserving protocols face a challenge: since nodes cannot inspect encrypted values, multiple transactions are needed to handle such dependencies, first committing to maximum values, then initiating separate transactions to claim refunds or adjustments. This adds latency, costs, and complexity to what should be straightforward operations.

Partial notes provide an elegant solution through careful application of elliptic curve properties. By leveraging both the additive homomorphism of curve points and the one-way nature of point multiplication, they enable users to commit to initial encrypted values while granting designated counterparties the ability to modify those values without accessing their contents. This primitive enables protocols to resolve state dependencies in a single transaction while maintaining cryptographic privacy guarantees.



3.6 Abstracting Notes

Aztec’s private state model brings privacy to users, but risks adding unfamiliarity and complexity for developers. Noir, Aztec.nr, and the PXE are designed to provide an idiomatic and clear abstraction layer for Notes, to manage this risk.

```
/// @param name The name of the token
/// @param symbol The symbol of the token
/// @param decimals The number of decimals of the token
/// @param private_balances The private balances of the token
/// @param total_supply The total supply of the token
/// @param public_balances The public balances of the token
/// @param minter The account permissioned to mint the token
#[storage]
struct Storage<Context> {
    // metadata
    name: PublicImmutable<FieldCompressedString, Context>,
    symbol: PublicImmutable<FieldCompressedString, Context>,
    decimals: PublicImmutable<u8, Context>,

    // private balances (notes)
    private_balances: Map<AztecAddress, BalanceSet<Context>, Context>,

    // public balances + total supply
    public_balances: Map<AztecAddress, PublicMutable<u128, Context>, Context>,
    total_supply: PublicMutable<u128, Context>,

    // optional mint control (present in the "mintable" variant)
    minter: PublicImmutable<AztecAddress, Context>,
}
```

An example definition of defining private balances (‘balances’) and public balances (‘public_balances’) in an Aztec smart contract.

The differences between the public and private state models should nevertheless not be completely hidden, especially in the context of smart contracts. Key assumptions that a developer can safely make, when working with public state, will simply not hold when working with private state. A private function cannot enforce that a collection of Notes used by a user represents that user’s total balance, for example. They may simply have selected a subset of all relevant Notes.

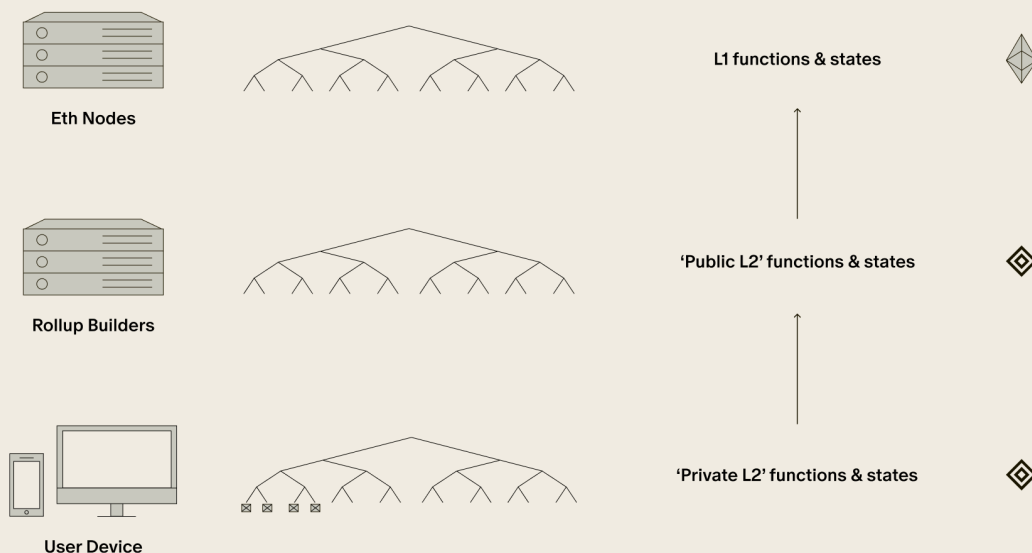
Handling UTXO-like state-models presents inherent challenges, particularly in a programmable environment where notes represent arbitrary state whose function and purpose are defined by a smart contract. The protocol tackles this by enabling contract writers to define “getter” functions in their contracts - code that is not constrained and is instead executed natively by the PXE. In this model, the PXE is the entity that finds and aggregates Notes, manages syncing with the latest blockchain state and simplifies routine operations like the consolidation of Notes.

The intended outcome is that the nuances of UTXO state are abstracted away from client-side application developers (like wallets) via the PXE.

4 Contracts and Programmability

At a high level, smart contracts on the Aztec Network are like smart contracts on other blockchains: they are arbitrary, developer-defined computer programs which set the rules and procedures which must be followed when interacting with particular data stored on the network.

Aztec supports both public and private functions in its smart contracts. Private functions are designed to operate on encrypted state, and need to be executed and proven by users on their own devices (as covered in [section 2](#)). Public functions operate on public state, and are executed and proven by Sequencers of the Aztec Network. While public functions can't compute over private state, they can be used to insert a note in the note hash tree, or complete partial notes with some public knowledge.



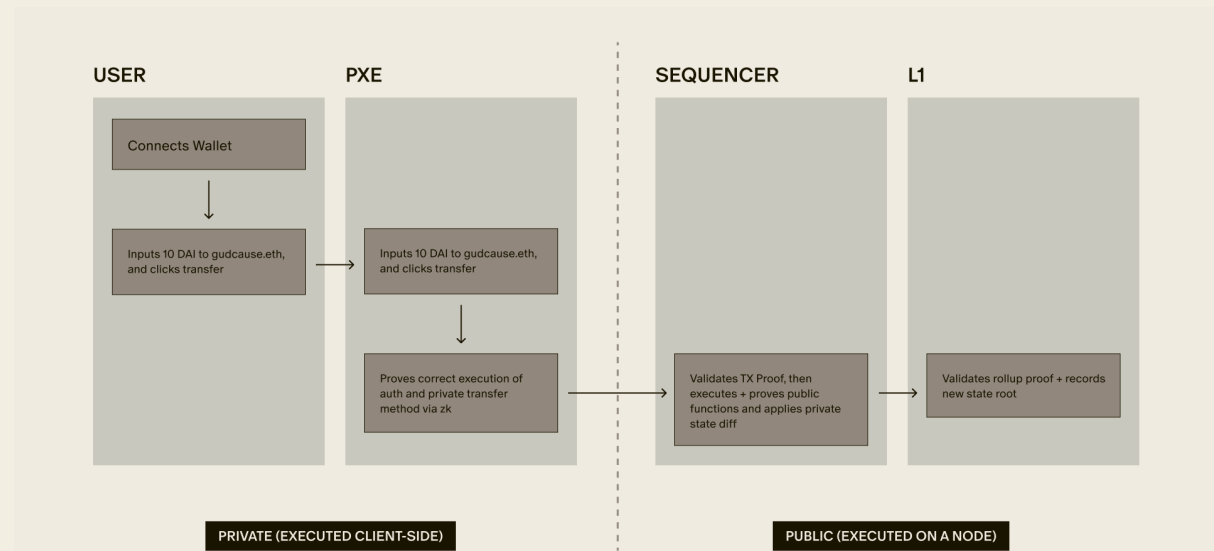
As we've seen in [section 3.3](#), Aztec aims to minimize the amount of data and metadata which gets unexpectedly and irrevocably published during normal use of the network. Accordingly, unless explicitly authorised by a user, no outside observer can know which private functions were called by who and over what data.

Public function calls, on the other hand, are fully transparent by default.

Private function calls can chain public function calls in a single transaction, but not the reverse, and users retain the full scope of composability between smart contracts they have come to expect from other blockchains. If a public function is called by a private function, the identity of the public function caller remains hidden.

4.1 Execution Flow

The execution flow for Aztec smart contracts is defined by the fact that private functions need to be executed on a user's device and all public functions are executed later by Aztec Sequencers.



As a result, private functions can enqueue calls to public functions, but any side effect or return value will be inaccessible to the private function. Public functions can't call private functions.

4.1.1 Private Functions

Defining a function as private has two high-level consequences:

1. Execution is the responsibility of the user. It is assumed (not mandated) that the user submitting the transaction will execute the private function on a device they trust, with access to private data (facilitated by the PXE)
2. In order for the associated transaction to be considered valid, it must contain a zero-knowledge proof of its valid execution (again, facilitated by the PXE)

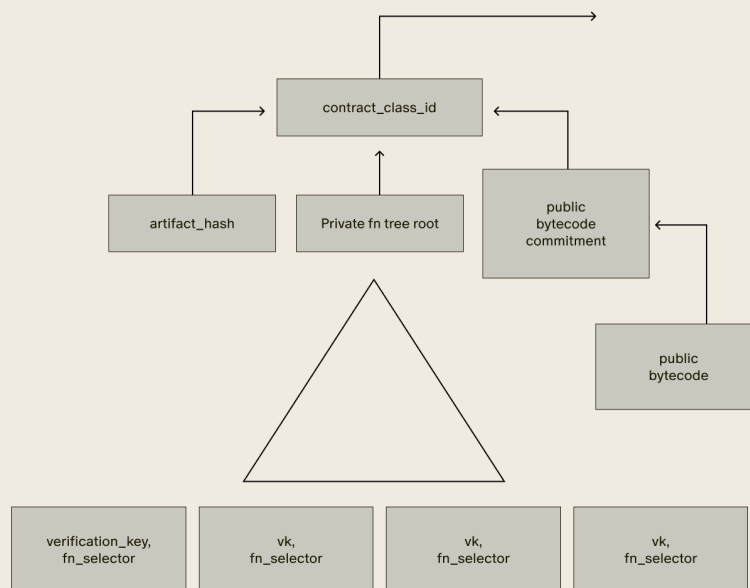
Private functions only define arbitrary state transitions. The proof which gets generated of their correct execution is "stateless": it simply guarantees that given a set of inputs, the pre-defined state transition was correctly applied. More explicitly, the proof contains, as part of its public inputs, a list of encrypted state objects and nullifiers to be created as a result of execution. Protocol-level validity rules about the requested state transitions are not validated by the function proof.

For private functions to be useful within Aztec, the above is not sufficient. Users also need to prove that they are following the rules of the protocol (for example, that the data they used as inputs to the stateless private function exists on the ledger, and that they

have the right to use it). We accomplish this by “wrapping” private function proofs in Kernel circuits ([section 4.1.3](#)). Contrary to private functions, which are developer or user defined, Kernel circuits are a core part of the protocol. Kernel circuits are also how we achieve “call-stack” privacy i.e., hiding from an unauthorised outside observer what function (and how many of them) has been called.

One of the responsibilities of the Kernel circuit is to prove that private functions being called have been registered on the Aztec Network. Each private function within a smart contract has an individual verification key, generated when a smart contract is compiled. All verification keys are components of the private function tree root for a particular contract instance, and, to be correctly registered, a reference to this contract instance must exist in the protocol nullifier tree.

Importantly, the bytecode of private functions is not stored on the Aztec Network. Instead, the function’s selector and the associated verification key are contained within the callee contract’s `pre_address` (See [section 4.4.2](#)). The combination of selector and verification key uniquely define a smart contract. This opens up the potential for proprietary and private smart contracts, with logic only shared amongst a permissioned group of users.



Private functions can call other private functions as well as public functions (though calls to public functions are not executed until later, when a transaction is included by a Sequencer). Facilitating this composability is also the Kernel circuit’s responsibility.

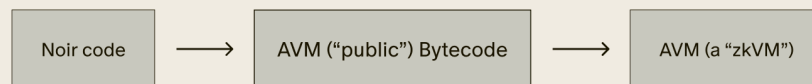
Aztec contracts and private functions can be written in Aztec.nr, a smart contract framework built on top of Noir ([section 4.5](#) and [4.6](#)), but the protocol doesn’t enshrine the use of any language. Independent teams can create alternative programming

languages, as long as high-level logic is ultimately compiled down to an appropriate circuit.

4.1.2 Public Functions

Public functions will be familiar to most developers who have worked with blockchains in the past. They get executed by Sequencers of the network, who also prove their correct execution. Any public function call is fully transparent to the network, visible to anyone.

All public functions in a smart contract are written in Aztec.nr and get compiled into a single blob of Aztec Virtual Machine (AVM) bytecode. The AVM ([section 4.1.4](#)) is Aztec’s custom provable virtual machine (commonly referred to as “zkVM”s in the industry, though the AVM, like other “zkVMs” provides no privacy). Contrary to private functions, this bytecode does need to be deployed to the Aztec Network. Since Sequencers will need to run it, it must be publicly available and is therefore stored by all Aztec nodes.



Unlike private functions, public functions don’t rely on Kernel circuits to establish that the executed logic abides by the rules of the protocol. Instead, these rules are directly embedded in the AVM.

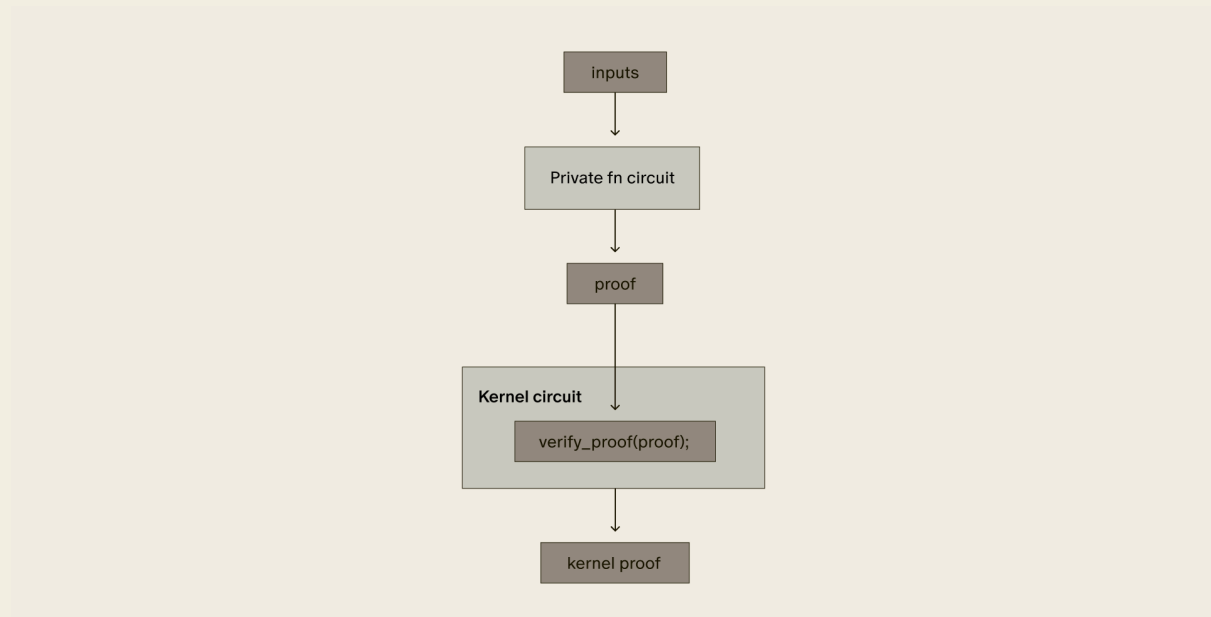
Like Ethereum, execution of public functions needs to be metered in order to avoid DOS attacks.

4.1.3 Kernel Circuits

Kernel circuits get their name from Kernels in an operating system context, and their use on blockchains originates from the 2018 “Zexe” paper⁸.

Their responsibility is to enforce that everything which happens in “userland” abides by the rules of “protocolland”.

⁸ <https://eprint.iacr.org/2018/962.pdf>



The Kernel circuit has a recursive structure where one iteration of the kernel circuit algorithm will execute a single private function call in a user’s transaction. If more than one function call is being executed, a kernel circuit must therefore also recursively verify a previous instance of the kernel circuit.

The kernel circuit operates the following data structures that collectively define the kernel **state**. Note that the kernel circuit contains two variants of the state: the input state and the return state. Both are defined as public inputs to the circuit. Kernel state consists of the following data structures:

1. A list of private function calls to be executed. Each private call contains a hash of the following:
 - a. A hash of the function’s input arguments
 - b. A hash of the function’s return values
 - c. The message sender
 - d. The contract address
 - e. The function selector
 - f. A proof of correctness
 - g. The function verification key
2. A list of public function calls to be executed
3. A list of messages to be transmitted to the L1 rollup smart contract
4. A list of encrypted events to be logged
5. A list of notes to be created
6. A list of nullifiers to be created
7. A list of note “read requests”
8. The block header of a historical block that state roots are sourced from

The kernel circuit will execute the following logic using the above data:

1. The previous kernel proof will be accumulated via a folding scheme (this is semi-analogous to recursively verifying the proof)
2. The input arguments are consistent with the return data present in the previous kernel circuit's public inputs
3. The block header is consistent with the previous kernel's block header
4. The most recent private function call is "executed" by executing the following logic:
 - a. The contract address is correctly derived from the provided verification key and function selector
 - b. The provided proof is accumulated via a folding scheme
 - c. The proof's public inputs are used to update the kernel's return state

The function proof's public inputs are interpreted according to a fixed ABI and may specify the kernel circuit add a bounded number of entries into the following data structures:

1. The note creation list
2. The nullifier creation list
3. The private function call list
4. The public function call list
5. Note read requests
6. Encrypted event logs
7. L1 message list

Note "read requests" describe a claimed note that must exist in the state tree in order for the transaction to be valid. This is validated by **either** satisfying a merkle tree membership check using the state root present in the block header, **or** identifying that the note is present in the note creation list. Note that private state is append-only - querying a historical state root is sufficient to prove existence. All read requests are executed in a single batch by a specialized **tail** kernel circuit.

The tail kernel circuit takes as an input a regular kernel circuit proof - however the return state of this proof must describe a private function call list that is **empty**, signifying that all of the transaction's private functions have been executed.

A tail kernel circuit proof is required for a transaction to be ready to be processed by a sequencer. Its public inputs define the sum of the transaction data that is visible to observers, and contains the following data structures:

1. Note hashes to be inserted into the note tree
2. Nullifiers to be inserted into the nullifier set
3. Messages to be sent from the Aztec network to the Ethereum blockchain via the Aztec rollup contract
4. Encrypted event logs to be emitted

5. A list of public functions (and their input arguments) to be executed

By way of high-level example, consider this pseudo code:

```
import Contract2;

contract Contract1 {

    private uint x;

    function1(uint a, uint b, uint c) {
        d = Contract2.function2_1(a, b);
        x += d;
        Contract2.function2_2(c, x);
    }
}

import Contract3;

contract Contract2 {

    private uint y_1;
    uint y_2;

    function2_1(uint a, uint b) {
        d = Contract3.function3_1(a, b);
        y_1 += d;
        function2_3(a);
        return d;
    }

    function2_2(uint c, uint x) {
        return c * x;
    }

    public function2_3(uint a) {
        y_2 += a;
        Contract3.function3_2();
    }
}

contract Contract3 {

    private uint z;

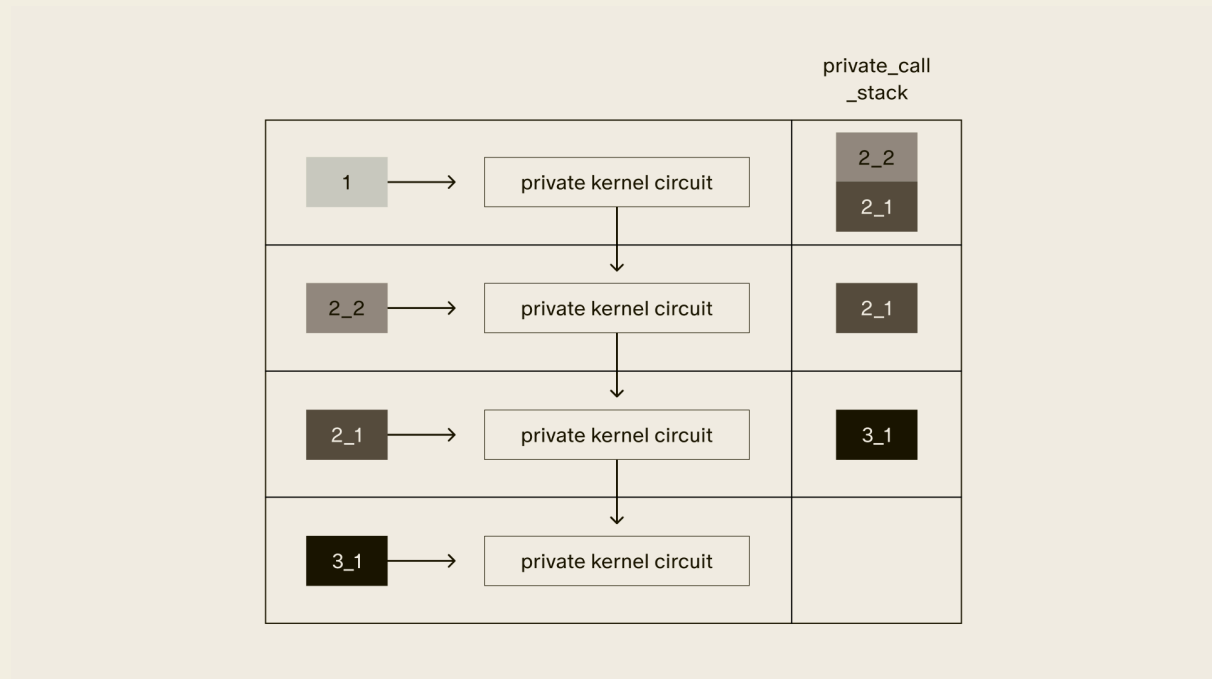
    function3_1(uint a, uint b) {
        return a * b;
    }

    public function3_2() {
        z++;
    }
}
```

Here, each private function must be proven individually. Each of these proofs will be verified by a kernel circuit, which will also verify any relevant kernel proofs of functions earlier in the call stack.

We recurse through kernel circuit proofs until the private call stack is empty, at which point the final kernel circuit emits a proof that can be rolled up by the Sequencer along with the AVM proof into the final rollup proof (which aggregates many user transactions and enforces more protocol rules that the kernel circuits can't).

It's important to highlight once again that for private functions, only the final kernel circuit proof (along with any outputs of the private computations and the public call stack) gets shared with the network. This final kernel circuit proof does not reveal which functions were executed, only that all of the rules of the protocol were followed.



4.1.4 AVM

While private execution is performed client-side and proven in a traditional ZK circuit, public execution follows a more standard Ethereum-style execution model in a Virtual Machine (VM). As with execution on Ethereum, Aztec's public execution must be performed by network participants to guarantee global, deterministic and sequential ordering of public state accesses.

While using a VM instead of a series of compiled circuits introduces some costs, it is required on a generally programmable platform like Aztec. Without one, errors during the execution of a public function cannot be distinguished from a proof which is simply invalid (i.e., a proof with unsatisfiable constraints). This would create an opportunity for a Sequencer or Prover to censor user transactions by crafting invalid witnesses, or for a user to grief the network by submitting transactions which can never be successfully proven.

Given that public execution must be performed network-side, network participants must be fairly compensated for their time and resources. Not only does that imply the need for gas/mana metering of public execution, but it also means that network participants must be compensated for transactions that error during public execution to ensure fair compensation and prevent DDoS attacks.

These two critical requirements, gas metering and compensation for failing transactions, render standard ZK circuits impractical for Aztec's public execution. A VM is the clear alternative.

Furthermore, as part of a ZK-rollup system, every transaction's execution must be provable, including its public portion. Only once a transaction is proven and rolled up in a block are its state updates accepted by the network. Therefore, a VM for public execution must be provable and compatible with Aztec's ZK-SNARK proving system.

The Aztec Virtual Machine was designed with all of these requirements in mind.

The AVM features a custom Instruction Set Architecture (ISA). Contracts with public logic contain public bytecode, a sequence of AVM instruction with a bit-format optimized for efficient constraining in a zkVM. A contract's public bytecode can be invoked by public execution requests.

The AVM:

- Executes specified public bytecode, instruction by instruction, given some arguments.
- Meters execution by tracking gas costs per-executed-instruction.
- Tracks both “mana” and “data availability” gas.
- Supports nested contract calls and conditional error recovery.
- Manages access to public state, L1↔L2 messages, public logs, and some limited private state.
- Finalizes state updates initiated during private execution.
- Handles fee payment based on accumulated gas costs.

All public execution for a transaction is performed as a unit and eventually proven in a single AVM proof.

4.1.5 Rollup Circuits

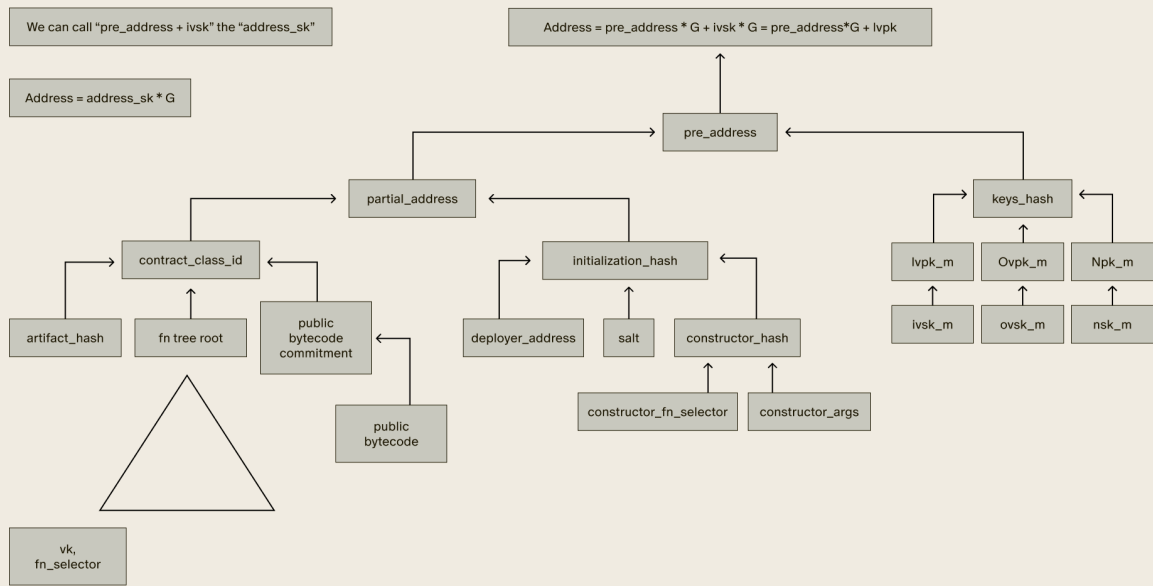
The primary role of the Rollup circuits is to aggregate many transactions into a single zero-knowledge validity proof. Batching transactions in this way means sharing the cost of verification of the proof among many users, thereby significantly reducing costs.

For the sake of efficiency, Rollup circuits also take on some additional responsibilities, including enforcing fee payments.

4.2 Contract Classes and Instances

4.2.1 Classes

Contract classes define the code of a particular contract, and any given contract class can only be deployed once. Contract classes don't own any state. In effect, they enshrine *implementations* as a first class citizen of the protocol: given multiple contract instances that rely on the same class, the class needs to be declared only once, reducing the deployment cost for all contract instances.



The contract class ID of a contract class is deterministic, and is stored in the Nullifier tree to ensure the same contract class is not deployed multiple times.

4.2.2 Instances

A contract instance is a concrete deployment of a contract class. A contract instance always references a contract class, which dictates what code it executes when called. A contract instance has state (both private and public), as well as an address that acts as its identifier. A hash of the function that ran on initialization, and optionally any public keys associated with the instance is also attached to a contract instance.

4.3 Fees

Fees play a critical role in the functioning of the network. First, because Sequencers and other network stakeholders must be compensated for their work, and second because the network must provide a mechanism to resolve competitions for the right to be included in a particular block.

Aztec's fee mechanism shares some similarities with Ethereum's. It aims to provide users with predictable fees, while ensuring excess demand gets appropriately priced. If the network is congested (demand for blockspace exceeds supply) for an extended period of time, the token total supply may deflate through the burning of a congestion fee.

Sequencers and Provers are rewarded for the services they provide in \$AZTEC.

4.3.1 Fee Payment Contracts

Users of Aztec have the ability to pay for their transactions using whichever token is most practical, but Sequencers do not have to manage the technical and exchange rate risks which would come with collecting fees in arbitrary assets.

To meet these requirements, the Aztec Network is supported by "Fee Payment Contracts" (FPCs). These contracts operate on both Ethereum L1, and Aztec Network. In effect, users task them with paying for their transaction, in exchange for a fee denominated in any of the assets the chosen FPC supports.

FPCs may choose to enable users to pay the fee associated with their transaction completely privately.

4.3.2 Node operator costs

The main cost of running a Sequencer comes from paying Ethereum gas and blob gas fees to propose blocks to the pending chain. For Provers, the primary cost comes from producing the proof (in the form of electricity/hardware), and verifying the proof on Ethereum (in the form of Ethereum gas). Both Sequencers and Provers pay a hardware cost for storing the data associated with the Aztec network.

4.3.3 Mana

To compensate node operators for their costs, user transactions are metered. The unit which represents the amount of work the transaction will incur node operators is called "mana". Each transaction has a fixed, overhead amount of mana that it will consume; each piece of state created (notes, nullifiers, logs, l2-to-l1 messages), or action taken in the AVM incurs additional mana charges.

4.3.4 Base fee per mana

User transactions are ultimately priced in \$AZTEC, even if they are using an FPC. To convert from the amount of Mana their transaction consumes into \$AZTEC, the protocol defines a base fee per mana which fluctuates with respect to costs borne by sequencers and provers, as well as supply and demand.

$$\text{base_fee_per_mana} = \text{azt_per_eth} * \text{congestion_multiplier} * (\text{sequencer_component} + \text{prover_component})$$

The `azt_per_eth` used in the calculation may be adjusted up to a fixed percentage by Sequencers when they propose a block.

The `congestion_multiplier` is derived based on the current amount of "excess_mana", i.e. `parent.excess` + `parent.spent` -

TARGET_MANA_PER_BLOCK. `parent.excess` refers to the amount of excess mana recorded in the parent block, and `parent.spent` refers to how much mana was used in the parent block. In this way, it is a running sum. If the sum of the `parent.excess` and `parent.spent` is less than the TARGET_MANA_PER_BLOCK, the excess mana for the current block is zero.

This “excess_mana” is fed into an exponential function, restricting it from moving beyond a known percentage, i.e.:

$$\text{congestion_multiplier} = \text{MINIMUM_CONGESTION_MULTIPLIER} * \exp(\text{excess_mana} / \text{CONGESTION_QUOTIENT})$$

MINIMUM_CONGESTION_MULTIPLIER will initially be 1, and the CONGESTION_QUOTIENT will be 854,700,854. Parameterized as such, the congestion multiplier is restricted from moving more than ~12.5% from block to block.

The Sequencer Component is derived as follows

$$(\text{CHARGED_GAS_TO_PROPOSE} * \text{L1BaseFee} + \text{BLOBS_PER_BLOCK} * 2^{17} * \text{L1BlobFee}) / \text{TARGET_MANA_PER_BLOCK}$$

CHARGED_GAS_TO_PROPOSE is hardcoded, and commensurate with the amount of Ethereum gas required to propose a block to the pending chain. Similarly, BLOBS_PER_BLOCK is hardcoded, and set based on the expected number of blobs needed to achieve a specific TPS.

The Prover Component is

$$\text{NUM_SUBSIDIZED_PROOFS} * \text{CHARGED_GAS_TO_VERIFY} * \text{L1BaseFee} / (\text{SLOTS_PER_EPOCH} * \text{TARGET_MANA_PER_BLOCK}) + \text{proverCostPerMana}$$

NUM_SUBSIDIZED_PROOFS is the number of proofs per epoch that the protocol wishes to incentivize. CHARGED_GAS_TO_VERIFY reflects the approximate amount of Ethereum gas required to verify a proof of validity for an Aztec epoch on Ethereum.

proverCostPerMana is a parameter tunable by Governance to compensate provers for the electricity and hardware costs of proving.

It should be noted that L1BaseFee and L1BlobFee are both values maintained by the rollup contract itself, and are updated periodically, automatically as blocks are produced. This affords more predictability to users pricing their transactions.

4.3.5 Fee distribution

When a block is proven, the portion of the transaction fees contained in the block are made available to the Sequencer and Prover, which they may subsequently claim.

The portion going to these actors is commensurate with their relative representation in the base fee calculation. However, the portion of fees derived from the congestion multiplier is “burned” by being transferred to an inaccessible address.

4.3.6 Block rewards

The canonical rollup has access to \$AZTEC held in the RewardDistributor contract, which it makes available as an additional incentive to Sequencers and Provers as blocks are proven. The split of block rewards between Sequencers and Provers is tunable, and set by Governance.

4.3.7 Multi-proofs and reward boosts

As noted above, it is possible for multiple provers to submit proofs of validity to the rollup. The provers who submit proofs for the greatest number of blocks for a particular epoch are entitled to a portion of the fees and block rewards for that epoch. Further, as individual provers consistently prove epochs, they accrue a multiplier of their proving rewards, represented as the share of total \$AZTEC to be paid out for that epoch; the multiplier decays as the prover is inactive.

4.4 Accounts

Every account in Aztec is a smart contract which defines the rules for how to authorize a transaction. Accounts can implement different schemes for transaction signing, nonce management, and fee payments. This feature is typically referred to as “Native Account Abstraction”, or “Protocol Enshrined Account Abstraction”.

In practice, this means users are not forced to use a secret key as their primary authentication method. Accounts in Aztec can be gated, for example, through passkeys, TouchID or FaceID, proofs of passport, or even a combination of all of them. Developers and users have the native ability to define authentication schemes that fit their desired UX and security preferences.

Account contracts must implement an 'entrypoint' function that receives the actions to be carried out and an authentication payload. In pseudocode:

```
publicKey: PublicKey;

def entryPoint(payload):

    let { privateCalls, publicCalls, nonce, signature } = payload;

    let payloadHash = hash(privateCalls, publicCalls, nonce);

    validateSignature(this.publicKey, signature, payloadHash);
```

```

foreach privateCall in privateCalls:

    let { to, data, value } = privateCall;

    call(to, data, value);

foreach publicCall in publicCalls:

    let { to, data, value, gasLimit } = publicCall;

    enqueueCall(to, data, value, gasLimit);

```

Since wallets are responsible for constructing transactions, there exists a tight coupling between account contract standards and wallet software (though a user can always move between wallets by changing their account contract type).

4.4.1 Nonces and replay protection

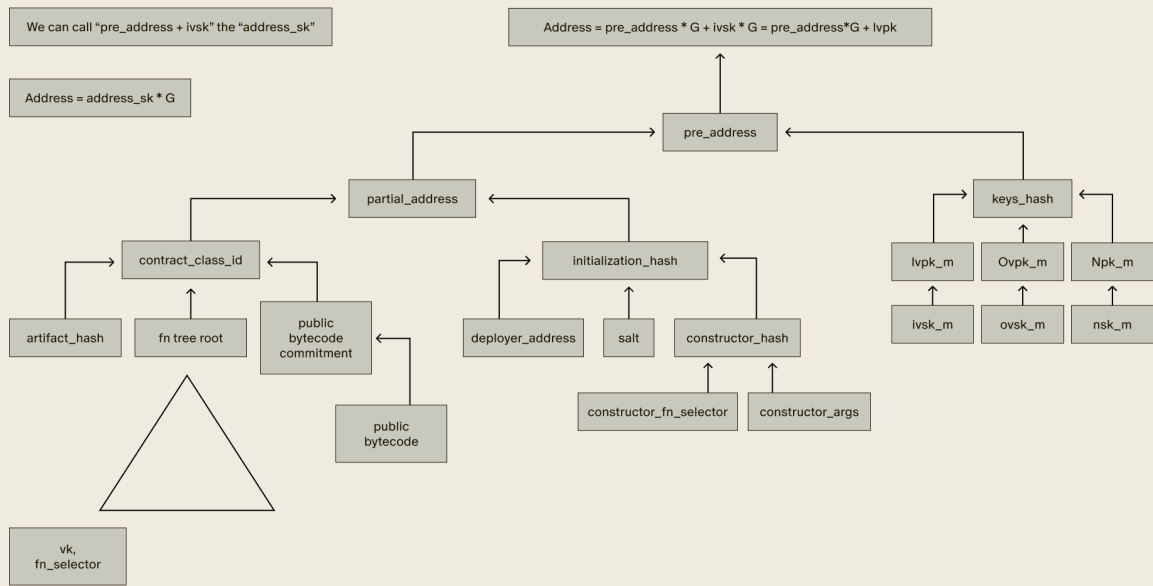
Every transaction execution should include a nonce, defined by the wallet (and enforced by the account contract). This nonce has two responsibilities: ensuring each transaction is unique and enforcing transactions have a defined order if a user submits multiple transactions in quick succession. Account contracts can enforce any ordering scheme.

As part of transaction execution, nonces associated with an account contract are added to the nullifier tree, ensuring they can only be used once. It is possible to cancel a pending transaction by submitting another transaction which emits the same nonce. The original transaction would then revert, unable to prove its associated nonce is unused. The second transaction would need to get confirmed first, which can be achieved by paying higher fees.

4.4.2 Addresses

Much like Ethereum, all smart contracts on Aztec are represented by an “address” (a hexadecimal number which uniquely represents an entity on the Aztec Network).

On Aztec, it is derived through hashing information which is specific to the entity represented by the address. This information includes the bytecode of the smart contract, and the various public keys that are used in a private context for encryption and nullification.



Because of Aztec’s protocol-enshrined Account Abstraction, the above scheme is how all addresses are derived. Aztec has no support for “EOA” addresses, derived only from a public and private key pair.

Aztec addresses can be pre-computed, and are deterministic. A user can locally compute the address of a contract and share it with others without needing to interact with the network.

Similarly to Ethereum, an address can own state within a particular smart contract (according to the rules defined in that smart contract). Novel to Aztec, addresses are designed to be public keys that can be encrypted to. Users can send encrypted state to a recipient knowing only their address. No other cryptographic data is needed.

To call a function of any smart contract on Aztec, its address needs to be provided.

4.4.3 Non-user-owned accounts

Since the entrypoint interface is not enshrined, there is nothing that differentiates an account contract from an application contract in the protocol. This means that a transaction can be initiated by any contract, so long as it has an entrypoint. One practical feature enabled by this design is ‘Paymasters’, i.e. allowing a contract to pay fees on behalf of a user.

4.4.4 Account initialization

Interestingly, a private-function-only contract on Aztec does not need to be ‘deployed’, in the sense of publishing data about the contract; its functions can be executed as part of transactions without the rest of the network having seen the functions or the contract’s

address. Private-function-only account contracts can enable users to start using the chain cheaply and privately. Some account contract designs might require public functions, in which case those public functions will need to be published to the network before they can be executed.

Aztec supports counterfactual addresses, where a user can be the recipient of a transaction prior to their account contract's deployment. A user can deterministically derive their address from their encryption public key and the account contract class ID they intend to deploy, and share this address with other users that want to interact with them before they deploy the account contract.

Wallets or applications can cover the costs of an account contract deployment, through fee abstraction [section 8.1.4](#).

4.4.5 Denial of Service (DOS) attack prevention

Protocol-enshrined Account Abstraction is vulnerable to DoS attacks, since Sequencers must authenticate the transaction according to the arbitrarily defined entrypoint logic of the emitting contract before deciding whether to include it in a block, and Account Abstraction theoretically gives developers the ability to define authentication logic which can be arbitrarily expensive.

Most current implementations mitigate these issues by restricting what's doable in the validation phase. EIP4337 defines a set of prohibited opcodes and limits storage access (see [Simulation](#) in the EIP), and requires a [reputation system](#) for global entities. Starknet simply [does not allow calling external contracts](#).

On Aztec, all Account Contract entry points must be private functions. They are executed on a user's machine, and as a result present no DoS risk to the network. Users are free to call other private functions, access storage, or do as much computing as needed as part of their authorization flow, giving developers maximum flexibility.

Beyond authorization, fee payment can also create DoS risk. Some fee payment design patterns require execution of a non-revertible public function, executed by Sequencers. Crucially, such public functions might contain logic that ensures the FPC provider gets properly reimbursed (through some abstract token) for taking on the risk of offering to pay a fee to the Sequencer on the user's behalf. To protect FPC providers against the risk of such public functions reverting – and hence not compensating the FPC provider – the protocol enables a non-revertible phase of execution (aka “Setup phase”) as well as a more conventional revertible phase. If a Sequencer includes a transaction which executes a non-revertible public function (a public setup function), and if the protocol demands that such a function cannot revert, the Sequencer is at risk of a DoS attack if the requested function takes an unexpectedly long time to execute. The Sequencer

cannot rely on the function running out of gas, because that would cause a reversion which would render the transaction invalid and not includable in a block.

This risk is only relevant if a user chooses to pay through a Fee Payment Contract ([section 4.3.1](#)), which involves calls to arbitrary third-party contracts. It is expected that Sequencers whitelist Fee Payment Contracts before they can be used. Explicitly, the expectation is that potential DoS issues resulting from Fee Payment Contracts are resolved via out-of-protocol Sequencer coordination.

4.5 Noir

Noir is an open-source, Rust-like language for safe and seamless construction of privacy-preserving Zero-Knowledge programs, requiring no previous knowledge on the underlying mathematics or cryptography.

Noir is designed to support use across a wide range of environments, including cloud-based backends and in user's smartphones, subject to ongoing implementation and optimization. From authorization systems that keep a password in the user's device, to complex on-chain verification of recursive proofs, Noir is designed to abstract away the complexity of cryptography from developers, without any significant overhead.

4.6 Aztec.nr

Aztec.nr is a smart contract framework built on top of Noir.

Aztec.nr allows developers to write private smart contracts in Noir and extend their functionality with templated functions that simplify state management.

5 Cryptography

Since 2018, our goal of bringing real world use to blockchains has forced us to repeatedly create novel cryptographic protocols. While our preference is always to use or extend tools that have benefited from widespread use and testing, our requirements for performance and flexibility often sit beyond the capabilities of the state of the art.

For the Aztec Network, we had two main constraints:

1. Users should be able to use the network on their phones or within a web browser
2. Developers must be able to call functions from within functions in smart contracts

The first constraint motivates cryptography designed for low memory use during proof computation, while the second motivates efficient recursion. Thankfully, solving for the

latter indirectly helps us solve for the former: efficient recursion allows us to decompose programs into smaller statements, each requiring less memory to prove.

To achieve these goals, we designed and built CHONK - **C**lient-side **H**ighly **O**ptimised **P**LonK. As the name suggests, its starting point is the PLONK proving system we first introduced in August of 2019, and which became a foundational piece of many proof systems across the blockchain industry. Like PLONK, CHONK is based on elliptic curves and pairings, and circuit constraints are expressed via selector polynomials and copy constraints.

CHONK deviates from PLONK in 4 main ways:

First, we use techniques introduced by Liam Eagen, Ariel Gabizon, Marek Seiffrank, Patrick Towa, and Zachary J. Williamson in their Stackproofs paper⁹ to make it possible to efficiently prove statements about a *sequence* of circuits. This is a key enabler of Kernel Circuits ([section 4.1.3](#)).

Second, in pursuit of improved Prover time and memory use, we replace univariate quotienting with sumcheck. This allows us to eliminate NNTs (Number Theoretic Transforms), at the expense of increasing the size of proofs. This technique was the main theme of the HyperPLONK paper¹⁰ by Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang.

Third, at time of drafting, we make use of the ProtoGalaxy¹¹ folding scheme to achieve cheaper recursion. ProtoGalaxy is particularly well suited to Aztec, given our need to support folding of non-uniform PLONK circuits.

Finally, we enhance ProtoGalaxy with a scheme we call Goblin PLONK^{12,13} which allows us to defer expensive non-native elliptic curve scalar multiplications normally performed by the folding verifier, until the final proving stage. This batching makes it possible to transition to another elliptic curve we call Grumpkin, where these operations are significantly more efficient. This approach was first introduced in this paper¹⁴ by Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza.

Aztec’s elliptic curve cryptography is instantiated over a curve from the Barreto-Naehrig family of pairing-friendly elliptic curves, known colloquially as BN254 and with the following parameters:

The first pairing source group is an elliptic curve EG1 over F_q where $q = 21888242871839275222246405745257275088696311157297823662689037894645226208$

⁹ <https://eprint.iacr.org/2024/1281>

¹⁰ <https://eprint.iacr.org/2022/1355>

¹¹ <https://eprint.iacr.org/2023/1106>

¹² <https://hackmd.io/@aztec-network/BkGNaHUJn/%2FdUsu57SOTBiQ4tS9KJmKMQ>

¹³ <https://eprint.iacr.org/2024/1651>

¹⁴ <https://eprint.iacr.org/2014/595>

¹⁴ <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md>

583, with equation $y^2 = x^3 + 3$. The group of rational points, G_1 , forms a prime-order group of order r , where $r =$

21888242871839275222246405745257275088548364400416034343698204186575808495617. The curve has embedding degree 12 with respect to r .

The second pairing source group is the subgroup G_2 of order r in the sextic twist EG_2 of EG_1 over F_{q^2} . F_{q^2} is a degree-2 field extension of F_q , via $F_{q^2} = F_q[X]/(X^2 + 1)$, where $(X^2 + 1)$ is the ideal generated by $f(X) = X^2 + 1$, whose roots are $\pm i$. The equation of EG_2 is $y^2 = x^3 + 3/(i+9)$.

There exists a bilinear map taking $G_1 \times G_2 \rightarrow G_T$ where G_T is a field extension of F_q of degree 12.¹⁴

6 Cross-chain messaging

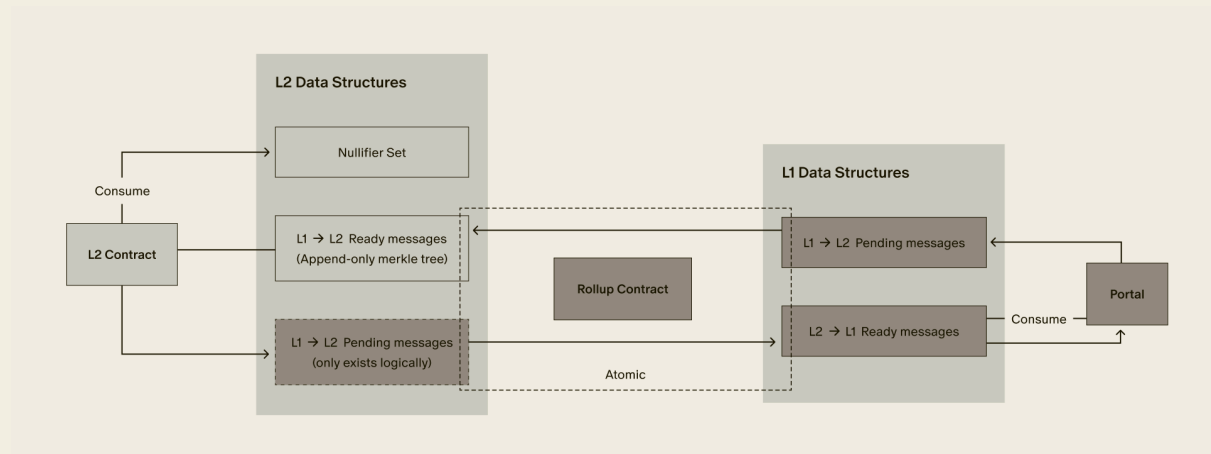
Unlike other L2 networks, Aztec does not provide a canonical bridge which has both the responsibility to facilitate cross-chain message passing *and* of custodying assets. Instead, Aztec supports “Portals”. Each Portal is free to set its own rules, and can choose to support any subset of assets users need on the Aztec Network.

The role of the Aztec rollup contract is simply to facilitate messaging between contracts deployed on the Aztec L2 and any Ethereum L1 contract, including Portals. These messages can have arbitrary payloads.

6.1 Portals

Portals are contracts which facilitate communications between Ethereum L1 and the Aztec Network. They will usually be pairs of contracts, one on each chain, which interact in prescribed ways through the Rollup contract’s Inbox/Outbox. Portals will be deployed and operated by independent providers.

They can serve as state mutexes between the Ethereum L1 and the Aztec L2. It is possible to “lock” assets within a Portal contract and have the Portal mint an equivalent proxy asset on Aztec. Similarly, if the proxy asset is burned on the Aztec L2, the Portal contract can unlock an equivalent amount on the Ethereum L1.



Portals can choose to play a more or less active part in the governance of the Aztec Network, as detailed in [section 8.2](#).

The most common envisioned use for Portals is as a way to bring assets issued on Ethereum L1 to Aztec. When used in this way, Portals behave like bridges.

Since Ethereum L1 has no native privacy, any inflows from and outflows to Ethereum will necessarily be public.

7 The Aztec Token

The Aztec Network needs to operate in a maximally neutral fashion, given the large number of distinct user groups it aims to support. The best way to ensure this neutrality is for block production to be the responsibility of a maximally distributed set of network participants (henceforth, Block Producers), and to make membership to that set permissionless. The competitive dynamics this creates give users strong guarantees that any valid transaction they submit will be included.

This set of Block Producers should be compensated for their work, and bad or lazy behavior should be explicitly penalised. Hence, users should be charged for their use of the network, and Block Producers should be required to stake assets that can be burned if a certain percentage of the Sequencer set elects to.

Additionally, as detailed in [section 8.2](#), users, app developers, and Block Producers should have the ability to steer the protocol's ongoing evolution. Ideally, the desires of those actors with a larger declared interest in future and ongoing use of the network should have more weight.

Finally, the network should be able to automatically incentivise use in periods of low user demand. Failure to maintain a predictable level of demand might make block production artificially high risk, and disincentivise the necessary infrastructure investment. This is especially important in the early days of the network.

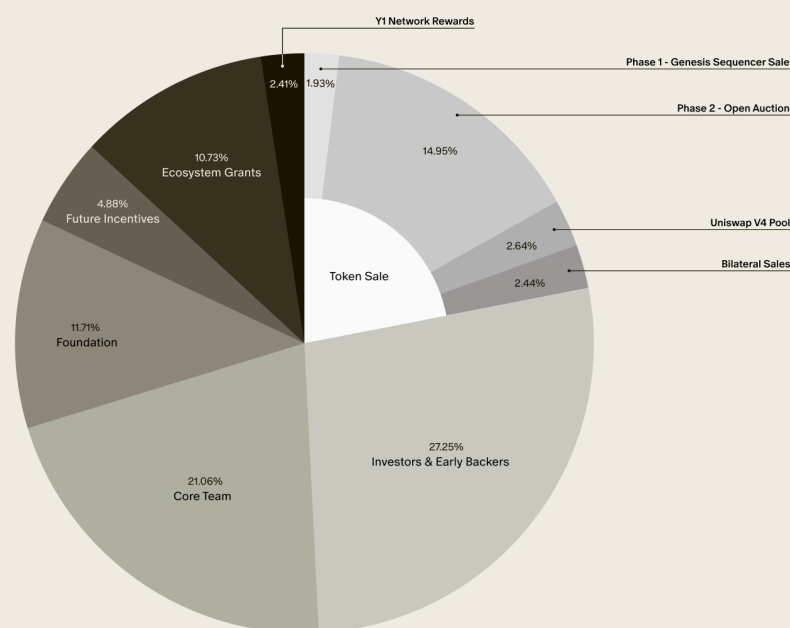
As a result, an Aztec utility token (\$AZTEC) will be created as an ERC20 token on Ethereum, and enshrined in the Aztec protocol. Creating a new utility token allows the network to incentivise use (in practice, by gifting tokens to block producers on top of the fees they collect from users), and gives users and network participants a means to meaningfully and programmatically influence the direction of the protocol.

\$AZTEC will have a total supply at launch of 10,350,000,000 tokens.

All functionality related to staking and block reward payments happens on Ethereum L1, which makes Ethereum the natural home of the Aztec token. It has no built-in privacy. This is essential in order to ensure transparency and observability of the status of the network. Users, on the other hand, need to pay for transactions happening away from L1, on Aztec's network. There must, as a result, be a bridge for the Aztec token from Ethereum L1 to Aztec.

Since the Aztec token is exclusively used to purchase blockspace, we elected to make it non-transferable once bridged to Aztec.

7.1 Distribution



7.1.1 Token Sale (21.96%)

An open community sale, to happen shortly prior to Ignition ([section 9.2](#)). This sale will optimise for 3 goals:

1. The distribution of stake to a wide set of Sequencers

2. A wide and fair distribution of tokens to future and current users of Aztec
3. Giving the Foundation assets it can use to steward and grow the adoption of the Aztec Network

The Token Sale will take the form of an auction in two phases. The first will happen at a fixed price, and will be aimed at onboarding early network Sequencers. The second phase will be as open as possible, and conducted to maximise the number of users successfully completing their desired purchase. Some portion of this 21.96% may be sold in bilateral sales to strategic investors.

7.1.2 Ecosystem Grants (10.73%)

Intended to be distributed as long term grants to non-insider early contributors, community members, and builders of core infrastructure or apps for the Aztec network. All grants in this category will be publicly documented by the Foundation.

7.1.3 Future Incentives (4.88%)

A pool of tokens fully controlled by Sequencers, through the network's governance process. It is intended for these tokens to be distributed in the first 24 months after Ignition ([section 9.2](#)). Sequencers may choose to distribute these tokens as they see fit.

7.1.4 Foundation (11.71%)

This endowment is intended for ongoing grants to future employees (components of their compensation), and on occasion as a means to fund the operations of the Foundation through conversion into fiat.

7.1.5 Insiders (48.31%)

Insiders include any current or ex-employee and any investor of Spilsbury Holdings Ltd (t/a Aztec Labs).

All Insider tokens are locked for a minimum of 12 months. Insiders tokens are non-stakeable for a minimum of 12 months.

7.1.5.1 Team (21.06%)

In recognition of the work and effort put in over the past 8 years.

7.1.5.2 Investors (27.25%)

Investors made the research and development required by Aztec possible.

7.1.6 Year 1 Network Rewards (2.41%)

These tokens have been pre-minted for first year network inflation and distribution to the Reward Distributor Contract.

7.2 Tokenomics

The Aztec Network must balance two objectives:

1. Transaction cost should be low enough for users to feel comfortable using the network
2. Network stakeholders should be compensated appropriately for the work of sequencing and proving user transactions

Meeting both of these objectives is especially difficult in periods of very high or very low demand. Accordingly, the protocol will automatically transition between inflationary and deflationary tokenomics depending on usage. In periods of low or very low demand, the protocol can elect to create additional rewards for network stakeholders by creating inflation. Conversely, as explained in [section 4.3](#), in periods of high or very high demand, the protocol may switch to deflating the token's total supply.

8 Decentralization

8.1 Network Roles

At launch, and until third-party developers deploy smart contracts, the network has no inherent or built-in functionality beyond block building and governance. This scaffolding is enabled by the actors listed below.

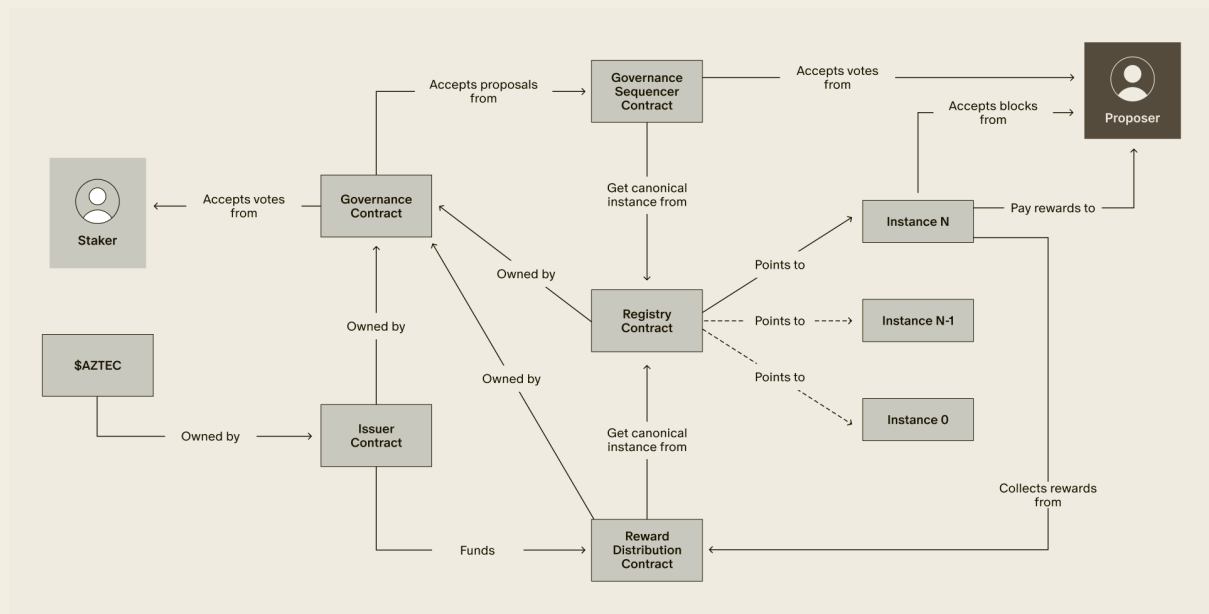
8.1.1 L1 Contracts

The Aztec Network is composed of multiple Ethereum L1 smart contracts. Together, they must provide the following functionality:

1. There should be a record of the latest state of the Aztec Network on Ethereum L1
2. The latest state of the Aztec Network should only be modified by valid user transactions; any invalid transactions should be rejected
3. The entities who do the work of collecting user transactions, proving their validity, and posting them to Ethereum L1 should be compensated for their work
4. Stakeholders in the network must be able to collectively influence the networks functionality and core parameters

The Ethereum L1 contracts which make up Aztec can be split into 3 functional categories:

1. **Rollup contracts**, responsible for setting and enforcing the rules which govern what counts as a valid L2 block, and for the cross chain messaging logic which facilitates L1<>L2 communication
2. **Governance contracts**, responsible for setting the rules through which the network can be upgraded
3. **Token and Reward contracts**, responsible for all economic aspects of the network



8.1.1.1 Rollup contracts

At any given moment, there will be at least one Aztec rollup contract deployed on Ethereum mainnet. The rollup contract is responsible for maintaining a succinct form of the Aztec Network's current state. Every block, Sequencers submit a payload of data which includes:

1. The proposed new state root of the Aztec Network.
2. All resultant state changes needed to reconstruct the full state of the rollup and generate the succinct state.
3. A signature proving the current Sequencer is eligible to produce the block.
4. A set of signatures from the network's Sequencer committee attesting to the fact the block can be proven.

Blocks are grouped into Epochs by the rollup contract. Once an Epoch is complete it must be proven within a time out for the chain to finalize. Proving occurs by submitting a zero-knowledge validity proof attesting to the fact that the proposed state transition abides by the rules of the network.

Importantly, no useful information about a user's actions can be extracted from data broadcast to execute this state transition, as long as said actions do not have explicit public side-effects. Both encrypted notes and nullifiers can be modelled as being drawn from a uniformly random distribution.

After successfully processing a valid Epoch, the rollup contract will signal to the Reward contracts that Aztec tokens can be claimed. In certain situations where a Sequencer behaves maliciously or does not follow protocol, the rollup contract may slash that Sequencer's stake, as defined in [section 8.6](#). The rollup contract is also responsible for managing the process through which Sequencers add or remove themselves from the set of potential block producers, as outlined in [section 8.4](#).

While there can be multiple Aztec rollup contracts deployed to Ethereum mainnet, there is only ever one canonical Aztec rollup contract at any given point in time. Choosing which instance gets labelled canonical is facilitated by the network's governance contracts, and (at launch) follows the process outlined in [section 8.2](#).

The canonical rollup contract takes on two ancillary roles:

- it provides an inbox/outbox which can be used by third-party contracts to send messages between Ethereum L1 and the Aztec Network
- it facilitates Sequencer voting for protocol hard forks or upgrades

8.1.1.2 Governance contracts

The Aztec governance system comprises three contracts: the Registry, Governance Contract (GC), and Governance Proposer Contract (GPC).

The Registry is responsible for keeping track of all rollup contracts which have at one point been labelled canonical, including the current canonical rollup contract. The GPC is the contract through which Sequencers can signal a desire to either update some protocol parameters, or to change which rollup contract instance is declared canonical. The GC is the contract where a proposal by Sequencers is ratified by token holders.

The function and purpose of each of these contracts is detailed in [section 8.2](#).

8.1.1.3 Token and Reward contracts

The Aztec token contract is a standard [ERC20](#) token deployed on Ethereum L1. It is fully transparent, has no special functionality beyond the implementation of the ERC20 specification, and is immutable.

This ERC20 contract is owned by the Issuer Contract, a contract which is responsible for minting new Aztec tokens, and distributing these tokens to the Reward Distribution Contract, the contract responsible for managing block production rewards. The Issuer Contract is the only contract with the authority to mint new Aztec tokens. Aztec

Governance sets an annual maximum inflation rate which the Issuer contract cannot exceed. In addition, the Issuer contract has a backstop maximum annual mint of 25 billion tokens. This limit is set to prevent infinite minting in the event of a compromise.

8.1.2 Sequencers

Aztec Sequencers are full nodes responsible for the production of blocks within the Aztec Network. They are responsible for coordinating state transitions and proof production, helping the rollup progress its state forward. Importantly, they have no visibility into the contents, purpose, or origin of any transactions they are sequencing, unless that transaction explicitly has public side-effects.

Sequencers are required to stake Aztec tokens ([section 7](#)) in order to participate in block production. Each block proposed by a Sequencer which is successfully included in a proven Epoch earns them Aztec tokens. The amount of tokens earned by a Sequencer will be determined both by the rate of block rewards (aimed in part at incentivising provision of sequencing and proving services in times of low network activity), and the fees paid by users to the network for the inclusion of their transaction.

Sequencers also play an important role in the ongoing governance of the network, primarily by instigating the process through which the canonical rollup contract is chosen. This process is described in more detail in [section 8.2](#).

As part of block production, Sequencers can take on two roles: attester or proposer.

Each Epoch (defined in [section 8.5](#)), a committee of proposers is sampled from the set of Sequencers using a pseudo random algorithm. The seed for this algorithm is sourced from the L1 RANDAO protocol. The number of proposers per epoch is a parameter set in the Rollup contract (and can, as a result, change through actions of governance).

Each proposer will be responsible for selecting transactions to include in a particular block.

Members of an epoch's Proposer committee who are not the current slot's Proposers are Attesters. Their responsibility is to assess the proposed block, sign the block's header if they believe it is valid, and distribute the contents of that block to the rest of the network. $\frac{2}{3}$ + 1 Attesters must sign a block's header for it to be considered valid.

8.1.3 Provers and Prover Marketplaces

An Aztec Prover is any entity dedicated to producing Aztec-specific zero knowledge (zk) proofs. Notably, they do not have to be full nodes, as long as they have reliable access to a full node, or the latest Aztec state. Provers can be a vertically integrated Sequencer, or a participant in a proving marketplace such as nil, gevulot, or kalypso, as long as they

choose to support the latest version of Aztec's full nodes and proving system. Sequencers and Provers coordinate off-protocol.

Provers, like Sequencers, have no visibility into the contents, purpose, or origin of any transactions included in the proof they are computing, outside of any explicitly public side effect.

8.2 Governance

The Aztec protocol is intended to be decentralized, immutable, and permissionless from launch. Nevertheless, from time to time, network stakeholders may find it valuable or necessary to update or add functionality to the network. In such situations, Aztec governance allows Sequencers to instigate a hard-fork of the network.

The set of contracts which facilitate these hard-forks include a Registry, and a two-part governance system. The Registry holds a list of all Rollup contracts which have at some point been considered the "canonical" Rollup contract.

Each rollup contract instance represents a specific and immutable version of the Aztec Network, providing strong guarantees to users. New Rollup contract instances can be created with or without copying state from the current canonical Rollup contract instance.

Sequencers can signal their desire for an upgrade by posting a payload including the address of the new Rollup contract to Ethereum L1, or voting for an already deployed payload. Sequencers can only signal payloads during the L2 slots for which they've been assigned block proposer duties. If a proposal receives support from m block proposers over n blocks, it is forwarded for approval by a wider set of holders of the Aztec Token. The initial configuration is expected to be $m = 750$ $n = 1000$, but these parameters are updatable by Aztec governance.

Sequencers play an outsized role in governance as we wanted to avoid common pitfalls of blockchain governance (namely: a difficulty reaching sensible quorums, and a lack of clear long-term orientation among participants). The requirement that tokens be staked in order to be eligible ensures a governance actor has made a conscious decision to contribute to the security of the Network, and is more likely to be engaged and informed. Importantly, users do not need to run a Sequencer themselves in order to stake. If they are unwilling or unable, they can delegate their tokens to an existing Sequencer while retaining full control of their voting rights. Governance may elect to remove this restriction in future.

After a proposal is successful it must pass a token vote. The token vote exists to ensure that a broader set of stakeholders such as apps, Portals, and Fee Paying Contract operators have an opportunity to impact the governance of the network. Participation in

the token vote requires locking Aztec tokens (removing the ability to transfer, stake, or unstake them) at least for the duration of a vote (though a token holder can choose to keep their tokens locked indefinitely). All votes are on proposals submitted through the Sequencer signaling mechanism described above. In practice, we foresee the token vote to be a force for stability for the network. The contract through which token holders vote has the power to update the Registry, change block rewards, and modify governance parameters, allowing the network to adapt to the needs of its participants.

Instigating a hard-fork, in other words, requires a Rollup contract instance to be deployed, for a governance-set m/n ¹⁵ Sequencers to signal their desire to transition to this new Rollup contract instance, and for the stakeholders of the network to approve their proposal through a token vote. If a proposal passes, the Registry is updated, and the new Rollup contract instance becomes the canonical version of the Aztec Network which Portals and users can then choose to honor.

To address potential delays in the standard governance process, Aztec also provides an emergency voting mechanism that can bypass the Sequencer signaling requirement. Under normal circumstances, Sequencers can only vote during their assigned L2 slots, and the network requires at least m Sequencers to signal before initiating a token vote. This dependency could create bottlenecks when urgent proposals need rapid consideration.

The emergency mechanism, called `proposeWithLock`, allows proposals to skip directly to the token voting stage. However, this expedited process comes with significant economic safeguards: the proposer must lock 258,750,000 tokens (equivalent to 2.5% of the total token supply at launch) for 10 years to trigger an emergency vote. This substantial commitment ensures the mechanism is reserved for genuinely critical situations. Notably, even under emergency conditions, Sequencers retain their authority and must still ratify the proposal through their votes.

8.2.1 Portals

Portals serve as the connection between the Aztec Network and Ethereum L1, allowing assets and data to be transferred between them. As the custodians of bridged assets, Portals have a responsibility to their users to ensure the security and integrity of these assets.

To fulfill this responsibility, Portals may elect to have their own internal governance mechanisms to independently decide which Rollup contract instance to recognize as the canonical Aztec Network. This allows Portals to reject upgrades that they deem unsafe

¹⁵ Set to minimize the risk that a malicious party controlling $\frac{1}{3}$ of the stake can single handedly trigger a proposal.

or not in the best interest of their users, even if the Aztec governance process succeeds. This provides an additional layer of protection against potentially harmful changes.

In practice, a Portal can choose to trust Aztec Network governance more, or less:

Portal variations and their properties around upgrades

	Action to accept upgrade	Action to reject upgrade*	Safety Relies on:	Note:
Aztec Governed	No Action	Individual User Forced Exit	<ul style="list-style-type: none"> Possible to guarantee forced exit before a rollout User follows news and knows when the upgrade is planned so he can initiate the forced exit. Cost of forced exit < cost of not exiting 	<p>This is essentially how "canonical" rollout bridges work. When there is one bridge controlled by governance, it is practically the same as this.</p> <p>I expect that people will use these as "trust-minimised" even if not fully will be useful for portals that themselves don't want any gov or control.</p>
Aztec Governed with Internal Governed Fallback	No Action	Internal Gov Vote	<ul style="list-style-type: none"> Internal governance being benevolent and competent a malicious internal governance could in most cases steal user funds. The internal governance have same issue as any other governance. 	<p>This is essentially how multi-sig bridges work. The signers of governance might be able to steal your funds if they desire to do so.</p> <p>Very similar to below, but expected to be less active. I think we will see "bridging" companies use this.</p>
Internal Governed	Internal Gov Vote	Internal Gov Vote	<ul style="list-style-type: none"> Internal governance being benevolent and competent a malicious internal governance could in most cases steal user funds. The internal governance have same issue as any other governance. 	<p>This is essentially how multi-sig bridges work. The signers of governance might be able to steal your funds if they desire to do so.</p> <p>Need more active governance to follow upgrades without having downtime.</p>
Ungovernable	Must Migrate	No Action	<ul style="list-style-type: none"> Someone (could be yourself) willing to sequence on the "old" version for you migration to work if after the fact. 	<p>Needs more sophisticated users that follow upgrades themselves. The user needs to take decisions himself so much have some deeper understanding of the upgrades to do so.</p> <p>Minority of a minority users expected to use it, people like Lasse.</p>

8.3 Data Availability

In order for the Aztec Network to progress, the latest state of the network must be accessible to:

- The Canonical rollout contract on Ethereum L1
- The set of Sequencers who will propose blocks in future

In practice, ensuring the state is available to the above actors is the responsibility of the Sequencer tasked with proposing the next block. The Sequencer must publish the new state root and a state diff, any current cross-chain messages, as well as any logs relevant to a block.

State transitions are only possible upon the successful verification of a validity proof. Historical state is not required to validate block production, but is required to fully reconstruct the state tree of the network. Under the assumption that a reasonable set of honest actors cache and make available historical Ethereum blobs, it is reasonable to post historical Network state to Ethereum blobs.

Data publication is the single biggest throughput bottleneck for Aztec. A single transaction will, on average, require publicly posting about 1500 bytes of data, so the bandwidth (**bytes per second**) of our chosen data availability solution is the main input to our theoretical maximum number of transactions per second. At time of writing, we

can achieve between 1 and 20 transactions per second by relying on Ethereum’s current blob target of 6 blobs per block. If this throughput proves limiting, governance may elect to use other data availability solutions like Celestia or EigenDA. Note (as mentioned in [section 3.4.1.2](#)), Aztec smart contract developers can also choose to make use of offchain private logs, which, if widely adopted, might reduce the average transaction size and improve throughput.

8.4 Staking

To join the Sequencer set, an Aztec Sequencer must stake tokens and enter an entry queue. The queue regulates how new Sequencers are admitted into the active set, allowing a fixed number of entries per epoch. At launch, this will start as low as 1 per epoch, ramping up to 8 per L2 epoch as the total set increases, which is approximately 40 minutes.

The queue also includes a bootstrap mode: no Sequencers are admitted until the queue reaches a minimum size. For the Ignition phase, this threshold will be set to 1,250 Sequencers. All parameters of the entry queue are configurable through governance.

This design, inspired by Ethereum’s entry queue, provides short-term stability and predictability to the Sequencer set. It also prevents sudden or unexpected turnover, which is particularly important for proof-of-stake consensus and governance security.

Sequencers may choose to exit the Sequencer set at any time, but are subject to an exit delay. This ensures that malicious actors remain slashable for a defined window before leaving. For Ignition, the exit period is set to 4 days.

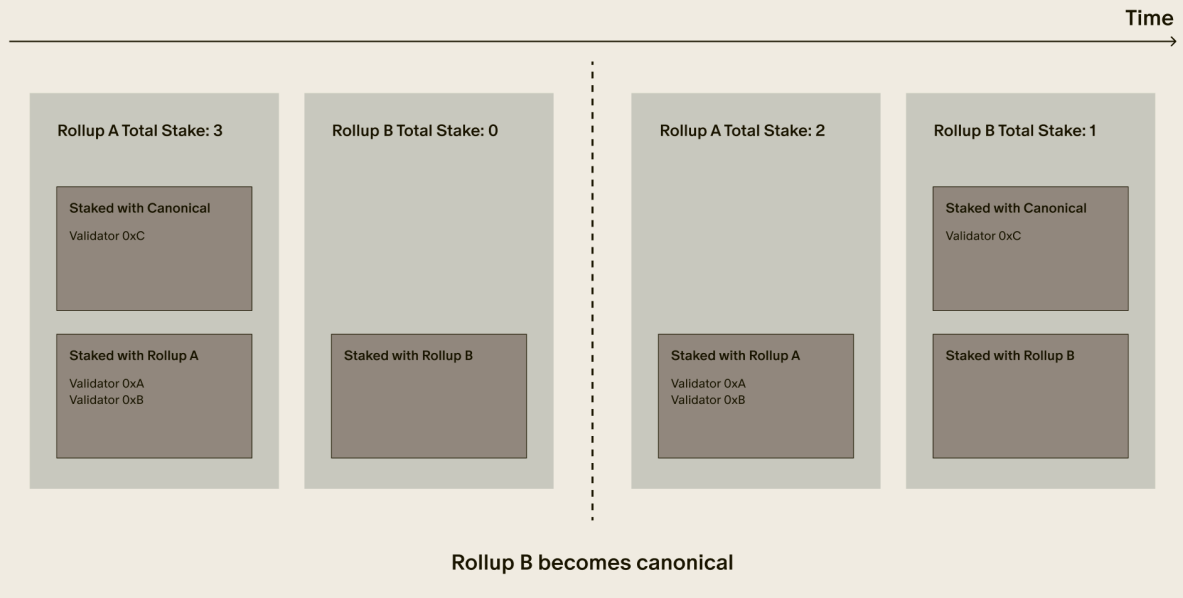
Once a staker has cleared the entry queue, they become an active Sequencer, eligible to participate in block production.

8.4.1 The Governance Staking Escrow Contract (GSE)

The stake of a Sequencer is always assigned to a particular Rollup contract. Sequencers can choose to deposit with a particular Rollup contract directly, but this creates risks if governance chooses to transition to a new canonical Rollup contract. In those situations, Sequencers who deposited their stake on the previous canonical Rollup need to manually exit and re-enter the Sequencer set, suffering both the exit delay and entry queue.

If too many Sequencers have to go through these steps, the network may suffer from low availability (not enough Sequencers on the new rollup to confirm new blocks with txs), or low confidence in the pending chain (the staking set is small and perhaps consisting of only a few entities).

We introduce the GSE (Governance Staking Escrow) contract to solve this problem. The GSE allows Sequencers to keep their stake assigned to the current canonical Rollup. If a new Rollup becomes canonical, any funds staked via the GSE are “moved” to the new canonical rollup without being subject to the exit delay.



8.5 Block Production

Time in Aztec is divided into discrete slots, which are grouped into Epochs. At launch Epochs are set to last 32 slots. A slot may or may not have at least one block, depending on whether the Sequencer for that slot proposed one.

The validity of each proposed block in a slot is eventually ensured by the production and onchain verification of a zero-knowledge validity proof covering a full Epoch. These proofs are essential to the finality of the network, but they take multiple minutes to compute. If the network only progressed once such a proof was verified onchain, users would experience excessively long block times.

To avoid this problem, Aztec uses a two phase approach to block confirmations. Blocks are first posted to a provisional ledger we call the Pending Chain, which is only secured by the stake of the Sequencer committee for an Epoch but offers users fast confirmations. Blocks reach finality when they are included in the Proven Chain, which happens once the corresponding zero-knowledge proof is verified on L1.

8.5.1 Pending Chain

The committee whose stake secures the Pending Chain is randomly selected every Epoch. At Ignition, 24 Sequencers will be selected for each committee, increased to a planned 48 at Alpha ([section 9.3](#)). This number may be changed by governance.

For each slot in an Epoch, one of these committee members is selected as Proposer. During their assigned slot, a Proposer is responsible for proposing blocks, by selecting batches of transactions from the mempool, confirming that these transactions are valid, and distributing their chosen set of validated transactions to other members in the committee as a block proposal. In order to confirm a transaction's validity, a Proposer will validate the user submitted zero-knowledge proof, and execute any related public function calls.

Outside of their assigned slots, members of the committee take on the role of Attester. An Attester's responsibility is to verify the current block's proposed transaction set and (if they believe it to be a set of valid transactions) distribute a signature over the hash of that set of transactions to other committee members. If any given block fails to collect the signatures of $\frac{2}{3}+1$ of the committee, it won't be accepted by the rollup contract as a valid block.

8.5.2 Proven Chain

At any point after a block gets included in the Pending chain, Provers can decide to create a zero-knowledge proof covering a contiguous set of Pending Chain blocks, starting from the last block that's already proven. This proof can cover just a few recent blocks or an entire epoch of up to 32 blocks; proofs of partial epochs are explicitly allowed.

Once a computed proof is verified by the Rollup contract on Ethereum L1, all blocks covered "graduate" from the Pending chain to the Proven chain, achieving finality. Once a block is included in the Proven chain, it can't be reorged, unless Ethereum reorgs.

Multiple Provers may submit overlapping proofs, but only proofs for the highest consecutive block range are eligible for block rewards. If multiple proofs finalize the same highest block height, they share rewards, with exact split determined by a quality of service metric that rewards long standing proving consistency.

8.5.3 Escape Hatch

It is useful for the Aztec network to have an emergency exit route in two situations:

1. Block production has stalled
2. Transactions are being actively censored

In both, we wish to bypass a faulty or dishonest Sequencer set (at least temporarily) and allow some other form of block production.

Unlike other L2 networks, the specifics of our design means that implementing a forced inclusion queue is very complex and risky for Aztec (see [PR#24](#) and [PR#80](#)).

Therefore, we have chosen an alternative: an Escape Hatch.

The Escape Hatch is a period where blocks can be proposed without needing to accumulate attestations from a committee. Prospective users of the Escape Hatch must deposit a bond (set to ensure costs equivalent to costs of performing an attack on the Pending Chain) in order to participate in a lottery, through which escape hatch slots will be assigned. The bond deposit and lottery resolution needs to occur a number of Epochs prior to the opening of the Escape Hatch.

If there are no bonds posted, the Escape Hatch is simply skipped.

As of publication, it is expected that 10% of the Epochs could be treated as an Escape Hatch, with an opening at least daily.

Note that governance will work as if there are no hatch, e.g., not allowing the hatch proposer to vote.

After a pre-defined period, bonds may be refunded.

8.6 Slashing and Security

Dishonest or badly performing Sequencers must be penalized. The Aztec network implements a consensus-based stake slashing mechanism we call the Tally Model.

Potential offences include:

- **Sequencer Inactivity** - A Sequencer fails to attest to at least two full epochs of block proposals. The % of missed attestations and the number of epochs that are tracked are configurable in the client.
- **Proposed Epoch is Pruned** - At the end of an epoch, Provers have a predefined number of epochs to submit proofs. By default, this submission window is set to 1 epoch which means the proof for epoch N must land before epoch N+1 finishes. If no such proof lands, then epoch N and anything built on top, is pruned (erased) and the chain restarts from the tip of the proven chain as of epoch N-1

Upon detecting a chain prune, the node will try to verify if the epoch *could* have been proven. It does so by re-executing the txs from the blocks that were pruned. If all txs are available, and the state roots from L1 match those on the node after re-execution, the node will then try to slash the entire committee on the pruned epoch.

- **Data Withholding** - Similar to the Epoch Prune offence, but the data withholding offence is triggered if upon an epoch prune, the node detects that the tx data for the epoch was not available.
- **Proposing Invalid Blocks** - A Proposer proposed a block which is assessed as invalid. An example is if the state root resulting from executing the txs in a block does not match what was posted to L1.

- **Posting Invalid Signatures** - A Proposer posts malformed signatures, or signatures over the wrong data.
- **Posting Insufficient Signatures** - A Proposer withholds some signatures or fails to post the necessary amount of signatures which are $\frac{2}{3} + 1$ of the committee.

Sequencers are not assessed on a block-by-block basis. Instead, L2 blocks are grouped in Rounds. In addition to their block-building responsibilities, Sequencers cast votes on the offences which occurred in a specified past Round.

The number of L2 blocks which make up a Round is defined on the Rollup contract, and cannot be changed. At launch, a Round is defined as 128 L2 blocks, or 4 Epochs.

Since the information required to notice or confirm an offence may take time to diffuse through the Sequencer network, the Rollup contract also defines a number of Rounds Sequencers must wait before they cast votes. At launch, this offset is set to 2 Rounds, meaning Sequencers in Round N will vote on offences which occurred in Round N-2.

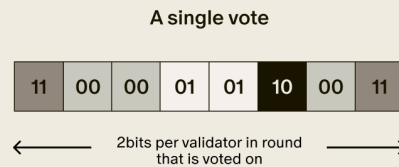
When a Sequencer comes online, they may also choose to define a “grace period”, which defines the number of Rounds they will wait prior to casting their first vote. This is set on the client directly.



In the diagram above, assuming a 128 slot Round and a 2 Round offset defined on the Rollup, a Sequencer with a 1 round grace period will begin submitting votes on Sequencers from Round 2 during Round 4.

8.6.1 What is a vote?

When a Sequencer is called upon to propose a block, they submit a byte array of votes covering the whole set of Sequencers of the previous Round being voted on.



Each slot in the array represents a vote on a particular Sequencer of the relevant Round, represented by 2 bits which define the desired slash amount, either no slash, 1%, 5%, or 15% of stake.

No other slashing amounts may be used. This means that any Sequencer may be slashed up to a maximum of 15% of their stake.

The L1 contract tallies votes and slashes Sequencers that reach quorum. The quorum, Round sizes, Round offsets, and allowable slashing amounts could be changed via governance.

8.6.2 The Slashing Vetoer

The slashing vetoer is operated by an independent group of security researchers that will provide security assessments on upgrades and administer emergency pause functionality on slashing proposals, if a pause is in the best interest of the network.

All Aztec slashing proposals must be voted on by Sequencers and ratified by the L1 contract by as described above. Once ratified, slashing proposals will be subject to a slashing execution delay period - expected to be around 3 days.

During this delay period, the slashing vetoer has the ability to prevent a slashing proposal from being executed. The vetoer also has the ability to disable slashing temporarily for a maximum period of 3 days.

This is meant as a failsafe mechanism in the event of a client software bug that causes Sequencers to be slashed unfairly.

9 Launch and Timelines

Building Aztec Connect taught us a very important lesson: decentralization, like privacy, needs to be a central design principle from the start. Any attempt to treat it as a feature to be added later leads to roadblocks and compromises.

Decentralization is required because a genuinely neutral computational environment—one owned and governed by its participants—is essential for real-world adoption of blockchains. To become the home to significant economic and social activity, we need to provide users with strong guarantees that the infrastructure they rely on won't be co-opted, seized, or destroyed by competitors or hostile actors. Users should have confidence that there is not a centralized network operator with the ability to bias the network in favor of their competitors and/or act against their interests.

We recognize that placing neutrality and decentralization in such a central place means having to deal with certain tradeoffs: coordinating distributed stakeholders introduces complexity, and protocol improvements demand more deliberate processes than in centralized systems. These tradeoffs typically manifest via lower network throughput, higher network costs and higher governance communication complexity versus centralized networks.

We gladly accept these tradeoffs.

Our launch strategy reflects this choice of priorities. Rather than going for speed through centralized control, the community will be empowered to implement decentralized governance from the outset, beginning with Aztec testnets. The path to mainnet is intended to be methodical and gradual, reflecting the need to battletest the tech stack and community commitment to building robust foundations for a truly participant-owned network.

9.1 Testnet

The Testnets are live and open networks which settle on the Ethereum Sepolia test network. The following testnets have been conducted ahead of Ignition.

9.1.1 Public testnet

Goal: prove our P0 infrastructure, and help external developers deploy end-to-end and composable applications

Date: May 2025

We aimed to have 300 Sequencers (geographically distributed, and independent), and two Proving marketplaces contributing to block building. We far exceeded that goal, reaching at peak over 5000 Sequencers and more than 20,000 full nodes.

9.1.2 Adversarial Testnets

Goal: Fix any issues in Public testnet, and stress test the network

Date: Q3 and Q4 2025

The Adversarial Testnets are a sequence of testnets which introduced fixes and new features, and are built to help us stress test the network. During Q3 and Q4 of 2025, we planned and are running a series of tests on throughput, resilience to bad Sequencer behaviour, and ability for a decentralised Sequencer set to upgrade the network. During this phase, we crossed 20,000 Sequencers. Sequencers in this set had to pass an [anti-sybil check in the form of a zkPassport proof](#), further demonstrating the decentralisation of the network.

A full walkaway test where the uptime and reliability of the network was tested after all core infrastructure run by either Aztec Labs or the Aztec Foundation was switched off was successfully conducted on [Insert Date].

9.2 Ignition

Ignition is the initial deployment of the Aztec L1 contracts to Ethereum Layer 1. Importantly, this initial deployment will not allow for any user transactions or contract deployments. Its purpose is to facilitate the bootstrapping of the initial Sequencer set, enable governance, and ensure block production works as intended on Ethereum Mainnet.

Goal: Bootstrap Aztec's Sequencer set, and start block production.

Date: Q4 2025

9.3 Alpha

Alpha is the first governance-led hard fork of the live network. It will be instigated at the discretion of the decentralized Sequencer set.

Alpha will be a version of the Aztec network with critical components audited deployed to Ethereum main-net, and can be used to mediate real funds bridged into the network. However, it should be assumed the use of the network still carries substantial security risks due to the novelty of the network technology stack and the limited tools available to a decentralized network to mitigate attacks (e.g., unable to suspend block production rights, or unwind network state, as is the case with centralized networks). Alpha is not meant to be treated as a mainnet network by users or developers. It will go through frequent hardforks, and users should expect critical bugs to be found during this period.

Goal: Upgrade the network to enable user transactions and programmability.

Date: Q1 of 2026

9.4 Mainnet (Beta)

Mainnet is reached after a period without any critical bugs reported for 90 days, AND when the network achieves an uptime of 99%. Timelines are, as a result, open ended.

Sequencers may elect to tighten governance parameters after this milestone.

We expect network participants to have significant trust in the stability and safety of the network from Mainnet (Beta) onwards.

9.5 Mainnet 1.0

1.0 will be reached once Mainnet (Beta) has run for a sufficient period of time without any reported issues.

10 The Foundation

The Aztec Foundation is a non-profit organization which conducts fundamental research in freedom-enhancing cryptography, supports and maintains the Noir programming language, and stewards the Aztec Network.

The Aztec Foundation is currently led by a 3 person board:

- Zac Williamson (Chairman), cofounder of Aztec and co-inventor of PLONK; and
- Herbert Sterchi, previously on the board of the Ethereum Foundation; and
- Arnaud Schenk, cofounder of Aztec

Zac Williamson also serves as President of the Foundation, and Arnaud Schenk is the Executive Director.

The Aztec Network is built and used by a wide variety of stakeholders, who will naturally have diverging expectations and needs. The Foundation provides neutral forums and occasional facilitation to ensure these stakeholders can effectively coordinate. The Foundation aims to increase transparency and participation in governance by summarizing and distributing relevant information through newsletters and social media.

The Foundation may also offer grants and commercial support to community contributors.

Given the expertise of the staff of the foundation, the Foundation may on occasion provide security audits, technical reviews and any other ancillary support to steward the Aztec protocol. It may also maintain a bug bounty program to uphold the network's integrity and to ensure user security.

The Foundation aims to regularly engage with companies, non-profits, and governments in order to help them understand and adopt Noir and the Aztec Network. The Foundation's goal is also to bootstrap a self-sustaining ecosystem of organizations building towards the future that Aztec and Noir enable.

The Foundation is also committed to supporting research at the forefront of cryptography, which it will release and open source. While the primary goal is to provide ancillary support to help advance the capabilities, performance, safety, and reliability of the Aztec Network, the Foundation intends to also commit significant effort towards blue sky research. In addition, Foundation intends to build and maintain reference open-source implementations of important cryptographic libraries.

Disclaimer:

The descriptions of features, phases, and timelines in this document are provided for informational purposes only. They reflect the current design intent of the Aztec Network and may be updated, modified, or delayed based on technical, or strategic considerations. No assurance is given that any particular functionality (including the Private Execution Environment or collaborative proving) will be available at launch or within any specific timeframe.