Preamble

This RFC outlines how the proposed Aztec governance enables stakeholders to make decisions related to upgrading the functionality of the network. Community members are invited to give feedback and discuss the proposed Aztec Governance system.

Section 0: Preliminaries

1. Portals

Portals are contracts which facilitate communications between Ethereum Layer 1 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.

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 or not in the best interest of their users.

In other words, portals do not have to conform to any one form of governance, including that of the rollup. For more readings on Portal governance, please see this post.

1. Deployment

Any specific deployment of the Aztec Rollup ("an instantiation") will contain several L1 contracts described in the Governance Contracts section.

An "instance" refers to a specific Rollup smart contract. If at t_0

there is one Rollup smart contract, called 0xa, and at time t_b

governance vote deploys and activates Rollup smart contract 0xb, then we have two rollup instances with 0xb being the canonical rollup.

1. Canonical Rollup

Some governance proposals result in a new Rollup smart contract being deployed to L1. From the perspective of the Aztec Governance, that new contract becomes the "true" Rollup once the governance vote is concluded.

Section 1: Summary

[

Screenshot 2024-12-09 at 1.29.07 PM

2242×1342 243 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/c/c9561ada5ac05ac1d5e1ae83766c2f59ceeaf2da.jpeg)

Sequencers put forward, or "nominate", proposals for voting by the Aztec citizens. To do this, sequencers interact with the Governance Proposer smart contract. Nominations are "signals" by sequencers that they wish to put up for vote the execution of certain code by the Governance smart contract.

If the Governance Proposer smart contract records a certain number of nominations/signals from sequencers, then the Governance Proposer smart contract initiates a voting process where holders of any Hypothetical Assets (as defined below) can participate. Holders of such Hypothetical Assets are called Aztec citizens.

All signalling and voting happen on the L1.

Who puts forward proposals for voting by governance?

Sequencers of the current canonical rollup (as indicated by the Registry smart contract) can propose changes to the Aztec community for voting. In order for a proposal to proceed through the governance process, N

sequencers must nominate the proposal in any given round. A round is defined as a sequence of contiguous M

L2 blocks.

Sequencers can only nominate a proposal during an L2 slot for which they've been assigned proposer duties. This minimizes timing games and provides a lower bound on the time required to successfully bring about a vote by governance.

This however would lead to an inability to nominate proposals for voting in the event that the sequencer rotation mechanism goes down. To mitigate this risk, a mechanism is proposed whereby any digital asset (ETH or any other relevant ERC-20 asset)(“Hypothetical Asset”) holder (“Holder”) can burn a large quantity of Hypothetical Asset to trigger a vote on a proposal, without having the sequencers nominating the proposal. Note that Hypothetical Asset holders would still need to vote on this proposal i.e. this proposed procedure is only a nomination.

To nominate a proposal, a validator of the current canonical rollup would deploy two sets of contracts:

1. The upgraded contracts they wish to upgrade to i.e. a new Rollup smart contract.
2. A smart contract containing code

which can be executed by governance to upgrade to the contracts deployed in (1).

Then when it is their turn as the proposer, sequencers call vote(address _proposal)

on the Governance Proposer contract, where _proposal

is the address of the code payload.

Note

This implies that only sequencers, who have staked the minimum amount of Hypothetical Assets, can nominate proposals.

How does a proposal get tabled for voting?

Once the upgrade code is deployed to address _proposal

, sequencers call the vote(address _proposal)

function on the Governance Proposer contract specifying the same address.

Nominations (or “signals” for proposals) occur in “rounds” of M

= 1,000

blocks. Round 0 is the interval from L2 slot 0 - L2 slot 999 and so on. The Governance Proposer smart contract keeps count of the sequencers who signalled for a proposal during any given round. A proposal must receive N

= 667

signals in any single round to move forward to the second ratification stage by governance.

Note

These values of N

and M

are for illustrative purposes only. It is expected that N

and M

will be immutable, chosen at the time of the Rollup smart contract deployment.

To move a proposal that has received N

signals in a single round of M

blocks to the voting phase, anyone can call the pushProposal(uint256_roundNumber)

on the Governance Proposer contract which will in turn call GOVERNANCE.propose(_proposal)

on the Governance contract and start the voting process.

Who can vote on proposals?

Holders have the ability to vote on proposals as long as they lock any Hypothetical Assets within the Governance contract. The act of locking the funds can be thought of as “activating” the voting power of Hypothetical Asset. Locked Hypothetical Assets used to vote on a proposal must wait a delay before being withdrawn to prevent malicious governance attacks.

Hypothetical Assets locked in the Governance contract are simply locked and not “at stake” i.e. there are no slashing conditions.

Since sequencers may be able to stake Hypothetical Assets with the the rollup instances in order to join the validator set, the rollup instance could in turn lock those Hypothetical Assets in the Governance contract and vote on behalf of the sequencers. This is expected behaviour.

Section 2: Governance Contracts

An Aztec deployment is a set of the following contracts:

Smart Contract

Immutability

Hypothetical Asset

Immutable

Issuer Contract

Immutable

Registry Contract

Immutable

Reward Distribution Contract

Mutable

Governance Proposer Contract

Mutable

Governance Contract

Immutable

Rollup Contract

Immutable

Hypothetical Asset Contract

Hypothetical Asset would live on Ethereum L1. It may have a minter

which would be the only actor capable of calling the mint

function on the Hypothetical Asset contract.

This is brought up to the community for discussion purposes only to illustrate how proposed governance mechanisms could work with a Hypothetical Asset.

Validators must stake the Hypothetical Asset with the Rollup smart contract to join that instance’s validator set, while holders must lock the Hypothetical Asset with the Governance contract to be able to vote on proposals.

Provers must deposit Hypothetical Asset in the escrow contract in order to bid for the right to create a proof for an epoch.

Issuer Contract

This contract will be the sole minter

of Hypothetical Asset. It will itself have an owner

which is the only actor capable of calling the mint

function on the Hypothetical Asset ERC20 contract.

The mint

function will limit the amount of Hypothetical Assets that could be minted in a given time period. The reasoning behind this limit is that it makes supply expansions more predictable since “infinite mints” cannot be made.

[

Issuer Contract is the only actor capable of minting Hypothetical Assets

928×490 23.7 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/e/e60ffc1b0893649e95484691dfe9760d7dccfd15.png)

contract Issuer is Ownable { ERC20 immutable public ASSET; // Hypothetical Asset uint256 immutable public RATE; uint256 public timeOfLastMint;

```
constructor(ERC20 _asset, uint256 _rate, address _owner) Ownable(_owner) {
    ASSET = _asset;
    RATE = _rate;
    timeOfLastMint = block.timestamp;
}

function mint(address _to, uint256 _amount) external onlyOwner {
    uint256 maxMint = RATE * (block.timestamp - timeOfLastMint);
    require(_amount <= maxMint, 'Insufficient mint available');
    timeOfLastMint = block.timestamp;
    ASSET.mint(_to, _amount);
}

}
```

Registry Contract

The governance of Aztec will be community driven - the Aztec community will be able to decide whether to upgrade or migrate to a new rollup instance. Portals / apps don't need to follow along with the Aztec governance and can specify the specific instance (i.e. "version") of the rollup that they view as canonical.

Therefore it will be necessary to keep track onchain of what versions of the rollup have existed as well as what version the Aztec Governance views as the current canonical rollup. Only the current canonical rollup from the perspective of the Aztec Governance will be eligible to claim any further Hypothetical Asset rewards.

In practice, the Registry is an append-only array of rollup instances. The array can only be appended by the Registry's owner - the Governance contract.

[

Registry Contract tracks all Aztec Rollup instances that were canonical at some point

946×628 80.9 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/e/e075e8d93868353d928a5ae39e078200f1ad9949.png)

contract Registry is IRegistry, Ownable { struct Instance { address rollup; uint96 blockNumber; }

```
Instance[] public instances;

constructor(address _gov) Ownable(_gov) {
    instances.push(
        Instance({address: address(0), blockNumber: block.number})
    );
}

function addInstance(address _rollup) external onlyOwner {
    instances.push(
        Instance({address: _rollup, blockNumber: block.number})
    );
}

function getRollup() external view returns(address) {
    return instances[instances.length - 1].rollup;
}

function getInstancesLength() external view returns(uint256) {
    return instances.length;
}

}
```

Reward Distribution Contract

This contract distributes ERC20 rewards only to the instance the Registry contract says is canonical. This is separated from

the Registry and the Issuer so that the distribution logic can be changed without replacing the Registry.

In practice, the flow is expected to be such that infrequently, the Aztec Governance votes that the Issuer smart contract mints a quantity of Hypothetical Asset and sends them to the Reward Distribution contract.

When a new epoch has been proven, the rollup smart contract will call the claim(_to)

on the Distribution contract which checks that the calling Rollup is the current canonical Rollup before releasing a Hypothetical Asset to the rollup.

The Rollup smart contract implements custom logic for how to split BLOCK_REWARDS

amongst the sequencers/committee/provers who provide real work in the form of electricity and hardware intensive computational resources to the network.

[

The Rollup contract calls the Reward Distributor contract to collect any BLOCK_REWARDS for the epoch.

984×662 70.3 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/e/e33dd1478ef5e27af1c39c51c5cd34368d6b3c7b.png)

contract RewardDistribution is Ownable { uint256 public constant BLOCK_REWARD = xxx;

```
IERC20 public immutable ASSET;
IRegistry public registry;

// constructor etc

function claim(address _to) external returns(uint256) {
    address canonical = registry.getRollup();
    require(msg.sender == canonical);
    ASSET.safeTransfer(_to, BLOCK_REWARD);
    return BLOCK_REWARD
}

function updateRegistry(IRegistry _registry) external onlyOwner {}
// ...

}
```

If the Rollup smart contract is no longer the canonical instance, then any calls to claim()

would revert. Therefore it is important that Rollup smart contract implementations not fail in this case to make sure sequencers can still service the instance after it is no longer canonical.

The protocol inflation rate is capped in the Issuer smart contract as the constant RATE

. It is not possible to change this cap once the Issuer smart contract has been deployed.

The Aztec Governance will choose how often to call mint

on the Issuer smart contract which will send any Hypothetical Assets to the Reward Distribution smart contract. The Reward Distribution smart contract defines a BLOCK_REWARD

immutable value. Every epoch, the Rollup contract can call the Distribution smart contract to claim BLOCK_REWARD

of Hypothetical Assets from the Distribution contract.

The BLOCK_REWARD

is the actual inflation rate and cannot exceed RATE

. Both RATE

and BLOCK_REWARD

will be set upon deployment of the Aztec Rollup by the Aztec Governance. Aztec Governance may vote to deploy a new RewardDistribution smart contract that defines a different BLOCK_REWARD

as long as it is less than RATE

Governance Proposer Contract

This is the only smart contract that is able to submit proposals to the Governance contract.

The Governance Proposer Contract will accept proposals only from sequencers of the current canonical instance, as indicated by the Registry.

contract GovernanceProposer is IGovernanceProposer {

// ... imports

IGovernance public immutable GOVERNANCE;
IRegistry public immutable REGISTRY;
uint256 public immutable N;
uint256 public immutable M;

constructor(IGovernance _governance, IRegistry _registry, uint256 _n, uint256 _m) {
  // ...

  require(N > M / 2);
  require(N <= M);
}

function vote(address _proposal) external override(IGovernanceProposer) returns (bool) {

// this code check has been removed to allow utilizing the same logic to vote on slashing proposals. // require(_proposal.code.length > 0); // ... Rollup instance = Rollup(REGISTRY.getRollup()); address proposer = instance.getCurrentProposer(); require(msg.sender == proposer); } // ... }

To vote to table a proposal, anyone may deploy to L1 the contracts being proposed to upgrade to as well as the upgrade logic i.e. the code

that Governance contract will execute. The current L2 slot's sequencer then call GovernanceProposer.vote(_code)

. This is referred to as a "signal".

The Governance Proposer contract will then count signals specifying that same _code

. For a proposal to be nominated for voting, it must garner at least N

signals in a single round, where a round is defined as a M

consecutive L2 slots.

Note

A sequencer's ability to signal is not affected by the rollup's availability since voting happens on the L1.

[

If enough signals are collected in one round, the Governance Proposer smart contract initiates a vote.

1538×944 66 KB

](https://europe1.discourse-cdn.com/flex013/uploads/aztec/original/2X/d/d4e0d9ecd7e7a97790bd29124c3466e45619ea56.jpeg)

If the signalling quorum has been reached, anyone can call a pushProposal(uint256 _roundNumber)

on the Governance Proposer smart contract to send the proposal to the Governance smart contract for voting. Only one successful proposal per round can be sent to the Governance contract for voting.

Governance Contract

This contract is the "assembly of Aztec citizens" that is the final arbiter of whether to enact the proposals from the Governance Proposer Contract or not.

This contract decides what is the canonical instance which gets BLOCK_REWARDS

.

The Governance Proposer contract tables proposals for voting, Aztec network participants who lock their Hypothetical Assets with the Governance contract may vote once for each Hypothetical Asset locked. They can vote either Yea or Nea.

Once a proposal garners the minimum number of votes, and the Yea votes exceed Nea by at least the quorum%

, the proposal can be executed by the Governance contract.

contract Governance is IGovernance { // ... imports

```
IERC20 public immutable ASSET;
address public proposalsContract;

// ...
constructor(IERC20 _asset, address _proposalsContract, uint256 _votingDelay, uint256 _votingDuration,
            uint256 _gracePeriod, uint256 _quorum, uint256 _voteDifferential, uint256 _minimumVotes) {
```

// ... configuration = DataStructures.Configuration({ votingDelay: Timestamp.wrap(_votingDelay), // Min time between proposal creation and when voting starts votingDuration: Timestamp.wrap(_votingDuration), // Max duration of voting period executionDelay: Timestamp.wrap(_executionDelay), // Min time between voting passing and proposal execution gracePeriod: Timestamp.wrap(_gracePeriod), // max time between proposal creation and proposal execution. quorum: _quorum, // % of deposited Assets that must participate in a vote (could be Yes or No) votingDifferential: _voteDifferential, // Yea must outweigh Nay by this % to pass vote
minimumVotes: _minimumVotes, // users with this much cummulative deposited Assets must participate in the vote }) }

```
// ...
function deposit(address _onBehalfOf, uint256 _amount) external override(IGovernance) {
    // deposits are allowed on behalf of other addresses. This is useful to enable sequencers to vote.
    users[_onBehalfOf].add(_amount);
    // ...
}

function initiateWithdraw(address _to, uint256 _amount) external override(IGovernance) returns(uint256) {
    // ...
    // No one can withdraw on behalf of someone else
    users[msg.sender].sub(amount);
    // ...
}

function propose(address _payload) external override(IGovernance) returns(bool) {
    require(msg.sender == proposalsContract);
    // ...
}

function vote(_proposalId, uint256 _amount, bool _support) external override(IGovernance) returns(bool) {}

function execute(uint256 _proposalId) exte rnal override(IGovernance) returns(bool) {
    // execute proposal via `call()`
}

}
```

Section 3: PoS

A PoS system is proposed to protect against sybil attacks on the validator set. The major features of the proposed PoS system are:

Separation of attester and proposer keys

Each node operator who wishes to join the validator set must specify a pair of identifiers - a proposer

address for proposing blocks and an attester

key for attesting on block proposals. Duplicate proposer

addresses are allowed whereas duplicate attesters

keys are not. Together the set of the proposer

address and the attester

key make up what is called an Operator

.

It is possible to specify a smart contract address as the proposer

as this will allow for advanced setups and will make building LSTs easier.

Withdrawer keys

Every Operator is enriched with a withdrawer

address. The withdrawer

is able to initialize a removal of the Operator from the set - without the help of the proposer or attester. The withdrawer can initiate a withdrawal of the stake to some recipient

address.

Validators can also vote for Governance proposals

:

Sequencers stake Hypothetical Assets with the Rollup smart contract. However, it is expected that the Rollup smart contract may in turn lock Hypothetical Assets with the Governance contract and vote on behalf of staked sequencers.

Entering the validator set

Any address with the required Hypothetical Assets could stake with the Rollup smart contract, specifying 3 identifiers in the process.

function deposit(address _attester, address _proposer, address _withdrawer, uint256 _amount) public { require(_amount >= MINIMUM_STAKE) HYPOTHETICAL_ASSET.transferFrom(msg.sender, address(this), _amount);

The two requirements are:

1. attester

must be unique and not already registered in the set.

1. msg.sender

has the necessary Hypothetical Assets.

Note

The attester

value may be implemented as a BLS key instead of an address.

Exiting the validator set

Exiting the validator set happens in two steps. A preliminary withdrawal must be initiated by the withdrawer

address.

function initiateWithdraw(address _attester, address _recipient) public { // The validator is identified by the unique attester key. ValidatorInfo storage validator = stakingStore.info[_attester]; // Only the withdrawer address may initiate a withdrawal. require(msg.sender == validator.withdrawer)

A few things are checked here:

1. Each Operator has a status of either VALIDATING

, LIVING

or EXITING

. An initial withdrawal triggers a status change from VALIDATING

or LIVING

to EXITING

.

1. Only the withdrawer

address can call the initiateWithdraw

function.

1. The withdrawer

address specifies a recipient

address to which any Hypothetical Assets will be sent.

1. No amount is specified as only full exits are supported.

Once the checks pass, the status of the attester

Operator becomes EXITING

and they are removed from the validator set. Any Hypothetical Assets may not be withdrawn until after EXIT_DELAY

time has passed.

After the exit delay has passed, any address can call a finalizeWithdraw()

function to trigger a transfer of the entire balance of Hypothetical Assets to the recipient

address specified in the initiateWithdraw()

call.

The EXIT_DELAY

serves as a buffer period to allow for slashing. Once EXIT_DELAY

time has passed, the validator cannot be slashed by the Aztec Governance. In other words, if it is possible to successfully call finalizeWithdraw()

then the balance of a validator's Hypothetical Assets cannot be slashed.

Section 4: Slashing

Background

We need to make sure that the chain is always (eventually) finalizing new blocks. The conditions required for the chain to finalize new blocks:

- Sequencers are including txs in blocks and making block proposals.
- More than 2/3 of the committee are making attestations.
- Validators are gossiping client side proofs and TxObject data to committee members and to the provers via the p2p layer for example.
- Provers are producing proofs.

In the event that a significant portion of the validator set goes offline (i.e. geopolitical emergency) and proposers are unable to get enough attestations on block proposals, the Aztec Rollup will be unable to finalize new blocks. This will require the community to engage to fix the issue and make sure new blocks are being finalized.

In current block production designs, if Aztec Rollup is not finalizing new blocks for a prolonged period of time, it may enter Based Fallback mode. The conditions that lead to Based Fallback mode are expected to be well defined by the community of sequencers, provers, client teams and all other Aztec Rollup stakeholders and participants.

During Based Fallback mode, anyone can propose blocks if they supply proofs for these blocks alongside them. This is in contrast to the usual condition that only the sequencer assigned to a particular slot can propose blocks during that slot. This means that the inactive validator set is bypassed and anyone can advance the chain in the event of an inactive / non-participating validator set.

But terminally inactive validators must be removed from the validator set or otherwise we end up in Based Fallback too often. Slashing is a method whereby the validator set votes to "slash" the stake of inactive validators down to a point where they are kicked off the validator set. For example, if Aztec Governance sets MINIMUM_STAKING_BALANCE=50%

then as soon as 50% or more of a validator's balance is slashed, they will be kicked out of the set.

Example of Slashable Offences

1. Committee is withholding data from the provers

Provers need the transaction data (i.e. TxObjects) plus the client-side generated proofs to produce the final rollup proof, none of which are posted onchain. Client side proofs + transaction data are gossiped on the p2p instead so that committee

members can re-execute block proposals and verify that the proposed state root is correct.

Recall from the RFC on block production that the committee is a subset of validators, randomly sampled from the entire validator set. Block proposers are sampled from this committee. ⅔ + 1 of this committee must attest to L2 block proposals before they are posted to the L1 by the block proposer.

A malicious committee may not gossip the transaction data or proofs to the rest of the validator set, nor to the provers. As a result, no prover can produce the proof and the epoch in question will reorg by design. Also recall from the RFC on block production that if no proof for epoch N is submitted to L1 by the end of epoch N+1, then epoch N + any blocks built in epoch N+1 are reorged.

1. A committee proposed an invalid state root

Committee members who receive a block proposal from a proposer, must execute the block's transaction to compare the resulting state root with that contained in the proposal. A committee member should only attest to a block proposal's validity after checking for the correctness of the state root.

A committee member could skip re-executing block proposals, for a number of reasons including saving compute, faulty client software or maliciousness. This opens up the possibility of a malicious proposer posting an invalid state transition to L1 by gathering signatures from non-executing committee members. Or a malicious entity that bribes ⅔ + 1 of the committee can obtain the required signatures to post an invalid state transition.

The epoch will not be proven and will be re-orged.

1. L1 congestion has made it impossible for provers to land proofs on time

An honest prover who has funds at stake (i.e. posted a bond to produce the epoch proof), could be unable to post the proof on L1 due to congestion, an L1 reorg or just an inactivity of the L1 proposers (i.e. inactivity leak).

Therefore prover bonds should not be slashed automatically. They are kept in escrow for a suitable amount of time (i.e. 30 days) to allow sequencers to vote to slash the bond if they deem the prover to be acting maliciously.

Slashing Mechanism

In all the previous cases, it is very hard to verify and automatically slash for these events onchain. This is mainly due to the fact that neither TxObjects nor client side proofs are posted onchain. Committee members also gossip attestations on the p2p and do not post them directly on chain.

Therefore an offchain "social slashing" or "Proof of Governance Slashing" mechanism is proposed whereby the validator set votes to slash dishonest, staked network participants such as other validators and provers, based on evidence that is collected onchain + offchain, and discussed and analyzed offchain (i.e. on a community forum).

The mechanism for slashing proposals is expected to be very similar to how sequencers signal proposals to the Governance Proposer smart contract. In other words, to vote on a particular slashing vote, it is expected that node operators will change an environment var to call a function on the L1.

If enough such slashing votes are recorded, then the actor(s) are slashed. The Aztec Governance will set the proportion of the validator set votes required to pass a slashing vote. To decrease coordination costs of slashing bad actors, client teams could consider equipping the node software with the functionality to auto-vote on slashing proposals based on p2p and onchain evidence.

Section 5: FAQ

Why require proposal nominations on L1 instead of within L2 blocks?

→ The main concern is that validators that disagree with proposals could choose to censor blocks that contain nominations of proposals they disagree with i.e. refuse to attest to such blocks.

Do users vote on the L1?

→ Yes currently users are also expected to vote on L1, directly on the Governance contract.

Do validators migrate stake when a new Rollup smart contract is deployed?

→ Sequencers and provers are free to choose which version/instance of the Aztec rollup they wish to provide services to. They are not bound to what Aztec Governance deems as the canonical version.

In the event of a new Rollup smart contract, for sequencers who wish to migrate to the new rollup, the proposed system requires them to first exit stake from the previous rollup and then stake in the new Rollup smart contract. This affords some protection to sequencers against upgrades they disagree with.

However recall from the section on Staking that exits require waiting an EXIT_DELAY

to allow for slashing.

In order to minimize the amount of time it takes to move stake, an auxiliary and optional Deposit contract could be utilized that accepts deposits on behalf of Rollup smart contracts. Some sequencers may elect to auto-move in response to a governance upgrade that deploys a new Rollup smart contract.

This reduces the time required for validators to switch to provide services to new instances of the rollup without moving any Hypothetical Assets. This is fine as long as validators remain “liable” to be staked for any actions they may have done on the old rollup.

Since utilizing such a Deposit contract is optional and up to the specific instance to implement or not, implementation details are not proposed in this RFC.

Reference Links

1. RFC on Block Production
2. The Republic: A Flexible, Optional Governance Proposal with Self-Governed Portals
3. Upgrade Proposal - The Empire Stakes Back

Disclaimer