Zenith: Privacy-Preserving PoS Protocol

Zano and Common Prefix

June 23, 2026

Abstract

This paper presents a detailed description of Zenith, a pure Proof-of-Stake (PoS) protocol based on Zarcanum [sk21]. The protocol is built on the ring-signature-based anonymous payment system of Zarcanum, while adopting the epoch structure model and longest-chain approach of Ouroboros Praos [DGKR18]. The paper evaluates the protocol's security against private chain attacks using coding simulations, deriving quantitative relations between the adversarial stake threshold and transaction confirmation depth under varying conditions, including the distribution of honest stake, chain growth rate, and network delay. The analysis is extended to a cross-epoch attack in the presence of a dynamic growth-rate adjustment mechanism, yielding analogous trade-offs between adversarial power and confirmation requirements. Finally, the paper shows that a natural theoretic manipulation attack is ineffective and does not provide any adversarial advantage.

Contributors. Andrey Sabelnikov and Valeriy Pisarkov from Zano contributed by inventing the construction. Dimitris Karakostas and other Common Prefix scientists helped distill and assemble the construction into pseudocode, performed the simulations, and did the write-up.

Executive Summary

This report presents Zenith, a pure Proof-of-Stake consensus protocol for the Zano network, and provides a quantitative security analysis against its primary attack vectors. The protocol builds on established academic foundations, specifically Ouroboros Praos, and the Zarcanum payment system, preserving privacy properties while adopting a well-studied epoch and leader-election structure. The key findings are as follows:

- **The protocol is robust against specific private chain attacks across a wide parameter space.** Security thresholds depend primarily on confirmation depth and are largely invariant to implementation details such as stake distribution and number of ephemeral tests, which simplifies deployment decisions considerably.
- **Parameter selection is well-understood and actionable.** With a chain growth rate of g=5% and network delays of up to 5 slots, the protocol maintains adversary thresholds above 45% for confirmation depths of $k \geq 500$. Concrete parameter recommendations are provided in Section 4.5.
- **Two items are identified for the protocol's hardening security roadmap:** a formal security proof for the d/v-CLSAG instantiation, and evaluation of aggregation-independent eligibility. While no concrete attack has been identified in either case, both are recommended as part of the security hardening roadmap.

1 Introduction

The Zano[1] network currently relies on a hybrid Proof-of-Work (PoW) and Proof-of-Stake (PoS) consensus. This model lacks the finality, security, and sustainability guarantees of modern PoS protocols. Moreover, hybrid designs must reconcile two fundamentally different mechanisms, introducing complexity and additional attack surfaces.

This report describes how to migrate to a pure Proof-of-Stake design based on Zarcanum [sk21]. The protocol preserves Zarcanum's UTXO and key-image architecture, while adopting the epoch structure and probabilistic leader-election model of Ouroboros Praos [DGKR18].

2 Setup

The protocol follows the time model of Ouroboros Praos [DGKR18]. Specifically, time is divided into discrete units called *slots*. In each slot, the parties attempt to create a block and extend the previously-known longest chain. The epoch's execution is organized in *epochs*, which consist of a known number of slots (l).

The payment model follows CryptoNote [VS13] and RingCT [NM+16]. Specifically, each output, which models a single party, comprises a public key and a commitment to the output's stake value. When an output is spent, it is hidden among a subset of historical outputs via a ring signature, such that the fact

[1] `https://zano.org`

that it is spent is hidden. In order to ensure that an output is not spent twice, a custom object ("key image") is created using the output's private key in a deterministic manner, albeit without revealing the public key (hence the output) that corresponds to it. In addition, transaction amounts are hidden via Pedersen commitments (RingCT).

2.1 Slot setup

At a high level, on each slot, every party takes the following steps to create a block:[2]

1. $\mathcal{P}$ creates an "ephemeral" key image, i.e., a hash-based object which is deterministically derived from the output's private key and the epoch's slot.
2. If the ephemeral key image falls below some threshold, which is computed using the output's stake value, then $\mathcal{P}$ can create a block for the slot as follows:
   (a) $\mathcal{P}$ collects a subset of the snapshot's UTxOs including the chosen output.
   (b) $\mathcal{P}$ creates a block with the following proofs:
      i. A proof that $\mathcal{P}$ knows the spending key of an output in the chosen subset.
      ii. A zero-knowledge proof that the stake value of the same output from Step 2(b)i was above the correct threshold.
   (c) $\mathcal{P}$ signs the block by turning the proofs from Step 2b to a signature of knowledge, proving that the private signing key is the same as the spending key of the output from Step 2(b)i.

2.2 Epoch setup

Every epoch e is associated with a nonce R_e. The nonce is computed as follows:

- Let:
  - $\mathcal{C}_{\mathcal{P}}$ be the chain of party $\mathcal{P}$;
  - $\mathsf{sl}_i, \ldots, \mathsf{sl}_{i+l}$ be the slots of epoch $e-2$;
  - $\mathcal{B}_a || \ldots || \mathcal{B}_b$ be the sub-chain of $\mathcal{C}_{\mathcal{P}}$ which consists of all blocks created in the slots of epoch $e-2$.
  - $\mathcal{I}_{\mathcal{B}}$ be the key image used for the creation of block $\mathcal{B}$;
- The nonce of epoch e is computed by hashing the previous epoch's nonce along with the key images of all blocks in epoch $e-2$: $R_e = \mathcal{H}(R_{e-1}, \langle \mathcal{I}_{\mathcal{B}_a}, \ldots, \mathcal{I}_{\mathcal{B}_b} \rangle)$.

[2] If $\mathcal{P}$ controls multiple UTxOs, then the steps are executed for every output.

Note on epoch randomness R_e: In order to ensure that the key images, epoch nonces and values used in leader election are not correlated, different hash functions $\mathcal{H}'$ and $\mathcal{H}''$ should be used when computing R_e and h_i. This can be achieved via domain separation, *i.e.* adding a specific string as a prefix to the input of the hash function $\mathcal{H}$. Concretely, this can be done by computing R_e using a hash function $\mathcal{H}'(x) = \mathcal{H}(\mathsf{NONCE} \,\|\, x)$, while the value h_i is computed with a hash function $\mathcal{H}''(x) = \mathcal{H}(\mathsf{LEADER} \,\|\, x)$.

Moreover, in order to mitigate potential attacks that introduce bias in R_e, it can be computed as in Ouroboros Praos. In order to do so, each block will include an extra value that behaves as VRF output computed as $h'_i = \mathcal{H}(\mathsf{NONCE} \,\|\, P_{\mathsf{utxo}} \,\|\, x_{\mathsf{utxo}} \cdot \mathcal{H}_p(\mathsf{NONCE} \,\|\, R_e \,\|\, \mathsf{sl}_r))$. The epoch randomness can be obtained by computing $R_e = \mathcal{H}(\mathsf{NONCE} \,\|\, h'_{\mathcal{B}_a} \,\|\, \cdots \,\|\, h'_{\mathcal{B}_b})$. A value R_e computed following this procedure would match the distribution of the Ouroboros Praos randomness beacon under common prefix, chain quality, and chain growth guarantees. Formally establishing these properties for Zarcanum, which depends on resolving the aggregation independence question discussed below, is identified as part of the protocol's security hardening roadmap. This is consistent with the open items noted throughout this report.

Staking Outputs: For every epoch the set of outputs that are eligible for participation is explicitly defined. Specifically, each output in the system contains a bit, which defines whether the output is a regular payment one, or a "staking" output.

A staking output should be identifiably spent. In other words, it should be possible for everyone to identify whether a staking output has been consumed. For example, a straightforward way to implement this is to not use any decoys in a transaction that consumes a staking output. With this information, it is straightforward for the consensus protocol to identify which staking outputs are unspent at the time when an epoch's snapshot is taken, such that only these outputs are eligible for participation in consensus.

The consequence of splitting the system's outputs in two sets (payment and staking) is that they cannot be combined when constructing ring signatures. Specifically, during a regular transaction, when a payment output is consumed, no staking outputs can be used in the set of decoys of the transaction's ring signature. Equivalently, when a block is created, no payment outputs can be used as decoys in the block's ring proof.

3 Protocol Construction

This protocol is run by every party $\mathcal{P}$ on every epoch e, which is parameterized by:

1. a hash function $\mathcal{H}$, which outputs an element of the field
2. a hash function $\mathcal{H}_p$, which outputs an EC point
3. the total stake Σ
4. a nonce R_e
5. a set of "staking" outputs $\mathbb{O}_e$

6. generators $G, X, H, \{H_t\}_{t\in[1,\dots]}$[3]
7. an active slot coefficient[4] g, which sets the "difficulty" parameter for the epoch: $D = \frac{1}{g} \cdot \Sigma$

Additionally, the following hold:

- for each output utxo: $0 \leq a_{\mathsf{utxo}} < 2^{64}$
- $D \geq 2^{64}$

3.1 Setup

Each party $\mathcal{P}$ controls an unspent output utxo, which is associated with a vector of private elements:

1. Keys:
   - s_{utxo} is the output receiver's private "spending" key
   - v_{utxo} is the output receiver's private "view" key
   - r_{utxo} is a private element shared between the output's creator and receiver
   - $x_{\mathsf{utxo}} = \mathcal{H}(v_{\mathsf{utxo}} \cdot r_{\mathsf{utxo}} \cdot G) + s_{\mathsf{utxo}}$ is the output's one-time spending key
2. Assets:
   - a_{utxo} is the output's amount of assets
   - f_{utxo} is a random scalar used for the commitment of the output's amount
   - H_t is the generator that corresponds to the output's assets
   - z_{utxo} is a private element shared between the output's creator and receiver
3. Staking-related elements:
   - $q_{\mathsf{utxo}} = v_{\mathsf{utxo}} \cdot \mathcal{H}(r_{\mathsf{utxo}} \cdot v_{\mathsf{utxo}} \cdot G)$ is a "concealing" scalar

Each output utxo is also publicly associated with the following elements:

1. a public key $P_{\mathsf{utxo}} = x_{\mathsf{utxo}} \cdot G$
2. a "blinded asset tag" $T_{\mathsf{utxo}} = H_t + z_{\mathsf{utxo}} \cdot X$
3. a commitment to the amount $A_{\mathsf{utxo}} = a_{\mathsf{utxo}} \cdot T_{\mathsf{utxo}} + f_{\mathsf{utxo}} \cdot G$
4. a public "concealing point" $Q_{\mathsf{utxo}} = q_{\mathsf{utxo}} \cdot G$

[3] The system supports multiple assets, each identified by an integer t. Each generator H_t is computed deterministically, based on a unique description of the corresponding asset.

[4] g defines the rate with which blocks are produced per slot. For example, if $g = 0.05$ then a block is created on expectation every 20 slots.

3.2 Eligibility Test

On a slot sl_r, $\mathcal{P}$ with output utxo computes the following:

1. "Ephemeral" key image: $\mathcal{I} = x_{\mathsf{utxo}} \cdot \mathcal{H}_p(P_{\mathsf{utxo}} \,\|\, R_e \,\|\, \mathsf{sl}_r)$.
2. N "stake kernels": $\forall i \in [1, N] : \mathsf{K}_i = R_e \,\|\, \mathsf{sl}_r \,\|\, \mathcal{I} \,\|\, i$.
3. Each stake kernel's hash: $\forall i \in [1, N] : h_i = \mathcal{H}(\mathsf{K}_i)$.
4. Retrieves the ephemeral key image of the local chain's last block $\mathcal{I}_\mathcal{C}$ and computes: $f' = \mathcal{H}(\mathcal{I}_\mathcal{C} \,\|\, \mathsf{sl}_r)$.

Then, $\mathcal{P}$ checks the following inequality for each stake kernel:

$$h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l < \lfloor \frac{l}{D} \rfloor \cdot a_{\mathsf{utxo}} \tag{1}$$

where l is the order of the main subgroup.

If, for some stake kernel K_i the inequality holds, then utxo is eligible to create a block on slot sl_r. In this case, the party proceeds creation of the eligibility proof, as follows.

$\mathcal{P}$ transforms 1 into an equality as follows:

$$\begin{aligned} h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l = (d \cdot a_{\mathsf{utxo}} - b_\alpha) \mod l \Rightarrow \\ b_\alpha = d \cdot a_{\mathsf{utxo}} - (h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l \end{aligned} \tag{2}$$

Now, $\mathcal{P}$'s goal is to prove that, for some $1 \leq d \leq \lfloor \frac{l}{D} \rfloor$, it holds:

$$b_\alpha = d \cdot a_{\mathsf{utxo}} - h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l < 2^{64} \tag{3}$$

without revealing $\mathsf{utxo}, f_{\mathsf{utxo}}, q_{\mathsf{utxo}}, a_{\mathsf{utxo}}$.

Note on value h_i: Since the value h_i is obtained by computing $\mathcal{H}(R_e \,\|\, \mathsf{sl}_r \,\|\, x_{\mathsf{utxo}} \cdot \mathcal{H}_p(P_{\mathsf{utxo}} \,\|\, R_e \,\|\, \mathsf{sl}_r) \,\|\, i)$, it behaves as the output of a Verifiable Random Function with unpredictability under malicious key generation similar to the VRF used in Ouroboros Praos. The values x_{utxo} and P_{utxo} behave as secret key and public key, respectively. A key difference with respect to the Ouroboros Praos VRF is that P_{utxo} is used in computing $\mathcal{H}_p(P_{\mathsf{utxo}} \,\|\, R_e \,\|\, \mathsf{sl}_r)$. The computation of h_i could be slightly modified and significantly simplified to match exactly the computation of the Oruoboros Praos VRF by setting $h_i = \mathcal{H}(P_{\mathsf{utxo}} \,\|\, x_{\mathsf{utxo}} \cdot \mathcal{H}_p(R_e \,\|\, \mathsf{sl}_r))$. Formally establishing that h_i is correctly distributed depends on guaranteeing that the epoch randomness Re has bounded bias, which in turn depends on resolving the aggregation independence question identified in the security hardening roadmap.

Note on eligibility test: Notice that the eligibility test in Zarcanum is not aggregation independent as defined in Ouroboros Praos. This means that the adversary may have an advantage in being elected as leader if it concentrates its entire stake in a few outputs, in comparison to honest parties who have their stake spread in a many outputs with low amounts. The main issue in this case is that such attacks can also be done adaptively (moving stake while the protocol is executed) in more complex patterns and while biasing the epoch randomness. Aggregation-independent eligibility checks, as defined in Ouroboros Praos, would

provide stronger formal guarantees against stake-concentration strategies. This represents a potential direction for strengthening the protocol's security model in future iterations, and is worth evaluating against the implementation complexity it would introduce.

Note on concealing point: The concealing element Q_{utxo} is needed here to prevent the creator of utxo from identifying that utxo was used for the proof creation. Specifically, the creator of utxo is the party $\mathcal{P}'$ that send the assets held by utxo to $\mathcal{P}$. During the creation of utxo, $\mathcal{P}'$ chose f_{utxo} and also knew the amount a_{utxo}, while also h_i and f' are public. Now assume that q_{utxo} didn't exist, so Equation (3) was $b_\alpha = h_i \cdot (f_{\mathsf{utxo}} + f') - d \cdot a_{\mathsf{utxo}} < 2^{64}$. In this case, when observing a block whose ring proof includes an output utxo that $\mathcal{P}'$ created, $\mathcal{P}'$ could locally check if $h_i \cdot (f_{\mathsf{utxo}} + f') - d \cdot a_{\mathsf{utxo}} < 2^{64}$; if so, then $\mathcal{P}'$ could deduce that utxo was used to create said block with some confidence. Instead, by using q_{utxo} in Equation (3), it is not possible to do this check anymore since $\mathcal{P}'$ doesn't know q_{utxo}.

3.3 Eligibility Proof

First, $\mathcal{P}$ sets:

$$d = \lfloor \frac{h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l}{a_{\mathsf{utxo}}} \rfloor + 1$$

and

$$b_\alpha = d \cdot a_{\mathsf{utxo}} - (h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l)$$

Second, $\mathcal{P}$ picks random non-zero scalars x, x', x'' and computes:

- $b_f = d \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') - h_i \cdot a_{\mathsf{utxo}}$
- $b_x = x'' - h_i \cdot x' + d \cdot x$
- $C = x \cdot X + a_{\mathsf{utxo}} \cdot H + (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \cdot G$
- $C' = x' \cdot X + (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \cdot H + a_{\mathsf{utxo}} \cdot G$
- $E = b_\alpha \cdot H + b_f \cdot G + b_x \cdot X$
- $F = h_i \cdot C' - d \cdot C + E$

Third, $\mathcal{P}$ collects a set of outputs $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$, which includes the chosen output utxo along with a number of decoy outputs.

Fourth, $\mathcal{P}$ creates the following proofs:

- Ring-based proof, which proves that for one of the outputs in $\mathbb{O}_{\mathsf{block}}$:
  - The prover knows $x, a_{\mathsf{utxo}}, f_{\mathsf{utxo}}, q_{\mathsf{utxo}}, H_t, z_{\mathsf{utxo}}$ of the output.
  - The output's asset tag is $H_t = H$.
  - For commitment C, the G coefficient is $(f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f')$ and the H coefficient is a_{utxo}.
- Schnorr proofs, which prove that:
  - C, C' are "mirror" commitments w.r.t. generators G, H.

– $F = h_i \cdot C' - d \cdot C + E$ has only a X coefficient.

- Range proof, which proves that the value committed in E (under the coefficient of generator H) is strictly less than 2^{64}.

Finally, $\mathcal{P}$ reveals all of the above proofs, along with d, C, C', E.
Intuitively, the flow of logic behind the proofs is as follows:

1. The ring proof proves that, for one of the outputs in the ring, (i) the block producer knows the output's private values; (ii) these private values are used to compute the G, H coefficients of C as above; (iii) the output's asset tag is H.
2. The first Schnorr proof proves that C' is the well-constructed from C w.r.t. G, H, i.e., the G coefficient of C is the H coefficient of C' and vice versa.
3. From Steps 1 and 2, it is deduced that the H coefficient of $d \cdot C - h_i \cdot C'$ is $h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') - d \cdot a_{\mathsf{utxo}}$, where $f_{\mathsf{utxo}}, q_{\mathsf{utxo}}, a_{\mathsf{utxo}}$ are the chosen output's private values.
4. The second Schnorr proof proves that E and $d \cdot C - h_i \cdot C'$ have the same H coefficient.
5. The range proof proves that Inequality (3) holds for the H coefficient of E.
6. From Steps 4 and 5, it is deduced that Inequality (3) holds for the private values of the chosen output.

In the following paragraphs we detail the implementation of all of the above proofs.

3.3.1 Ring Proof

First, $\mathcal{P}$ sets the block's payload $\mathcal{B}$, which contains transactions, timestamps etc.
The ring proof is defined in 1. The proof is an instance of a $^{\mathrm{d}}/_{\mathrm{v}}$-CLSAG [sow25].

Definition 1 *The Zarcanum signature scheme consists of the tuple* (Setup, KeyGen, Sign, Verify)*:*

- Setup $\rightarrow$ *par:* Setup *selects: (i) a prime p; (ii) group generators $G, X, \{H_t\}_{t\in[0,\dots]}$; (iii) hash functions $\mathcal{H}, \mathcal{H}_p$ (modeled as random oracles); (iv) domain separating tags (strings)* $\mathsf{dsep}, \mathsf{dsep}_0, \mathsf{dsep}_1, \mathsf{dsep}_2, \mathsf{dsep}_3, \mathsf{dsep}_4$ *Then,* Setup *outputs* $par = (p, G, X, \{H_t\}_{t\in[0,\dots]}, \mathcal{H}, \mathcal{H}_p, \mathsf{dsep}, \mathsf{dsep}_0, \mathsf{dsep}_1, \mathsf{dsep}_2, \mathsf{dsep}_3, \mathsf{dsep}_4)$.
- KeyGen $\rightarrow (\mathsf{sk}, \mathsf{pk})$*: When queried for a new key,* KeyGen *samples a fresh secret key and computes the associated public key:*[5]

$$\mathsf{sk} = (x_{\mathsf{utxo}}, a_{\mathsf{utxo}}, f_{\mathsf{utxo}}, H_t, z_{\mathsf{utxo}}, q_{\mathsf{utxo}})$$
$$\mathsf{vk} = (P_{\mathsf{utxo}}, T_{\mathsf{utxo}}, A_{\mathsf{utxo}}, Q_{\mathsf{utxo}}) = (x_{\mathsf{utxo}} \cdot G, H_t + z_{\mathsf{utxo}} \cdot X, a_{\mathsf{utxo}} \cdot (H_t + z_{\mathsf{utxo}} \cdot X) + f_{\mathsf{utxo}} \cdot G, q_{\mathsf{utxo}} \cdot G)$$

- $\mathsf{Sign}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \mathsf{sk}) \rightarrow \sigma$*:* Sign *parses the following:*
  – $\langle \mathcal{B}, f', R_e, \mathsf{sl}_r \rangle \leftarrow \mathcal{M}$

[5]Here, the "public key" is a UTxO, so the terms "key" and "KeyGen" refer to digital signature terminology.

- $\langle \mathsf{vk}_0, \ldots, \mathsf{vk}_{n-1} \rangle = \langle (P_0, T_0, A_0, Q_0), \ldots, (P_{n-1}, T_{n-1}, A_{n-1}, Q_{n-1}) \rangle \leftarrow \mathbb{O}_{\mathsf{block}}$
- $(x_{\mathsf{utxo}}, a_{\mathsf{utxo}}, f_{\mathsf{utxo}}, H, z_{\mathsf{utxo}}, q_{\mathsf{utxo}}) \leftarrow \mathsf{sk}$

Next:

- Sign *picks random scalars* $x, \hat{f}$ *and computes:*
  - $C = x \cdot X + a_{\mathsf{utxo}} \cdot H + (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \cdot G$
  - $A^{(P)} = a_{\mathsf{utxo}} \cdot T_{\mathsf{utxo}} + \hat{f} \cdot G$
- Sign *finds the index* π*, such that* $\mathsf{vk}_\pi = (x_{\mathsf{utxo}} \cdot G, H + z_{\mathsf{utxo}} \cdot X, a_{\mathsf{utxo}} \cdot T_{\mathsf{utxo}} + f_{\mathsf{utxo}} \cdot G, q_{\mathsf{utxo}} \cdot G)$.
- Sign *computes the following:*
  - *Key images (main and auxiliary):*
    - $\mathcal{I}_{base} = \mathcal{H}_p(P_{\mathsf{utxo}} \,\|\, R_e \,\|\, \mathsf{sl}_r)$
    - $\mathcal{I} = x_{\mathsf{utxo}} \cdot \mathcal{I}_{base}$
    - $\mathcal{K}_1 = (f_{\mathsf{utxo}} - \hat{f}) \cdot \mathcal{I}_{base}$
    - $\mathcal{K}_2 = z_{\mathsf{utxo}} \cdot \mathcal{I}_{base}$
    - $\mathcal{K}_3 = (x - a_{\mathsf{utxo}} \cdot z_{\mathsf{utxo}}) \cdot \mathcal{I}_{base}$
    - $\mathcal{K}_4 = q_{\mathsf{utxo}} \cdot \mathcal{I}_{base}$
  - *Input hash and aggregation coefficients:*
    - $\mathcal{V} = \mathcal{H}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C)$
    - $\mu_0 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_0)$
    - $\mu_1 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_1)$
    - $\mu_2 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_2)$
    - $\mu_3 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_3)$
    - $\mu_4 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_4)$
  - *Aggregate key images for the* $\{G, X\}$*-layers:*
    - $\mathcal{W}_g = \mu_0 \cdot \mathcal{I} + \mu_1 \cdot \mathcal{K}_1 + \mu_4 \cdot \mathcal{K}_4$
    - $\mathcal{W}_x = \mu_2 \cdot \mathcal{K}_2 + \mu_3 \cdot \mathcal{K}_3$
  - *Aggregate secret keys for the* $\{G, X\}$*-layers:*
    - $x_g = \mu_0 \cdot x_{\mathsf{utxo}} + \mu_1 \cdot (f_{\mathsf{utxo}} - \hat{f}) + \mu_4 \cdot q_{\mathsf{utxo}}$
    - $x_x = \mu_2 \cdot z_{\mathsf{utxo}} + \mu_3 \cdot (x - a_{\mathsf{utxo}} \cdot z_{\mathsf{utxo}})$
- Sign *picks random* α_g, α_x *and computes:*
  - $L_\pi^1 = \alpha_g \cdot G$
  - $L_\pi^2 = \alpha_g \cdot \mathcal{I}_{base}$
  - $L_\pi^3 = \alpha_x \cdot X$
  - $L_\pi^4 = \alpha_x \cdot \mathcal{I}_{base}$
  - $c_{\pi+1} = \mathcal{H}(\mathsf{dsep}, \mathcal{V}, L_\pi^1, L_\pi^2, L_\pi^3, L_\pi^4)$
- Sign *sets* $i = (\pi + 1) \mod n$ *and does the following, until* $i = \pi$*:*
  - *Picks random* r_i^g, r_i^x*.*
  - *Computes:*
    - $W_i^g = \mu_0 \cdot P_i + \mu_1 \cdot (A_i - A^{(P)}) + \mu_4 \cdot Q_i$
    - $W_i^x = \mu_2 \cdot (T_i - H) + \mu_3 \cdot (C - A_i - Q_i - f' \cdot G)$

· $L_i^1 = r_i^g \cdot G + c_i \cdot W_i^g$
· $L_i^2 = r_i^g \cdot \mathcal{H}_p(P_i \,\|\, R_e \,\|\, sl_r) + c_i \cdot \mathcal{W}_g$
· $L_i^3 = r_i^x \cdot X + c_i \cdot W_i^x$
· $L_i^4 = r_i^x \cdot \mathcal{H}_p(P_i \,\|\, R_e \,\|\, sl_r) + c_i \cdot \mathcal{W}_x$
· $c_{i+1} = \mathcal{H}(\mathsf{dsep}, \mathcal{V}, L_1, L_2, L_3, L_4)$

* *Sets:* $i = (i+1) \mod n$.

– Sign *computes:*

* $r_\pi^g = \alpha_g - c_\pi \cdot x_g$
* $r_\pi^x = \alpha_x - c_\pi \cdot x_x$

– *Finally,* Sign *outputs:*

$$\sigma_{\mathsf{ring}} = (c_0, \{r_i^g, r_i^x\}_{i=0}^{n-1}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C)$$

- $\mathsf{Verify}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma) \to \{0,1\}$*:* Verify *takes as input a message* $\mathcal{M}$*, a matrix* $\mathbb{O}_{\mathsf{block}}$*, and a signature* σ *and then:*

– Verify *parses the following:*

* $(c_0, \{r_i^g, r_i^x\}_{i=0}^{n-1}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C) \leftarrow \sigma_{\mathsf{ring}}$
* $\langle (P_0, T_0, A_0, Q_0), \ldots, (P_{n-1}, T_{n-1}, A_{n-1}, Q_{n-1}) \rangle \leftarrow \mathbb{O}_{\mathsf{block}}$
* $\langle \mathcal{B}, f', R_e, \mathsf{sl}_r \rangle \leftarrow \mathcal{M}$

– Verify *computes:*

* $\mathcal{V} = \mathcal{H}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C)$
* $\mu_0 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_0)$
* $\mu_1 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_1)$
* $\mu_2 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_2)$
* $\mu_3 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_3)$
* $\mu_4 = \mathcal{H}(\mathcal{V}, \mathsf{dsep}_4)$
* $\mathcal{W}_g = \mu_0 \cdot \mathcal{I} + \mu_1 \cdot \mathcal{K}_1 + \mu_4 \cdot \mathcal{K}_4$
* $\mathcal{W}_x = \mu_2 \cdot \mathcal{K}_2 + \mu_3 \cdot \mathcal{K}_3$

– Verify *sets* $c_0' = c_0$.

– *For every* $i \in [0, \ldots, n-1]$, Verify *computes:*

* $W_i^g = \mu_0 \cdot P_i + \mu_1 \cdot (A_i - A^{(P)}) + \mu_4 \cdot Q_i$
* $W_i^x = \mu_2 \cdot (T_i - H) + \mu_3 \cdot (C - A_i - Q_i - f' \cdot G)$
* $L_i^1 = r_i^g \cdot G + c_i \cdot W_i^g$
* $L_i^2 = r_i^g \cdot \mathcal{H}_p(P_i \,\|\, R_e \,\|\, sl_r) + c_i \cdot \mathcal{W}_g$
* $L_i^3 = r_i^x \cdot X + c_i \cdot W_i^x$
* $L_i^4 = r_i^x \cdot \mathcal{H}_p(P_i \,\|\, R_e \,\|\, sl_r) + c_i \cdot \mathcal{W}_x$
* $c_{i+1} = \mathcal{H}(\mathsf{dsep}, \mathcal{V}, L_1, L_2, L_3, L_4)$

– *If* $c_0 = c_n'$ Verify *outputs* 1*, otherwise it outputs* 0.

Note on aggregation: Using random aggregation coefficients to compute aggregate key images and aggregate secret keys as random linear combinations of the key images and secret keys in the ring deviates from standard techniques for constructing ring signatures based on the OR-composition of statements proven via Sigma protocols. While no concrete attack has been identified in this instance, a formal security proof for this specific instantiation of d/v-CLSAG remains open. We recommend this be addressed in a dedicated security analysis as part of the protocol's maturation roadmap to avoid known pitfalls [DEF+19].

3.3.2 Schnorr Proofs

As discussed above, we need two Schnorr proofs which prove that:

1. C, C' are "mirror" commitments w.r.t. generators G, T_t.
2. F has only a X coefficient.

All of these can be directly implemented as the standard Schnorr proof of knowledge protocol [Sch89] and be made non-interactive using the Fiat-Shamir transformation [FS86]. For completeness we describe each of the three proofs below.

First Schnorr proof Given C, C' and G_1, G_2, G_3, party $\mathcal{P}$ wants to prove knowledge of (x, x', a, b) such that:

- $C = x \cdot G_1 + a \cdot G_2 + b \cdot G_3$
- $C' = x' \cdot G_1 + b \cdot G_2 + a \cdot G_3$

The non-interactive prover proceeds as follows:

- Pick random s_0, s_1, r_0, r_1.
- Compute:
  - $R_0 = s_0 \cdot G_1 + r_0 \cdot (G_2 + G_3)$
  - $R_1 = s_1 \cdot G_1 + r_1 \cdot (G_2 - G_3)$
- Set $c = \mathcal{H}(R_0 \,\|\, R_1 \,\|\, C \,\|\, C')$.
- Compute:
  - $y_0 = r_0 + c \cdot (a + b)$
  - $y_1 = r_1 + c \cdot (a - b)$
  - $z_0 = s_0 + c \cdot (x + x')$
  - $z_1 = s_1 + c \cdot (x - x')$
- Reveal proof $\langle c, y_0, y_1, z_0, z_1 \rangle$

The verifier computes:

- $R_0 = y_0 \cdot (G_2 + G_3) + z_0 \cdot G_1 - c \cdot (C + C')$
- $R_1 = y_1 \cdot (G_2 - G_3) + z_1 \cdot G_1 - c \cdot (C - C')$

and checks $\mathcal{H}(R_0 \,\|\, R_1 \,\|\, C \,\|\, C') \stackrel{?}{=} c$.

Second Schnorr proof Given C and G, party $\mathcal{P}$ wants to prove knowledge of x such that $C = x \cdot G$.

The non-interactive prover proceeds as follows:

- Pick random r.
- Compute $R = r \cdot G$
- Set $c = \mathcal{H}(R \,\|\, C)$.
- Compute $y = r + c \cdot a$.
- Reveal proof $\langle c, y \rangle$.

The verifier computes $R = y \cdot G - c \cdot C$ and checks: $\mathcal{H}(R \,\|\, C) \stackrel{?}{=} c$.

3.3.3 Range Proof

The final proof is a range proof, which proves that the coefficient b_α of generator T_t in the commitment E is strictly smaller than 2^{64}.

Because E is double-blinded, meaning it consists of three generators, a standard range proof like Bulletproofs [BBB+17] cannot be directly applied here.

Nonetheless, this can be easily resolved by creating a fresh commitment $B = b_\alpha \cdot H + e \cdot G$ (where e is picked at random). Given B, $\mathcal{P}$ needs to prove that E, B have the same H coefficient. This can be easily done with a Schnorr proof which proves that the aggregate commitment $E - B$ has only G, X coefficients (see below).

Afterwards, $\mathcal{P}$ can directly create a standard range proof which shows that the T_t of B is in the range $[0, 2^{64})$.

Third Schnorr proof Given C, C' and G_1, G_2, G_3, party $\mathcal{P}$ wants to prove knowledge of (a, x, x', y) such that:

- $C = a \cdot G_1 + x \cdot G_2 + y \cdot G_3$
- $C' = a \cdot G_1 + x' \cdot G_2$

The non-interactive prover proceeds as follows:

- Pick random r, r'.
- Compute $R = r \cdot G_2 + r' \cdot G_3$.
- Set $c = \mathcal{H}(R \,\|\, C \,\|\, C')$.
- Compute:
  - $s = r + c \cdot (x - x')$
  - $s' = r' + c \cdot y$
- Reveal proof $\langle c, s, s' \rangle$

The verifier computes $R = s \cdot G_2 + s' \cdot G_3 - c \cdot (C - C')$ and checks $\mathcal{H}(R \,\|\, C \,\|\, C') \stackrel{?}{=} c$.

3.4 Protocol Definition

The full Zenith protocol is defined in Figures 1-4.

First, Figure 1 outlines the setup of the protocol. It describes the global parameters to which all parties have access, it defines the structure of the outputs (UTxOs) that are eligible for consensus participation, how each epoch's nonce is generated and how the epoch's difficulty is set. Essentially, this step describes all information needed for a party to run the consensus protocol within each epoch.

Protocol Zenith (Setup)

Let L denote the number of slots in each epoch and g denote the target active slot coefficient.
This protocol is run by every party $\mathcal{P}$ on every epoch e with snapshot set $\mathbb{O}_e$ (staking UTxOs up to epoch $e-2$). A party controls an output $\mathsf{utxo} \in \mathbb{O}_e$ if it knows the secret key $\mathsf{sk} = (x_{\mathsf{utxo}}, a_{\mathsf{utxo}}, f_{\mathsf{utxo}}, H_t, z_{\mathsf{utxo}}, q_{\mathsf{utxo}})$ connected to the output. Parties proceed as follows:

- **Setup:** Parties have access to an initialization functionality $\mathcal{F}_{\mathsf{INIT}}$, which holds system parameters p for the ring proof of Section 3.3.1, the (confidential) initial stake distribution and the genesis block containing RingCT transactions following this stake distribution. In the beginning of the execution, every party $\mathcal{P}$ calls $\mathcal{F}_{\mathsf{INIT}}$ with $(\mathsf{init}, \mathcal{P})$, receiving as response parameters p, the genesis block $\mathcal{B}_0$ and the secret key $\mathsf{sk} = (x_{\mathsf{utxo}}, a_{\mathsf{utxo}}, f_{\mathsf{utxo}}, H_t, z_{\mathsf{utxo}}, q_{\mathsf{utxo}})$ corresponding to a UTxO $\mathsf{utxo} = (x_{\mathsf{utxo}} \cdot G, H_t + z_{\mathsf{utxo}} \cdot X, a_{\mathsf{utxo}} \cdot (H_t + z_{\mathsf{utxo}} \cdot X) + f_{\mathsf{utxo}} \cdot G, q_{\mathsf{utxo}} \cdot G)$ contained in $\mathcal{B}_0$. Additionally, each party $\mathcal{P}$ initializes their local chain $\mathcal{C}_{\mathcal{P}}$: $\mathcal{C}_{\mathcal{P}} = \mathcal{B}_0$.
- **Epoch Nonce Generation:** Let $\mathbb{I}$ be the (ordered) list of the key images of all blocks created in epoch $e-1$. Each party $\mathcal{P}$ computes the nonce R_e of epoch e as follows: $R_e = \mathcal{H}(R_{e-1} \,\|\, \mathbb{I})$.
- **Difficulty Recalibration:** At the beginning of each epoch e, every party $\mathcal{P}$ recalibrates the difficulty parameter D for epoch e as follows:
  1. Count the number of blocks B_{e-2} produced in the local chain $\mathcal{C}_{\mathcal{P}}$ during epoch $e-2$.
  2. Compute the observed chain growth rate for epoch $e-2$: $g_{\mathrm{obs},e-2} = \frac{B_{e-2}}{L}$
  3. Compute the new difficulty for epoch e: $D_e = D_{e-2} \cdot \frac{g_{\mathrm{obs},e-2}}{g}$

Figure 1: Protocol Zenith (setup)

Figure 2 outlines the protocol run by a participant $\mathcal{B}$ in order to generate a block during some epoch.

Next, Figure 3 describes the process for validating a new block that is created during some epoch.

Finally, Zenith can directly use the fork choice rule of Ouroboros Genesis [BGK+18] (Figure 4). This rule states that for long range forks, the party picks the chain which grew at a faster rate immediately after the point of the fork. As shown in [BGK+18], under honest majority this rule enables a party to pick the correct chain with access only to the genesis block, that is without requiring any recent information (such as checkpoints).

Protocol Zenith (Block Creation)

- **Block Creation:** On a slot sl_r, for every unspent output utxo in the epoch's snapshot $\mathbb{O}_e$ that party $\mathcal{P}$ controls, it proceeds as follows:
  1. Computes an "ephemeral key image": $\mathcal{I} = x_{\mathsf{utxo}} \cdot \mathcal{H}_p(P_{\mathsf{utxo}} \,\|\, R_e \,\|\, \mathsf{sl}_r)$.
  2. Compute N "stake kernels": $\forall i \in [1, N] : \mathsf{K}_i = R_e \,\|\, \mathcal{I} \,\|\, \mathsf{sl}_r \,\|\, i$.
  3. Compute each stake kernel's hash: $\forall i \in [1, N] : h_i = \mathcal{H}(\mathsf{K}_i)$.
  4. Retrieve the ephemeral key image $\mathcal{I}_{\mathcal{C}_\mathcal{P}}$ of the last block in $\mathcal{C}_\mathcal{P}$ and compute: $f' = \mathcal{H}(\mathcal{I}_{\mathcal{C}_\mathcal{P}} \,\|\, \mathsf{sl}_r)$.
  5. If, for some stake kernel K_i, it holds $h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l < \lfloor \frac{l}{D} \rfloor \cdot a_{\mathsf{utxo}}$, then utxo is eligible for creating a block on slot sl_r, so $\mathcal{P}$ proceeds. Otherwise, $\mathcal{P}$ ignores the next steps, advancing to the next $\mathsf{utxo} \in \mathbb{O}_e$.
  6. Set the block payload block (transactions, metadata, timestamps, ...).
  7. Set $\mathcal{M} = \langle \mathsf{block}, f', R_e, \mathsf{sl}_r \rangle$.
  8. Compute:
     - $d = \lfloor \frac{h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l}{a_{\mathsf{utxo}}} \rfloor + 1$.
     - $b_\alpha = d \cdot a_{\mathsf{utxo}} - (h_i \cdot (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \mod l)$
     - $C = x \cdot X + a_{\mathsf{utxo}} \cdot H + (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \cdot G$
     - $C' = x' \cdot X + (f_{\mathsf{utxo}} + q_{\mathsf{utxo}} + f') \cdot H + a_{\mathsf{utxo}} \cdot G$
     - $E = b_\alpha \cdot H + b_f \cdot G + b_x \cdot X$
     - $F = h_i \cdot C' - d \cdot C + E$
  9. Sample a random subset of outputs $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$ such that $\mathsf{utxo} \in \mathbb{O}_{\mathsf{block}}$ and generate a ring proof as in Definition 1: $\mathsf{Sign}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \mathsf{sk}) \to \sigma_{\mathsf{ring}}$, where $\sigma_{\mathsf{ring}} = (c_0, \{r_i^g, r_i^x\}_{i=0}^{n-1}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C)$.
  10. Compute Schnorr proof $\pi_{\mathsf{s1}} = (C', \langle c, y_0, y_1, z_0, z_1 \rangle)$, which proves that C, C' are "mirror" commitments (Section 3.3.2).
  11. Compute Schnorr proof $\pi_{\mathsf{s2}} = (E, \langle c_e, y_e \rangle)$, which proves that F has only a X coefficient (Section 3.3.2).
  12. Generate Schnorr proof $\pi_{\mathsf{s3}} = (B, \langle c_b, s_b, s'_b \rangle)$ and range proof π_{range} (Section 3.3.3), which prove that commitment $\mathcal{E}$ contains $b_\alpha < 2^{64}$ under generator H.
  13. Diffuse the final block $\mathcal{B} = \langle \mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma_{\mathsf{ring}}, \pi_{\mathsf{s1}}, \pi_{\mathsf{s2}}, \pi_{\mathsf{s3}}, \pi_{\mathsf{range}} \rangle$ on the network.

Figure 2: Protocol Zenith (block creation)

3.5 Block construction

After the party $\mathcal{P}$ has constructed the eligibility proof (cf. Section 3.2), it collects all of its elements as follows:

- block: block's payload (transactions, timestamps, etc)
- $\mathcal{M} = \langle \mathsf{block}, f', R_e, \mathsf{sl}_r \rangle$
- $\mathbb{O}_{\mathsf{block}} = \langle (P_0, T_0, A_0, Q_0), \ldots, (P_{n-1}, T_{n-1}, A_{n-1}, Q_{n-1}) \rangle$: list of outputs

Protocol Zenith (Block Verification)

- **Block Verification:** For every block $\mathcal{B} = \langle \mathsf{block}, \mathbb{O}_{\mathsf{block}}, \sigma_{\mathsf{ring}}, \pi_{\mathsf{s1}}, \pi_{\mathsf{s2}}, \pi_{\mathsf{s3}}, \pi_{\mathsf{range}}, \rangle$ received in epoch e, proceed as follows:
  1. Parse $\langle \mathsf{block}, f', R_e, \mathsf{sl}_r \rangle \leftarrow \mathcal{M}$ and check that it extends the local chain according to the chain rule of Figure 4.
  2. Check that $\mathbb{O}_{\mathsf{block}} \subseteq \mathbb{O}_e$.
  3. Check that $\mathsf{Verify}(\mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma) = 1$ as defined in Definition 1.
  4. Verify the Schnorr proofs $\pi_{\mathsf{s1}}, \pi_{\mathsf{s2}}, \pi_{\mathsf{s3}}$.
  5. Verify π_{range} using the range proof verifier.
  6. If all checks pass, accept the block and add it to the local chain: $\mathcal{C}_{\mathcal{P}} = \mathcal{C}_{\mathcal{P}} \parallel \mathcal{B}'$.

Figure 3: Protocol Zenith (block verification)

Protocol Zenith (Chain Selection)

Let: 1. k denote the ledger's safety parameternumber of slots in each epoch; 2. s denote the rule parameter of Ouroboros Genesis [BGK+18]; 3. $\mathcal{C}_{\mathsf{loc}}$ denote the party's local chain.

- **Fork choice:** Upon receiving a chain $\mathcal{C} \neq \mathcal{C}_{\mathsf{loc}}$, $\mathcal{P}$ does the following:
  1. If $\mathcal{C}$ forks from $\mathcal{C}_{\mathsf{loc}}$ at most k blocks, then if $|\mathcal{C}| > |\mathcal{C}_{\mathsf{loc}}|$ then $\mathcal{P}$ sets $\mathcal{C}_{\mathsf{loc}} \leftarrow \mathcal{C}$.
  2. Otherwise, let j be the maximum slot for which $\mathcal{C}, \mathcal{C}_{\mathsf{loc}}$ have the same slot (i.e., j is the slot on which the two chain fork). If $|\mathcal{C}[0 : j + s]| > |\mathcal{C}_{\mathsf{loc}}[0 : j + s]$ then $\mathcal{P}$ sets $\mathcal{C}_{\mathsf{loc}} \leftarrow \mathcal{C}$.

Figure 4: Protocol Zenith (chain selection)

used in the ring signature

- $\sigma_{\mathsf{ring}} = (c_0, \{r_i^g, r_i^x\}_{i=0}^{n-1}, \mathcal{I}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, A^{(P)}, C)$: ring signature
- $\pi_{\mathsf{s1}} = (C', \langle c, y_0, y_1, z_0, z_1 \rangle)$: first Schnorr proof, on "mirror" commitments C, C'
- $\pi_{\mathsf{s2}} = (E, \langle c_e, y_e \rangle)$: second Schnorr proof, on coefficient of F
- $\pi_{\mathsf{s3}} = (B, \langle c_b, s_b, s'_b \rangle)$: third Schnorr proof, on shared commitment value between B, E
- π_{range}: range proof

The final block is the concatenation of all of the above:

$$\mathcal{B} = \langle \mathcal{M}, \mathbb{O}_{\mathsf{block}}, \sigma_{\mathsf{ring}}, \pi_{\mathsf{s1}}, \pi_{\mathsf{s2}}, \pi_{\mathsf{s3}}, \pi_{\mathsf{range}} \rangle$$

Proof size Given a ring of size n, the block's proof consists of:

- $10 + 4 \cdot n$ elliptic curve points
- $11 + 2 \cdot n$ scalars
- one range proof

4 Private Chain Attack: Simulation Analysis

This section presents a simulation-based security analysis of Zenith against a *private chain attack*, a fundamental threat model for longest-chain Proof-of-Stake protocols.

4.1 Attack Description

The private chain attack is a fundamental attack against longest-chain consensus protocols. In this attack, an adversary $\mathcal{A}$ controlling a fraction of the total stake attempts to build a private chain that is at least as long as the honest chain, thereby being able to replace it under the longest-chain selection rule and reversing transactions that were previously considered confirmed.

Specifically, the attack proceeds as follows:

1. The adversary $\mathcal{A}$ controls a fraction $\beta \in (0, 1)$ of the total stake Σ. The remaining fraction $(1 - \beta)$ is controlled by honest parties.
2. Starting from a common prefix, $\mathcal{A}$ builds a private chain in isolation, without broadcasting any blocks to the honest network.
3. Honest parties follow the protocol faithfully: they build on the longest chain they have seen, broadcasting each new block immediately.
4. A transaction is considered confirmed once it is buried under k blocks on the honest chain. The adversary succeeds if, by the time the honest chain has grown by k blocks, its private chain has also reached length $\geq k$. In this case, the adversary can release its chain, which is at least as long as the honest one, causing the network to adopt it and thereby reverting the confirmed transaction.

The question we investigate is: for a given confirmation depth k, what is the maximum adversary stake fraction β^* such that the attack succeeds with significant probability ($\geq 0.1\%$)?

4.2 Simulation Model

4.2.1 Per-Slot Block Production

In Zenith, on each slot every staking output independently evaluates N *stake kernels* (cf. Section 3.2). An output with value a and difficulty D passes the eligibility test for a given kernel with probability proportional to a/D. The total stake is distributed across multiple parties, each potentially controlling multiple outputs.

In the simulation, we model this as follows. Let h denote the number of honest parties and a the number of adversary parties. Each honest party holds an equal share of the honest stake, $\alpha_h = \Sigma \cdot (1-\beta)/h$, and each adversary party holds an equal share of the adversary stake, $\alpha_a = \Sigma \cdot \beta/a$. The per-kernel eligibility probability for an honest party is $p_h = \alpha_h/D$, and similarly $p_a = \alpha_a/D$ for an adversary party.

Rather than simulating each party and each kernel individually, we compute the probability that *at least one* block is produced per slot on each side. For the honest chain, the number of independent Bernoulli trials per slot is $n_h = N \cdot h$, so the probability of at least one honest block in a slot is:

$$P_h = 1 - (1 - p_h)^{n_h} \tag{4}$$

An analogous expression holds for the adversary:

$$P_a = 1 - (1 - p_a)^{n_a}, \quad n_a = N \cdot a \tag{5}$$

This formulation captures the key property of the longest-chain rule: when multiple honest parties produce blocks in the same slot, only one extends the chain (the rest are orphaned), so what matters is the probability of *at least one* block, not the expected count.

4.2.2 Difficulty Calibration

The difficulty parameter D is calibrated so that the expected chain growth rate is exactly $g = 0.05$ (one block per 20 slots on average) when the adversary holds no stake ($\beta = 0$). This is achieved by solving:

$$P_h\big|_{\beta=0} = 1 - (1 - p_h)^{N \cdot h} = g$$

for D, which yields:

$$D = \frac{\Sigma}{h \cdot \left(1 - (1-g)^{1/(N \cdot h)}\right)} \tag{6}$$

Figure 5 shows the computed difficulty multiplier D/Σ as a function of N. The multiplier scales approximately linearly with N, reflecting the fact that more independent tests per slot require proportionally higher difficulty to maintain the target chain growth rate. Notably, the number of honest parties h has negligible effect on the difficulty multiplier, since the per-party probability adjusts inversely with h.

4.2.3 Simulation Procedure

For each parameter configuration (N, h, a, k, β), the simulation proceeds as follows:

1. Compute D via Equation 6.
2. Compute P_h and P_a via Equations 4 and 5.
3. Simulate slot by slot: on each slot, independently sample two Bernoulli random variables with parameters P_h and P_a, incrementing the respective chain length counters.

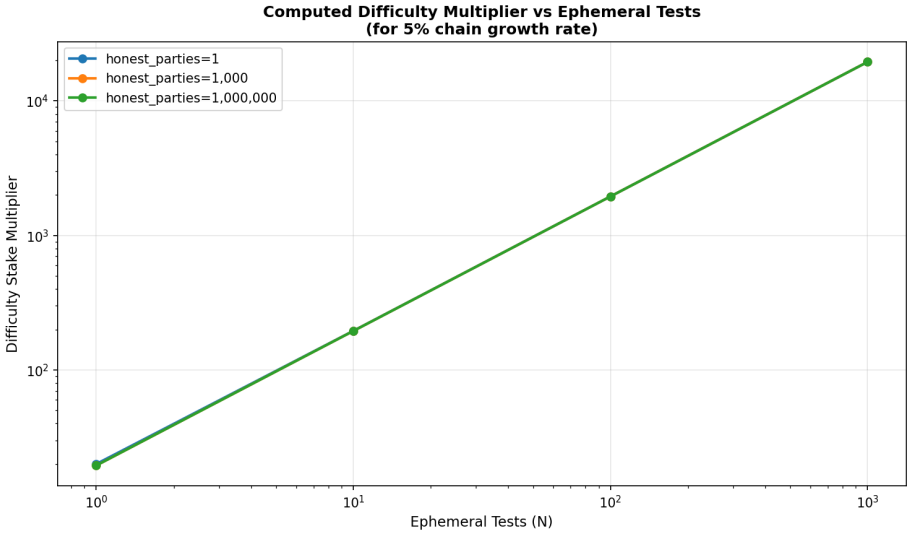

Figure 5: Computed difficulty multiplier D/Σ as a function of the number of ephemeral tests N, for various numbers of honest parties h. The multiplier scales linearly with N and is invariant to h.

4. Stop as soon as the honest chain reaches k blocks.

5. The adversary succeeds if, at that point, a_chain $\geq k$.

Note that the number of slots elapsed is not fixed but varies across trials: it is the (random) time it takes for the honest chain to accumulate k blocks. Given the $g = 0.05$ chain growth rate, the expected number of slots is $k/g = 20k$.

For each configuration, the adversary power β is decremented in steps of 0.01 percentage points from 49% downward. The simulation runs 1,000 independent trials at each β value. The *adversary power threshold* β^* is the smallest β for which the attack succeeds in at least one of the 1,000 trials.

4.2.4 Parameter Space

The simulation covers the following parameter ranges:

- **Ephemeral tests:** $N \in \{1, 10, 100, 1{,}000\}$
- **Honest parties:** $h \in \{1,\ 1{,}000,\ 1{,}000{,}000\}$
- **Adversary parties:** $a = 1$
- **Confirmation depth:** $k \in \{5, 10, 15, \ldots, 50, 100, 150, \ldots, 500, 1{,}000, 1{,}500, 2{,}000, 2{,}500\}$ (23 values)
- **Total stake:** $\Sigma = 10^7$
- **Tests per configuration:** 1,000

This yields a total of 276 data points across 12 distinct configurations.

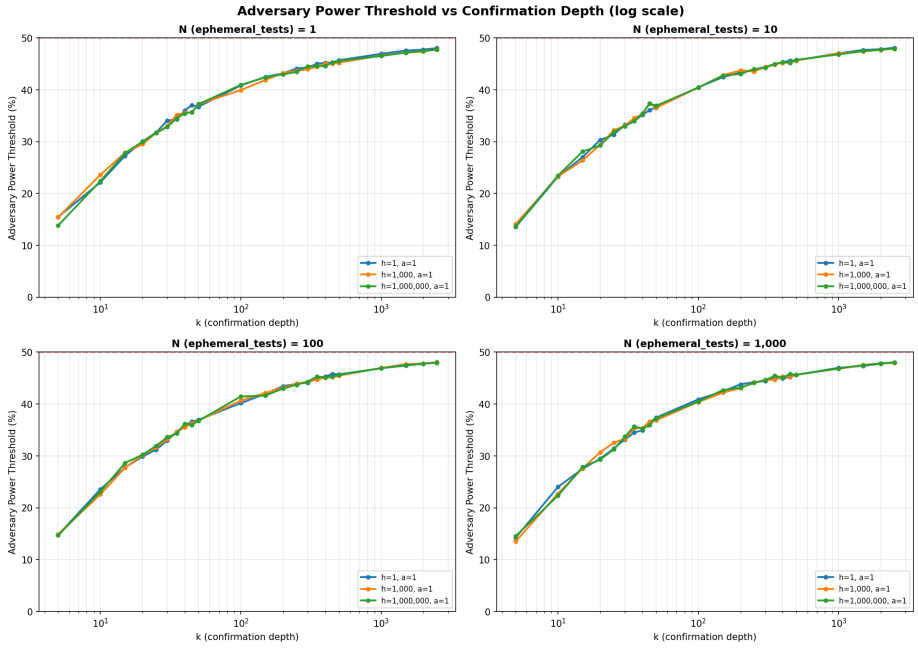

Figure 6: Adversary power threshold β^* as a function of confirmation depth k (log scale), for $N \in \{1, 10, 100, 1{,}000\}$. Each subplot shows three honest-party configurations ($h = 1$, 1,000, 1,000,000). The red dashed line marks the 50% upper bound.

4.3 Results

4.3.1 Adversary Threshold vs. Confirmation Depth

Figure 6 plots the adversary power threshold β^* against the confirmation depth k for all configurations, grouped by the number of ephemeral tests N.

Figure 7 provides a combined view overlaying all 12 configurations on a single plot, confirming that the curves collapse onto a single trajectory.

Table 1 summarizes the average adversary power threshold across all configurations for representative values of k.

4.3.2 Key Findings

Finding 1: The threshold depends only on k. The most prominent observation is that, for any fixed confirmation depth k, the adversary power threshold β^* is virtually identical across all parameter configurations. The spread across the 12 configurations (varying N and h over four and three orders of magnitude, respectively) is consistently below 2.5 percentage points, well within the noise margin of 1,000-trial Monte Carlo estimation.

The invariance on N is expected, since the difficulty calibration (Equation 6) normalizes away the effect of N, ensuring that the per-slot block production probabilities on each side depend only on the stake fractions β and $(1 - \beta)$.

The following findings should be read in conjunction with the parameter recommendations of Section 4.5, which identify the confirmation depths appropriate

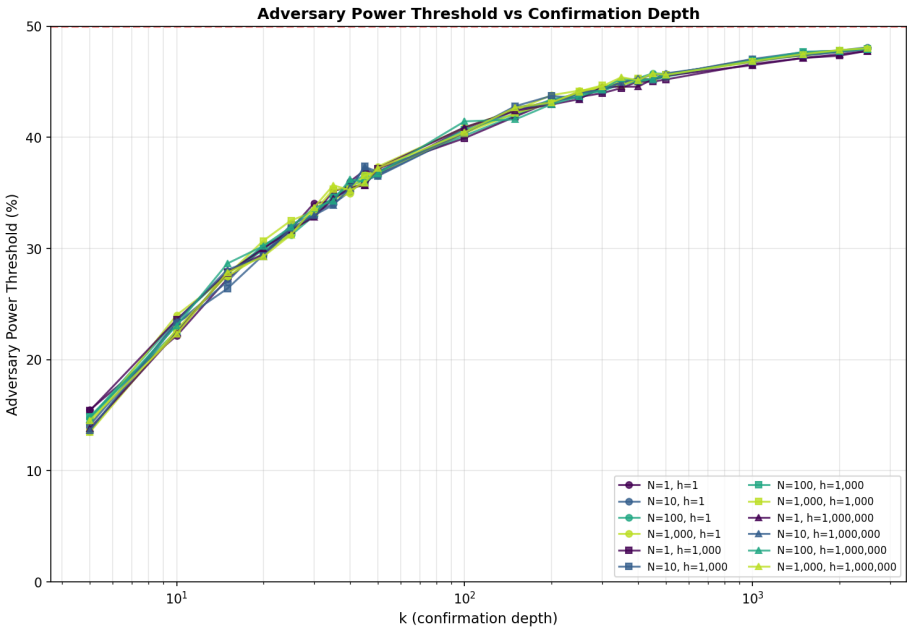

Figure 7: Combined overview of all 12 parameter configurations. All curves follow the same trajectory, confirming that the adversary threshold depends only on k.

k	β^* (avg, %)	Range (%)	Spread (%)
5	14.37	13.47 – 15.46	1.99
10	23.01	22.14 – 24.00	1.86
25	31.70	31.20 – 32.53	1.33
50	36.92	36.52 – 37.37	0.85
100	40.57	39.93 – 41.44	1.51
250	43.87	43.44 – 44.21	0.77
500	45.59	45.21 – 45.76	0.55
1000	46.86	46.50 – 47.05	0.55
2500	47.95	47.76 – 48.10	0.34

Table 1: Average adversary power threshold β^* across all 12 configurations, for selected confirmation depths k. The "Spread" column shows the difference between the maximum and minimum threshold observed across configurations.

for production use.

Finding 2: The threshold converges to 50% as k grows. The threshold β^* increases monotonically with k, approaching 50% from below but never reaching it. At small confirmation depths, the threshold is noticeably below 50%: for $k = 5$, the average threshold is approximately 14.37%, rising to 23.01% at $k = 10$ and 36.92% at $k = 50$.

This convergence pattern can be explained via a biased random walk model. Consider the *gap* $G_t = H_t - A_t$, where H_t and A_t denote the honest and adversary chain lengths after slot t, respectively. On each slot, the gap changes by one of four increments: $+1$ (only honest produces a block), -1 (only adversary produces), 0 (both produce), or 0 (neither produces). Since $P_h > P_a$ whenever $\beta < 50\%$, the gap has a positive *drift*: its expected per-slot increment is

$$\delta = \mathbb{E}[G_{t+1} - G_t] = P_h - P_a > 0. \tag{7}$$

After T slots, the expected gap is $\mathbb{E}[G_T] = \delta \cdot T$ and, by the central limit theorem, the standard deviation of G_T is $O(\sqrt{T})$.

The simulation runs until the honest chain reaches k blocks, which takes approximately $T \approx k/P_h$ slots. At this stopping time, the expected gap is proportional to k:

$$\mathbb{E}[G_T] \approx \frac{\delta}{P_h} \cdot k,$$

while the fluctuations around this expected value scale as $O(\sqrt{k})$. The adversary wins only if $G_T \leq 0$, i.e., its chain is at least as long as the honest chain. For any fixed drift $\delta > 0$, the probability of this event decays as k grows, because the expected gap (linear in k) eventually dominates the fluctuations (sublinear in k).

4.4 Sensitivity to Chain Growth Rate

The results presented above fix the chain growth rate (active slot coefficient) at $g = 5\%$. A natural question is whether the adversary threshold β^* depends on this parameter. To investigate, we ran additional simulations varying g over a wide range: $g \in \{1\%, 5\%, 10\%, 20\%, 50\%\}$, while fixing $N = 1$, $h = 1$, and $a = 1$.

Figure 8 plots the adversary power threshold against k for each chain growth rate. Table 2 summarises the results for selected values of k.

k	g=1%	g=5%	g=10%	g=20%	g=50%	Spread
5	14.22	14.45	14.99	14.94	18.17	3.95
10	22.60	22.92	23.63	24.56	25.68	3.08
50	36.49	37.28	36.96	37.77	38.53	2.04
100	40.35	41.05	40.67	41.13	41.68	1.33
500	45.76	45.69	46.11	46.14	46.36	0.67
2500	48.00	48.03	48.16	47.97	48.21	0.24

Table 2: Adversary power threshold β^* (%) for selected confirmation depths k across different chain growth rates $g = g$. The "Spread" column shows the difference between maximum and minimum threshold across growth rates.

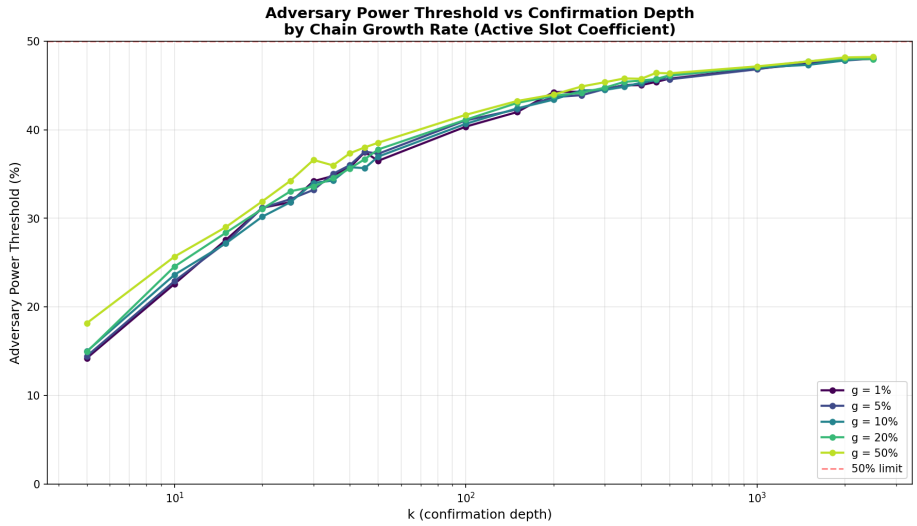

Figure 8: Adversary power threshold β^* as a function of confirmation depth k for different chain growth rates $g \in \{1\%, 5\%, 10\%, 20\%, 50\%\}$. The curves largely overlap, with modest divergence at small k.

Observation: The threshold is largely insensitive to growth rate. The key finding is that the adversary threshold depends primarily on the confirmation depth k, not on the chain growth rate g. The curves for different growth rates largely overlap, with all configurations converging toward 50% as k increases.

However, a modest effect is visible at small k: higher growth rates yield slightly higher thresholds. At $k = 5$, the threshold ranges from 14.22% (at $g = 1\%$) to 18.17% (at $g = 50\%$), a spread of approximately 4 percentage points. This spread narrows as k grows, falling below 1 percentage point for $k \geq 500$.

4.5 Effect of Network Delay

The simulations presented so far assume perfect synchrony: all honest parties observe the same chain tip and never produce competing blocks, unless these are created in the same slot. In practice, network delays cause honest parties to occasionally build on stale tips, creating temporary forks. When two honest blocks are produced within a short window, one is orphaned—representing wasted honest stake that the adversary can exploit.

To quantify this effect, we extended the simulation to model network delay. Let Δ denote the *delay parameter*, measured in slots: an honest block produced in slot t is not visible to other honest parties until slot $t + \Delta$. During this window, other honest parties may produce competing blocks on the same parent, leading to orphans. The adversary, by contrast, observes all blocks immediately (a standard worst-case assumption) and can coordinate its private chain without internal forks.

We ran simulations for $\Delta \in \{1, 5, 10\}$ across all five chain growth rates $g \in \{1\%, 5\%, 10\%, 20\%, 50\%\}$. Figure 9 presents the results.

Table 3 summarises the adversary threshold at $k = 2500$ for each combination of growth rate and delay.

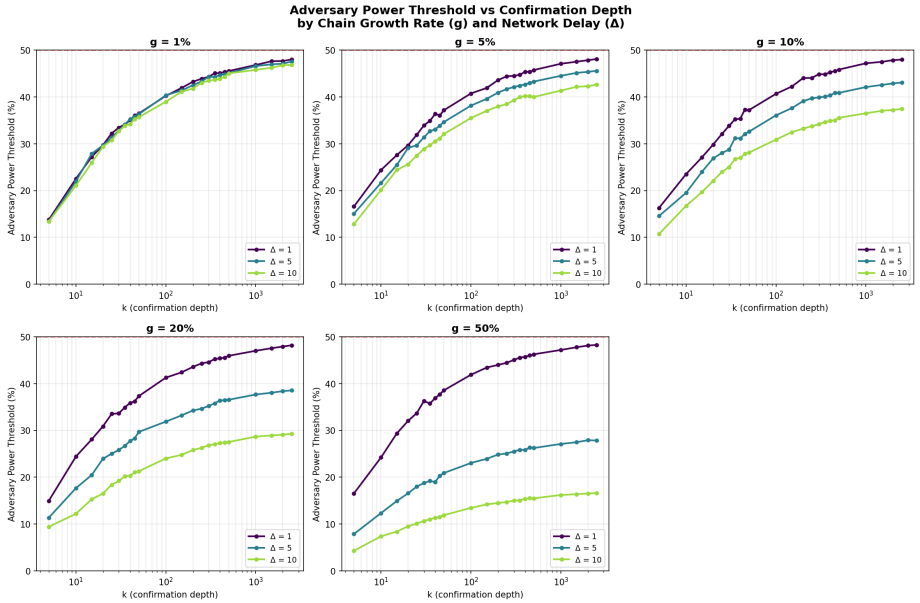

Figure 9: Adversary power threshold β^* as a function of confirmation depth k for different chain growth rates (subplots) and network delays $\Delta \in \{1, 5, 10\}$ slots. Higher delay and higher growth rate both reduce the threshold, with a strong interaction between the two.

Finding: Security is sensitive to the interaction between network delay and growth rate. At low growth rates ($g = 1\%$), the threshold remains close to 48% even with $\Delta = 10$, because blocks are rare enough that honest parties almost never collide. At high growth rates ($g = 50\%$), however, with $\Delta = 10$, an adversary controlling only 16.6% of the stake can succeed at $k = 2500$. This scenario assumes a chain growth rate far beyond the recommended deployment parameters and is presented to illustrate the boundary conditions of the security model.

The mechanism is as follows. When g is high, many slots produce honest blocks. With network delay Δ, honest blocks produced within Δ slots of each other may compete for the same parent, causing orphans. The orphan rate scales roughly as $g \cdot \Delta$: the probability of a collision in any given slot is approximately the probability that another honest block was produced in the preceding Δ slots, which is $\approx 1 - (1 - g)^{\Delta} \approx g \cdot \Delta$ for small $g \cdot \Delta$.

These orphaned blocks represent honest stake that does not contribute to chain growth, effectively reducing the honest chain's growth rate while leaving the adversary's private chain unaffected.

Implications for parameter selection. These results highlight a critical trade-off in protocol design. A high chain growth rate g improves transaction throughput and reduces confirmation latency in wall-clock time, but it amplifies the adversary's advantage under realistic network conditions. Conversely, a low growth rate is robust to network delay but increases the time users must wait for confirmations.

g	$\Delta = 1$	$\Delta = 5$	$\Delta = 10$	Spread
1%	48.0	47.6	46.8	1.2
5%	48.1	45.6	42.6	5.4
10%	48.0	43.1	37.4	10.5
20%	48.2	38.5	29.3	18.9
50%	48.3	27.8	16.6	31.7

Table 3: Adversary power threshold β^* (%) at $k = 2500$ for different chain growth rates g and network delays Δ. The "Spread" column shows the difference between $\Delta = 1$ and $\Delta = 10$.

For Zenith, the choice of g should account for the expected network delay in the deployment environment. If $\Delta \approx 5$ slots is realistic, then $g = 5\%$ maintains a threshold above 45%, while $g = 20\%$ drops the threshold to around 38%.

4.6 Cross-Epoch Difficulty Manipulation Attack

The previous analyses assume a fixed difficulty parameter D throughout the attack. In practice, Zenith employs dynamic difficulty adjustment to maintain the target chain growth rate despite fluctuations in participating stake. This section investigates whether an adversary can exploit the difficulty adjustment mechanism to gain an advantage in the private chain attack.

4.6.1 Difficulty Adjustment in Zenith

Because transactions in Zenith are private, the protocol cannot directly observe the total stake Σ participating in consensus. Instead, it infers the active stake from the observed chain growth rate. At the end of each epoch, the protocol measures the number of blocks produced and compares it to the target rate g. If fewer blocks were produced than expected, the protocol assumes that the total participating stake has decreased and lowers the difficulty accordingly.

Concretely, let g_{obs} denote the observed chain growth rate in epoch i. The protocol recalibrates difficulty for epoch $i{+}1$ by assuming that the stake producing g_{obs} should instead produce at the target rate g. This yields:

$$D_{i+1} = D_i \cdot \frac{g_{\text{obs}}}{g} \tag{8}$$

4.6.2 The Two-Epoch Attack

An adversary controlling a fraction β of the total stake can exploit this mechanism as follows:

Epoch i (withholding phase). The adversary refrains from producing any blocks, effectively removing $\beta \cdot \Sigma$ from the active stake. The honest chain, now powered by only $(1 - \beta) \cdot \Sigma$, grows at a reduced rate:

$$g_{\text{obs}} = (1 - \beta) \cdot g$$

The protocol interprets this slowdown as a reduction in total stake and lowers difficulty for the next epoch.

Epoch $i+1$ (attack phase). With the reduced difficulty D_{i+1}, the adversary rejoins and launches a private chain attack. When the full stake Σ participates under the lowered difficulty, the *effective* chain growth rate becomes:

$$g_{\text{eff}} = \frac{g}{1-\beta} \tag{9}$$

For example, with $\beta = 30\%$ and $g = 5\%$, the effective growth rate in the attack epoch is $g_{\text{eff}} = 0.05/0.70 \approx 7.14\%$.

4.6.3 Impact on Attack Success

We simulated the two-epoch attack across a range of adversary power levels $\beta \in \{5\%, 10\%, \ldots, 45\%\}$ and network delays $\Delta \in \{1, 5, 10\}$. For each configuration, we measured the maximum confirmation depth k that the adversary can successfully attack, comparing the baseline scenario (no manipulation, $g = 5\%$) against the manipulated scenario ($g_{\text{eff}} = 5\%/(1-\beta)$).

Figure 10 presents the results and Table 4 summarizes the findings for selected adversary power levels. Before interpreting these results, it is worth noting that the scenarios in Table 4 represent a stress-tested worst case: they require the simultaneous combination of near-40% adversarial stake, a network propagation delay of 10 slots, and active difficulty manipulation sustained over a full epoch. In practice, well-configured deployments operating within the parameter recommendations of Section 4.5 would not typically be exposed to this combination of conditions. The table is included for completeness and to establish conservative security bounds.

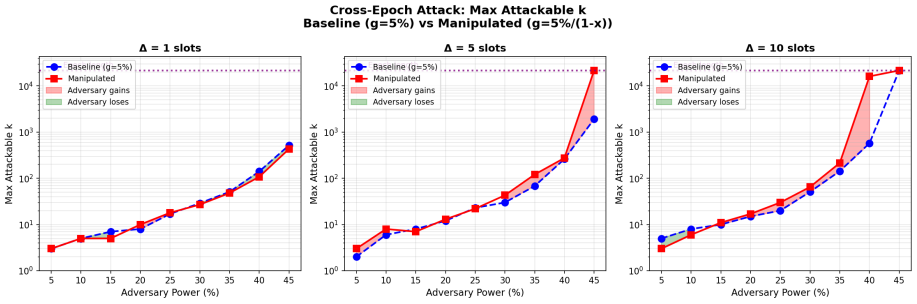

Figure 10: Maximum attackable confirmation depth k as a function of adversary power, comparing the baseline (dashed blue, $g = 5\%$) against the manipulated growth rate (solid red, $g = 5\%/(1-\beta)$). The purple dotted line indicates the epoch length (21,600 blocks).

Finding 1: Manipulation is ineffective at low network delay. At $\Delta = 1$ (near-synchronous network), the manipulation consistently *hurts* the adversary for $\beta \geq 30\%$. For instance, a 40% adversary can attack up to $k = 141$ without manipulation, but only $k = 108$ with manipulation.

Finding 2: Manipulation is effective at high network delay. At $\Delta = 10$, the attack becomes highly effective for strong adversaries. A 40% adversary

	$\Delta = 1$		$\Delta = 5$		$\Delta = 10$	
β	Base	Manip	Base	Manip	Base	Manip
20%	8	10	12	13	15	17
30%	29	27	30	43	51	65
40%	141	108	261	275	574	16,092
45%	518	431	1,915	21,600	21,600	21,600

Table 4: Maximum attackable k for selected adversary power levels, with and without difficulty manipulation. "Base" uses $g = 5\%$; "Manip" uses $g = 5\%/(1-\beta)$. Values of 21,600 indicate the adversary can attack the entire epoch.

improves from $k = 574$ to $k = 16{,}092$. At 45% adversarial stake, the attack already succeeds for the entire epoch even without manipulation.

Finding 3: The attack exhibits a threshold behavior. The benefit of manipulation depends on the combination of β, Δ, and the resulting g_{eff}. When $g_{\text{eff}} \cdot \Delta$ is small, manipulation is ineffective. When $g_{\text{eff}} \cdot \Delta$ exceeds a critical threshold, the attack is more successful.

4.6.4 Implications

The cross-epoch attack represents a scenario that should be mitigated in networks with non-negligible propagation delays, and the following measures are recommended as standard protocol hardening. An adversary can sacrifice one epoch's block rewards to artificially inflate the growth rate in the subsequent epoch, potentially attacking much deeper confirmation depths than would otherwise be possible.

This motivates potential mitigations:

- **Bounded difficulty adjustment:** Limiting the per-epoch difficulty change (e.g., to $\pm 10\%$) would cap the attainable g_{eff} and reduce the attack's potency.
- **Multi-epoch smoothing:** Averaging observed growth rates over multiple epochs would make it harder for an adversary to induce large difficulty swings through short-term withholding.

4.7 Scope and Limitations

The simulation is designed to provide conservative, worst-case bounds on protocol security for a well-defined attack scenario. The assumptions below are standard in the blockchain security literature and are chosen to favor the adversary with respect to network and timing conditions. However, the simulated adversary follows a fixed strategy, and the thresholds reported here should be understood as a baseline against this class of adversary. Deployments requiring stronger guarantees against more sophisticated strategies, such as adaptive chain release, grinding attacks, or multi-output exploitation, should pair these results with the mitigations outlined in Section 4.6.4.

Simplified adversarial model. The adversary in this simulation follows a simple strategy: it independently accumulates blocks on a private chain at its expected rate and hopes to match the honest chain's length of k blocks before the honest chain gets there. This does not model more sophisticated strategies such as:

- **Adaptive chain selection:** An adversary could strategically choose when to release its private chain, for instance only when it has accumulated a temporary lead, then reverting to the honest chain otherwise.
- **Grinding attacks:** An adversary might attempt to influence the epoch nonce or manipulate stake kernel inputs to increase its election probability.

Independent slot assumption. The simulation treats each slot as an independent Bernoulli trial. In the actual protocol, there may be correlations across slots due to shared randomness (the epoch nonce), adaptive difficulty adjustments, or the adversary's ability to selectively withhold blocks. These correlations could affect the convergence rate of the threshold.

Single-adversary model. The simulation fixes $a = 1$ adversary party. This does not capture scenarios where the adversary controls many small outputs and could exploit this for grinding or other combinatorial advantages.

Finite-sample effects. The threshold is determined by whether the attack succeeds in *at least one* out of 1,000 trials. This means the effective error probability bound is $\varepsilon \leq 1/1{,}000$, which is relatively coarse. A more fine-grained analysis with more trials would yield tighter bounds on the threshold, particularly for large k where the threshold is close to 50% and small probability differences matter.

5 Analysis: Potential Manipulation Attack on Inequality

Assume that $h_i, f_{\mathsf{utxo}}, q_{\mathsf{utxo}}, f'$ are sampled uniformly at random from $1, \ldots, 2^l$. Hence, we can re-write Inequality 3 as

$$d \cdot a_{\mathsf{utxo}} - r \mod l < 2^{64},$$

where r is a uniformly random element of $\mathbb{Z}_l$.

However, in this attack d is not computed honestly but chosen arbitrarily from the range $1 \leq d \leq \lfloor \frac{l}{D} \rfloor$ to ensure that the adversary wins. Hence, the adversary chooses d arbitrarily such that it satisfies the following inequalities:

$$d < \frac{r + 2^{64}}{a_{\mathsf{utxo}}} \mod l,\ d \leq \lfloor \frac{l}{D} \rfloor.$$

This is only possible if the random value r is such that

$$\frac{r + 2^{64}}{a_{\mathsf{utxo}}} \mod l < \lfloor \frac{l}{D} \rfloor,$$

which happens when

$$r < a_{\mathsf{utxo}} \cdot \lfloor \frac{l}{D} \rfloor - 2^{-64} \mod l$$

Let $Succ$ be a random variable taking value 1 when the attack succeeds (*i.e.*, it is possible to choose d in a way that the adversary wins) and 0 otherwise. Let R be a uniformly random variable taking values from $\mathbb{Z}_l$. We have that

$$Pr[Succ = 1] = Pr[R < a_{\mathsf{utxo}} \cdot \lfloor \frac{l}{D} \rfloor - 2^{-64} \mod l] = \frac{a_{\mathsf{utxo}} \cdot \lfloor \frac{l}{D} \rfloor - 2^{-64}}{l}$$

In other words, with probability $\frac{a_{\mathsf{utxo}} \cdot \lfloor \frac{l}{D} \rfloor - 2^{64}}{l}$ the adversary can choose d such that it is selected as leader.

We analyze the probability that this attack succeeds for the following parameters:

1. Total stake $\Sigma = 2^{64}$
2. Active slot coefficient $g = 0.05$
3. "Difficulty" parameter $D = \frac{1}{g} \cdot \Sigma = \frac{1}{0.05} \cdot 2^{64} < 2^{4.322} \cdot 2^{64} = 2^{68.322}$
4. Group order $l = 2^{256}$

In general we have

$$Pr[Succ] = \frac{a_{\mathsf{utxo}} \cdot \lfloor \frac{2^{256}}{2^{68.322}} \rfloor - 2^{64}}{2^{256}} = a_{\mathsf{utxo}} \cdot 2^{-68.322} - 2^{-192} \approx a_{\mathsf{utxo}} \cdot 2^{-68.322}.$$

An adversary controlling 40% of the stake has slightly more than $2^{62.678}$ tokens, so it has a probability of success for this attack of $Pr[Succ] > 2^{-5.644} > 0.019998$.

It is important to understand the advantage the adversary gets when mounting this attack versus when computing the value d as an honest party does, which is computing the following expression instead of minimizing d arbitrarily

$$d = \lfloor \frac{r}{a_{\mathsf{utxo}}} \rfloor + 1,$$

which when substituted in Inequality 3 yields

$$b_\alpha = \lfloor \frac{r}{a_{\mathsf{utxo}}} \rfloor \cdot a_{\mathsf{utxo}} + a_{\mathsf{utxo}} - r \mod l$$

which removing the floor function is approximately equal to the following expression as a_{utxo} cancels out

$$b_\alpha \approx r + a_{\mathsf{utxo}} - r \mod l = a_{\mathsf{utxo}}.$$

We d to satisfy both of the following conditions,

$$d = \lfloor \frac{r}{a_{\mathsf{utxo}}} \rfloor + 1, 1 \leq d \leq \lfloor \frac{l}{D} \rfloor,$$

which happens when

$$\lfloor \frac{r}{a_{\mathsf{utxo}}} \rfloor + 1 \leq \lfloor \frac{l}{D} \rfloor.$$

Assuming $a_{\mathsf{utxo}} | r$ and $D | l$ in order to remove the floor function, we have that r must be such that

$$\frac{r}{a_{\mathsf{utxo}}} \leq \left(\frac{l}{D} - 1 \right), \text{ or } \frac{r}{a_{\mathsf{utxo}}} < \frac{l}{D}, \text{ or } r < a_{\mathsf{utxo}} \cdot \frac{l}{D},$$

which happens with probability

$$Pr\left[R < a_{\mathsf{utxo}} \cdot \frac{l}{D} \right] = \frac{a_{\mathsf{utxo}} \cdot \frac{l}{D}}{l} = \frac{a_{\mathsf{utxo}}}{D} = \frac{a_{\mathsf{utxo}}}{g^{-1} \cdot \Sigma} = g \cdot \frac{a_{\mathsf{utxo}}}{\Sigma},$$

which means that for Zarcanum parameters $Pr\left[R < a_{\mathsf{utxo}} \cdot \frac{l}{D} \right] = 0.05 \cdot \frac{a_{\mathsf{utxo}}}{2^{64}} \approx a_{\mathsf{utxo}} \cdot 2^{-68.322}$. Hence, manipulating the choice of d adversarially *does not* give the adversary an advantage in being selected as leader.

6 Conclusion

This report has presented Zenith, a pure Proof-of-Stake consensus protocol for the Zano network, and quantified its security properties through simulation-based analysis.

The protocol demonstrates robust resistance to simulated private chain attacks across a wide parameter space. Security thresholds depend primarily on confirmation depth k and are largely invariant to details such as stake distribution and number of ephemeral tests. With a chain growth rate of $g = 5\%$ and network delays up to 5 slots, the protocol maintains adversary thresholds above 45% for confirmation depths $k \geq 500$.

Two items have been identified for the protocol's security hardening roadmap: (1) a formal security proof for the d/v-CLSAG instantiation, and (2) evaluation of aggregation-independent eligibility checks. While no concrete attacks have been identified in either case, addressing these through dedicated analysis would further strengthen the protocol's security foundation.

The parameter recommendations in Section 4.5 provide actionable guidance for deployment, and the cross-epoch attack analysis establishes bounds for difficulty adjustment mechanisms.

References

[BBB+17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066, 2017.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Cham, April / May 2018.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[NM+16] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *ledger*, 1:1–18, 2016.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[sk21] sowle and koe. Zarcanum: A proof-of-stake scheme for confidential transactions with hidden amounts. Cryptology ePrint Archive, Report 2021/1478, 2021.

[sow25] sowle. d/v-CLSAG: Extension for concise linkable spontaneous anonymous group signatures. Cryptology ePrint Archive, Paper 2025/2335, 2025.

[VS13] Nicolas Van Saberhagen. Cryptonote v 2.0. 2013.