Trustless Bridges via Random Sampling Light Clients

Bhargav Nagaraja Bhatt, Fatemeh Shirazi and Alistair Stewart

Web3 Foundation, Switzerland
{bhargav,fateme,alistair}@web3.foundation

Abstract. The increasing number of blockchain projects introduced annually has led to a pressing need for secure and efficient interoperability solutions. Currently, the lack of such solutions forces end-users to rely on centralized intermediaries, contradicting the core principle of decentralization and trust minimization in blockchain technology. In this paper, we propose a decentralized and efficient interoperability solution (aka *Bridge Protocol*) that operates without additional trust assumptions, relying solely on the Byzantine Fault Tolerance (BFT) of the two chains being connected. In particular, *relayers* (actors that exchange messages between networks) are permissionless and decentralized, hence eliminating any single point of failure. We introduce *Random Sampling*, a novel technique for on-chain light clients to efficiently follow the history of PoS blockchains by reducing the signature verifications required. Here, the randomness is drawn on-chain, for example, using Ethereum's RANDAO. We analyze the security of the bridge from a crypto- economic perspective and provide a framework to derive the security parameters. This includes handling subtle concurrency issues and randomness bias in strawman designs. While the protocol is applicable to various PoS chains, we demonstrate its feasibility by instantiating a bridge between Polkadot and Ethereum (currently deployed), and discuss some practical security challenges. We also evaluate the efficiency (gas costs) of an on-chain light-client verifier implemented as a smart contract on ethereum against SNARK-based approaches. Even for large validator set sizes (up to 10^6), the signature verification gas costs of our light-client verifier are a magnitude lower.

Keywords: PoS Blockchains · Trustless Bridges · Light Clients · Decentralised Relayers · RANDAO Bias · Adaptive Security

1 Introduction

Blockchains are designed as islands, it is easy to verify that a transaction is valid within the originating blockchain when one is following its history, but challenging otherwise. While interoperability may be easily resolved with trusted centralized intermediaries, it is not a desirable solution. Recent history has shown how risky this can be - centralised entities can be compromised or even act maliciously. According to a Chainalysis report [Cha22], failures in centralized bridges account for over 60% of all crypto hacks, resulting in losses exceeding $2B to date. In fact, four out of the top five incidents on the *rekt leaderboard* [rek24] are bridge-related hacks. The security of public blockchains hinges on decentralization, therefore, the mantra is to avoid relying on trusted intermediaries.

Bridges are by far the most attacked components in the blockchain space [ell22b, ell22a], because secure and efficient bridges are difficult to design and resolving attacks requires cooperation between chains which is nearly impossible. Hence it is important that the bridge must not have weaker security than either of the source or target chain. We introduce the notion of *crypto-economically sound* bridges, where we carefully trade off

soundness for efficiency. Assuming the honesty assumption used in the consensus of both chains, in expectation an attack on our bridge would be as expensive as the minimum market cap of the chains. We present an interactive *Commit-Challenge-Response* protocol for interoperability between a source and target PoS blockchain. Our protocol has an on-chain light client of the source chain deployed on the target chain (e.g., as a smart contract). Computation on blockchain networks is expensive, particularly in the context of verifying all signatures from Proof-of-Stake (PoS) validators on the source chain, and hence not feasible. In the process, we also define the notion of succinct finality, designed specifically to facilitate light clients following finality. Our approach improves the efficiency of following the history of a chain by randomly subsampling the signatures to be checked. Intuitively, the protocol works as follows: a set of relayers claim that 2/3 of validators signed a message (e.g., a block's finality). On-chain light client draws randomness from the target chain that determines a small subset of signatures to be checked out of those claimed. If the checks pass, we accept that the message was signed by at least one honest validator. Our solution has three highlights:

1. Relayers are permissionless and no additional honesty assumptions are required for bridge safety.

2. Drastically efficient in terms of on-chain computation costs for operating the bridge.

3. Scales well against large validator sets on the source chain.

We review the crypto-economic security of our solution and take into consideration attacks such as griefing and safety violations. This includes detecting subtle concurrency issues in the strawman designs, and fixes which rely on dynamically increasing the security parameter only in case of an attack. We then provide a framework to derive security parameters. Further, we apply the Fiat-Shamir heuristic to transform our interactive protocol into a non-interactive digital signature, and explore its impact on efficiency and interplay with crypto-economic arguments.

Finally, we instantiate our solution for a Polkadot-Ethereum bridge BEEFY, which has been implemented recently for public use [sno24]. We discuss practical challenges like implementing slashing and handling bias in RANDAO beacon. We perform a thorough security analysis of RANDAO biasability, which may be of independent interest to any protocol relying on RANDAO.

A prominent interoperability solution comparable to our work is Ethereum's Altair Light Client protocol [alt24] using sync-committees. In Altair, the source chain subsamples the validators and is fixed for the whole epoch, while our solutions lets the verifier on the target chain sample randomly. This key insight lets our solution improve over Altair along three dimensions:

- for the same soundness gurantees, our protocol is more efficient by a factor of 20 for large validator sets ($\approx 10^6$)

- handles adaptive corruption of validators. In Altair, an elected *sync-committee* remains stagnant for a whole epoch, exposing the committee against targeted attacks.

- our protocol exposes a security parameter that allows tuning the security and efficiency based on the application requirements.

Altair is only feasible for blockchains with extremely large validator sets. Our solution on the other hand is feasible and secure for any of PoS chain that allows slashing of malicious validator stake.

2 Preliminaries and System Model

Proof of Stake consensus mechanisms for blockchains require the nodes to *stake* the native cryptocurrency for a fixed period. In return, these nodes earn the opportunity to become *validators*, receiving rewards for producing blocks and participating in consensus. The security of the network is derived by the fact that the stakes of misbehaving nodes can be slashed (forfeited).

Light-clients are blockchain nodes that run in resource constrained environments like browsers or mobile devices to follow a decentralized consensus protocol. They do not maintain the entire blockchain history but instead validate the most critical pieces of information, such as block headers, to verify the integrity of the blockchain state. In particular, light clients have direct applications in building trustless and decentralized bridges between blockchains. Our definition of a bridge is in the broadest sense and application-agnostic. In this work, we define **bridges** as protocols that let two chains communicate and follow the finality (thru block headers) of each other. Applications like asset-swaps can be built on top of this basic functionality.

2.1 System Model

We formalize our setting, where the objective of an on-chain light client $\mathcal{LC}$ on *destination* chain $\mathbb{D}$ is to follow the finality of a *source* PoS blockchain $\mathbb{S}$. For brevity, we model the light client as a smart contract, however, our results hold for other computation models for updating state on $\mathbb{D}$. Our setting comprises of three actors: Validators on the source chain $\mathbb{S}$, trustless relayer $\mathbb{R}$, and a light-client $\mathcal{LC}$ deployed on $\mathbb{D}$. We make some standard assumptions on the **PoS model** of $\mathbb{S}$, satisfied by most PoS networks like Cosmos, Ethereum, Polkadot:

1. **Consensus:** The blockchain $\mathbb{S}$ runs a consensus mechanism with deterministic finality, e.g., *Byzantine Fault Tolerant* algorithms like CasperFFG, GRANDPA, Tendermint, PBFT, etc. [BG17, SK20, BKM18, CL99].

2. **Payloads:** Each block B in $\mathbb{S}$ contains a payload $\mathcal{P}_B$ capturing the state of $\mathbb{S}$ after executing block B. Typically, the payload is the hash of a crypto accumulator (e.g., Merkle Tree or Merkle Mountain Range Root [Tod16]) where its leaves are the state of $\mathbb{S}$.

3. **Justifications:** Oblivious of the underlying consensus mechanism, once a block B is finalised in $\mathbb{S}$, the payload $\mathcal{P}_B$ is signed by all honest validators in the associated validator set $\mathcal{V}_B$ of size n. We assume at most f validators are malicious, such that $3f + 1 < n$. The validator use an unforgeable signature scheme σ to attest the payloads. Let σ_i be the signature of V_i on $\mathcal{P}_B$, which can be verified against its public-key pk_i. We often identify the validator V_i by its public-key. A block B is considered *justified* (or finalised when context is clear) if its payload $\mathcal{P}_B$ has been signed by at least $^{2n}/_3 + 1$ validators in $\mathcal{V}_B$. The set of such valid signatures are called justifications, denoted $\mathcal{J}_B = \{\sigma_i | i \in [1, .., n]\}$. We say a justification is valid iff $|\mathcal{J}_B| \geq {}^{2n}/_3 + 1$ and σ_i's are valid signatures from validators in $\mathcal{V}_B$. Assuming the underlying consensus mechanism is safe, note that for a particular block-height h, only a unique block B can have valid justifications.

4. **Stakes:** Validator V_i has a stake s_i locked on $\mathbb{S}$. Any malicious behavior by V_i results in partial or full slash of validator's stake. We assume the delay in detecting an offense and enforcing the slash and is bounded by Δ, the *slashing delay*.

5. **Epochs:** The blockchain S is divided into consecutive set of blocks called *epochs*, denoted $\mathcal{E} = \{\mathcal{E}_0, \cdots, \mathcal{E}_i, \cdots\}$. For any i, the validator set $\mathcal{V}_{\mathcal{E}_i}$ remains unchanged for all blocks within an epoch $\mathcal{E}_i$.

6. **Epoch Transitions:** A commitment (e.g. Merkle Root) to the next validator set $\mathcal{V}_{\mathcal{E}_{i+1}}$ is included in the payload of $\mathcal{E}_i$'s last block. PoS networks provide this functionality to facilitate light-clients to sync-up efficiently to the head of the chain by tracking only blocks with validator set changes.

Note: *Justifications* are succinct proofs of finality designed to simplify the light-client's process for following the chain head. We require honest validators to justify a payload only if the payload has already been finalized by the underlying finality gadget. Unlike consensus mechanism where an honest validator may sign a non-finalized block, an honest validator never signs justification for a block that has not already been finalized. Hence, the justification of a block guarantees stronger properties than being finalized by the underlying finality gadget like Grandpa or Casper-FFG. We show it is straightforward to construct a justification layer on top of any finality gadget in Section 5.1.

Relayers (denoted $\mathbb{R}$) are entities that collect justifications $\mathcal{J}_B$ from $\mathbb{S}$ and interact with the light-client $\mathcal{LC}$ to convince it of the newly finalised blocks. In the process, they may collect fees for their efforts. Relayers are completely permissionless, i.e, they do not deposit any stakes either on $\mathbb{S}$ or $\mathbb{D}$ nor there is any registration. Relayers can communicate with $\mathbb{D}$ by calling transactions of the smart contract $\mathcal{LC}$. The only requirement on $\mathbb{D}$ is:

1. **Randomness:** $\mathbb{D}$ provides a source of randomness $\mathcal{R}$ accessible to $\mathcal{LC}$. Ideally, $\mathcal{R}$ is unbiasable by the relayers, and for that matter, even by $\mathbb{D}$'s validators. We model the source of randomness as a random function $\mathcal{R} : Blocks_{\mathbb{D}} \to \mathbb{N}$, where $\mathcal{R}(B^{\mathbb{D}}) \in \mathbb{N}$ is the randomness generated at block $B^{\mathbb{D}}$.

2.2 Attacker Model

We model the attackers as rational agents, ie., an attack is launched only if the expected outcome of an attack is positive. Our attacker model allows collusion between the relayers and validators on $\mathbb{S}$, as well as validators on $\mathbb{D}$. We tolerate adversarial faction up to the limit determined by the underlying consensus mechanism on both $\mathbb{S}$ and $\mathbb{D}$, without adding new trust assumption for safety of the bridge. In particular, relayers are completely permissionless and trustless. The relayer have no stake attached on either $\mathbb{S}$ nor $\mathbb{D}$. The relayer also can arbitrarily initiate and break an interactive session with the light client. This also means relayer can spawn multiple session concurrently. However, we assume the existence of at least one honest relayers for liveness of the bridge.

2.3 Problem Statement

We tackle the problem of building trustless bridges using light clients [XZC+22]. Typically, a light client synchronizing block headers of the source chain is implemented as a smart contract on the destination chain. This ensures the destination chain can verify information about the state of $\mathbb{S}$ without relying on external parties. Moreover, it allows anyone to prove the existence of transactions on $\mathbb{S}$ for smart contracts on $\mathbb{D}$ (using Merkle proofs), paving the way for generic applications. We formally lay out the desired properties of a light client protocol between verifier $\mathcal{LC}$ and prover $\mathbb{R}$, with an objective to update $\mathcal{LC}$'s view of the latest finalised block on $\mathbb{S}$. The $\mathbb{R}$ (prover) wants to convince the $\mathcal{LC}$ (verifier) that at least one honest validator on $\mathbb{S}$ signed the payload $\mathcal{P}_B$ of a recently finalised block B. We introduce the notion of *crypto-economic soundness* for light-client protocols which lets us trade-off negligible soundness guarantees for efficiency.

Definition 1. ϵ-Soundness Let Π be the protocol between $\mathbb{R}$ and $\mathcal{LC}$. Assume no honest validator in $\mathbb{S}$ signed $\mathcal{P}_B$. If the $\mathbb{R}$ (prover) can convince $\mathcal{LC}$ (verifier) of $\mathcal{P}_B$'s authenticity with probability at most ϵ, then Π is defined to be ϵ-sound. We term ϵ as the soundness error of Π.

Our attacker model does not make any honesty assumptions on relayers. Any safety violation in our approach is traced back to the validators signing malicious payloads. We require the light client protocol to be *accountable*, i.e, malicious validators on $\mathbb{S}$ can be identified and slashed. Our goal is to design a light client protocol that is *crypto-economically sound*, as defined below:

Definition 2. Crypto-economic Soundness Let Π be ϵ-Sound. If ϵ is such that the expected outcome of an adversary attacking Π is negative, then Π is Crypto-economically Sound.

3 Interactive Random Sampling Protocol

We describe an interactive light-client protocol Π_{Int} between a relayer $\mathbb{R}$ listening to finalised blocks on $\mathbb{S}$ and light-client $\mathcal{LC}$ deployed on $\mathbb{D}$. $\mathbb{R}$ wishes to convince $\mathcal{LC}$ of a new finalised block B on $\mathbb{S}$ which succeeds the latest finalised block known to $\mathcal{LC}$. Here, by *convince*, we mean that the light-client is ensured that at least one honest validator signed B.

Once a block B is finalised on $\mathbb{S}$, the relayers collects the justifications $\mathcal{J}_B$ and initiates Π_{Int}. In practice, the relayer can obtain the justifications by running a full-node of $\mathbb{S}$ and listening to it's consensus gossip-network. Naively, $\mathcal{LC}$ can check the finality of B by verifying $f+1$ signatures (where f denotes the malicious nodes on $\mathbb{S}$) in $\mathcal{J}_B$, ensuring at least one honest validator signed B. Combining the above observation with byzantine assumptions ($3f+1 < n$) on $\mathbb{S}$, the light-client is required to verify $n/3+1$ signatures in $\mathcal{J}_B$. Π_{Int} is a *Commit-Challenge-Response* [Cra96] which reduces the number of signature verifications (sub-linear w.r.t validator-set size) performed on the verifier's side. In particular, Π_{Int} can be viewed as an instantiation of Interactive Oracle Proofs (IOP) [BCS16] for the language of *digital signatures* with extensions to equip it with accountability. We synonymously use the term *Prover* for the relayer $\mathbb{R}$ and *Verifier* for the light-client $\mathcal{LC}$.

3.1 Description

The prover $\mathbb{R}$ initiates the protocol claiming to have valid justification $\mathcal{J}_B = \{\sigma_i | V_i \in \mathcal{V}_B\}$ for the finality of block B, where σ_i's are signatures of V_i on payload $\mathcal{P}_B$. Instead of sharing the whole justification (super-majority of signatures), the relayer shares its *Claims*, a bitfield of length $|V|$ (validator-set size) which represents the validator signatures claimed to be possessed by $\mathbb{R}$. For accountability reasons, the tuple in the *commit* phase also includes a validator signature σ_j along with the claims, which we term as the *backing validator* of the current claim. Note that *Backing Validators* is only an expository term we use to describe the protocol and prove its security. As such, backing validators are not special actors in the protocol. We assume the verifier knows the Merkle Root $\mathcal{C}$ of the set of validators (identified by their public keys) for the epoch in which B is finalised. [1] The verifier has access to public randomness $\mathcal{R} : Blocks_{\mathbb{D}} \to \mathbb{N}$ which it can query at a particular block on $\mathbb{D}$. In the *challenge* phase, it uses $\mathcal{R}$ to query a random subset of the signatures. The prover responds by sending the signatures and openings of the randomly sampled signatures in the *response* phase. The verifier maintains two variables in its state: $latestSyncedBlockHeight$ and $latestSyncedPayload$, denoting the block-height and payload of the latest know block finalised on $\mathbb{S}$. If the signatures and

[1] As outlined in section 2.1, the epoch transition mechanism on source chain comprises of a *hand-over* process, where the Merkle Root of the validator set responsible for next epoch is included in the state by the last block of the current epoch. Such hand-overs are mandatorily enforced as part of the protocol. In unforeseen circumstances when such *handovers* are unsuccessful, fall-backs mechanism can designed using on-chain governance. To bootstrap the protocol, either the genesis block or a trusted snapshot can be used.

Protocol 1 Π_{Int} : Interactive Random Sampling

1. **Commit:** Prover $\mathbb{R}$ sends a tuple $(\mathcal{P}_B, \mathit{Claims}, \sigma_j, op_j)$ to the verifier, where:
   - $\mathcal{P}_B$: Payload for the block B.
   - $\mathit{Claims} = [b_1, \cdots, b_n]$: a bitvector of length n. An honest prover sets $b_i = 1$ iff they possess the signature σ_i on $\mathcal{P}_B$ that verifies against pk_i.
   - σ_j: signature of the backing validator v_j.
   - op_j: opening of $\mathcal{C}$ to pk_j (i.e., the Merkle co-path).
2. **Challenge:** Verifier $\mathcal{LC}$ checks if *Claims* has at least $n - f$ indices set to 1s, else it terminates. If the signature σ_j is valid and op_j opens to $\mathcal{C}$ for pk_j, then the verifier samples m indices $i_1, \ldots i_m$ where each i_k is chosen uniformly at random (using $\mathcal{R}$) from the positions of bits in *Claims* set to 1s.
3. **Response:** Prover sends signatures σ_{i_k}, the public keys pk_{i_k} and openings op_{i_k} of $\mathcal{C}$ to pk_{i_k} for $k \in [1, m]$ to the verifier.
4. **Verify:** Verifier performs the following checks and terminates if any fail:
   - the openings $op_{i_1}, \ldots, op_{i_m}$ against the corresponding public keys $pk_{i_1}, \ldots, pk_{i_m}$ and $\mathcal{C}$ at the randomly chosen indices $\{i_1 \ldots i_k\}$.
   - signatures $\sigma_{i_1}, \ldots, \sigma_{i_m}$ against the public keys $pk_{i_1}, \ldots, pk_{i_m}$ and $\mathcal{P}_B$.
   - If $height(B) > latestSyncedBlockHeight$, $\mathcal{LC}$ makes the following state updates:
     - $latestSyncedBlockHeight := height(B)$
     - $latestSyncedPayload := \mathcal{P}_B$

     Else, B is stale and state remains unchanged.

Figure 1: Π_{Int}: Verifier randomly samples a subset of validator signatures to be verified.

opening submitted in the *response* phase verifies, the payload ($\mathcal{P}_B$) is accepted the state is accordingly updated.

We assume the signature scheme σ is unforgeable and the commitment scheme $\mathcal{C}$ is binding. We ignore the negligible probability that the prover can find a signature that verifies against the public key of an honest validator that did not sign the message or that they can find an opening of $\mathcal{C}$ at position i to a value other than pk_i that verifies. To ensure randomness is unpredictable to the prover at *commit* phase, verifier uses the randomness $\mathcal{R}(d)$ only revealed after the commit phase as concluded. Equipped with the above assumptions, we now present the soundness and completeness results for Π_{Int}.

Theorem 1. ***(ϵ-Soundness)*** *Consider prover $\mathbb{R}$ initiates Π_{Int} for block B. If no honest validator in $\mathbb{S}$ signed $\mathcal{P}_B$, then the verifier $\mathcal{SC}$ accepts $\mathcal{P}$ with probability at most 2^{-m}, where m is the security parameter, i.e., the number of signatures randomly sampled by the verifier.*

Proof. Let's assume no honest validator signed $\mathcal{P}_B$. The prover must provide a bitfield *Claims* with at least $n - f$ 1s. Consider k for some $1 \leq k \leq m$. The public key pk_{i_k} belongs to a dishonest validator only if i_k is one of at most f possible values out of $n - f$ 1s at the least. Since $n > 3f$ and $\mathcal{R}$ is unpredictable and uniformly random, with probability

at least $1 - f/(n-f) > 1 - f/2f = 1/2$, pk_{i_k} belongs to an honest validator. Since the i_ks are each chosen independently, the probability that no pk_{i_k} for any k belongs to an honest validator is at most 2^{-m}.
It remains to show that if some pk_{i_k} belongs to an honest validator, then the prover cannot convince the verifier. The prover cannot provide an opening of $\mathcal{C}$ at position i_k to a value other than pk_{i_k} because the commitment scheme is binding. The prover cannot provide a signature σ_{i_k} that verifies against pk_{i_k} because the signature scheme is unforgeable and honest validators did not sign $\mathcal{P}_B$. Hence the prover can not convince the verifier in this case. We note that if $m > f$, the the protocol is deterministically sound. If the verifier needs to be absolutely sure, the verifier can verify $m > f$ signatures, since at least one of those signatures will be of an honest validator. □

Theorem 2. ***(Completeness)*** Π_{Int} *is* complete. *If relayer (prover) posses valid justifications* $\mathcal{J}_B$ *of a valid block* B*, then the light clients (verifier) updates it's state of the latest synced block to* B*.*

Example 1. Suppose $\mathbb{S}$ has 100 validators of which at most 33 are byzantine. A block gets finalised with at least 67 signatures in its justifications. The relayer collects these justifications and starts Π_{Int} to convince the verifier that the block has been finalised. If the verifier samples $m = 34$ signatures in the *challenge* phase, it can be sure that at least one of these signatures is from an honest validator. If the verifier only samples $m = 10$ signatures in challenge phase and the *verify*-phase goes thru, then Theorem 1 guarantees that the probability of the light client accepting a malicious (invalid justifications) block is at most $1/2^{10}$.

3.2 Crypto-Economic Security

Π_{Int} guarantees probabilistic ϵ-Soundness with the soundness error $1/2^m$, solely dependent on m, the number of signatures sampled in the *challenge* phase. This leads to a natural question: how to set the security parameter m? We provide a crypto-economic framework for deriving the security parameter, striking a balance between efficiency and security.

Accountability and Slashing Exposure: We specifically require $\mathbb{R}$ to include at least one signature σ_j (i.e., backing validator V_j's signature) from their *Claims* in the *commit*-phase. If the payload is malicious, then the *backing validator* who signed the malicious payload can be identified and slashed on $\mathbb{S}$. In absence of backing validator signatures in the *commit* phase, the provers can initiate arbitrarily many instances of Π_{Int} sequentially or concurrently. The relayer would then have the choice to continue only in instances where the randomness drawn by the verifier in *challenge* phase is favorable. The validators have no economic disincentive for signing malicious payloads. Consequently, the relayers can increase the number of attempts (more turns at rolling the dice) in submitting malicious payloads. Therefore, any attempt by validators at signing malicious blocks (i.e., a block not finalised on $\mathbb{S}$) needs to be penalised/slashed.

We assume adversaries are rational, implying that an attack is initiated only if the expected economic value of the attack is positive. Our model considers the economic value of a successful attack to be the market-cap of $\mathbb{S}$, denoted M. If the attack is unsuccessful, we consider the lowest stake s of the validators on $\mathbb{S}$ to be the economic loss for the attackers. We assume the stake of all validators on $\mathbb{S}$ are identical, however, our framework can be extended to weighted PoS systems. The expectation of the economic value of an attack is computed as: $E(X) = (\epsilon \cdot M) - s$, where X is random variable for the economic value of the attack and ϵ is soundness error of Π_{Int}. Setting $E(X) < 0$, we derive $m \geq log_2(M/s)$.

3.3 Discussions on Safety and Liveness

If a relayer initiates the *Commit* phase for a canonical block but does not respond to the *Challenges* by the verifier (either due to genuine or malicious reasons), there is no necessity to slash the *backing validator* revealed in the commit phase. Slashing any validator that signs a non-canonical block (i.e. with no valid justifications) suffices for the ϵ-soundness of Π_{Int}. A soundness attack (i.e. relaying a malicious block) is public already at the *Commit* phase and eventually the *backing validator* who signed a malicious payload gets slashed, guaranteeing ϵ-soundness without any honesty assumptions on the relayers.

We assume there is one honest relayer who relays at least one block per epoch for liveness of the bridge, i.e., the verifier is in sync with the latest finalised blocks on the source chain. Presence of at least one honest relayer per epoch ensures that the verifier can track the validator set changes. A subtle corner case impacting soundness arises when there is no honest relayer for a time-period greater than the *unbonding period*, i.e, time allowed for an active validator to unbond their stake and exit the network (usually 20-30 days). If no blocks are relayed and updated by the light client over such a long period, then there is a possibility that Validators on source chain unbond their stake and hence there is no stake to slash for Π_{Int} attempts after the unbonding period. However, we feel this is unlikely for bridge applications with a clear monetary incentive to stay live. Moreover, the lengthy unbonding period provides ample time to detect and rectify the situation.

4 Dynamic Random Sampling

Π_{Int} is prone to a subtle concurrency attack that increases the soundness error (ϵ) exploiting the following observations:

- Relayers are trustless and permissionless (in-line with our design principle). Hence, an adversary can spawn arbitrarily many relayers that initiate Π_{Int}.
- It takes non-negligible time Δ for slashing a validator (on $\mathbb{S}$) after detecting its signature on a malicious payload. Moreover, it takes the duration of a full epoch ($|\mathcal{E}|$) for the light client to discover the change in validator set (i.e., malicious validator has been ejected).

Concretely, lets assume an adversary $\mathcal{A}$ controls f validators on $\mathbb{S}$, implying $\mathcal{A}$ can obtain $n/3$ validators signatures on any malicious payload. The adversary proceeds by initiating c concurrent instances $\mathcal{I}_1, \ldots, \mathcal{I}_c$ of Π_{Int} for a particular block B (with malicious payload signed by the f malicious validators) with the same backing signature σ but different claims bitvectors $\{b_1, \ldots, b_c\}$ in the *Commit* phase. Since there is a delay $\Delta + |\mathcal{E}|$ before the light-client is aware of the slashing and validator set change, the adversarial relayer can reuse the same backing validators signature without increasing slashing exposure of his instances. The adversary then proceed with the protocol only in those instances where the random sampling of challenge indices in the *Challenge* phase is favorable, i.e., the challenge indices correspond to malicious validators. Concretely, the success probability of the attack can now be quantified as $1-(1-\epsilon)^c$. However, the slashing exposure is at most s_j, as only σ_j, the signature of the malicious backing validator is exposed. Analogous to the expected obtained in Section 3.2, we can compute expected outcome under the above attack scenarios with c concurrent instance of Π_{Int} as:

$$E_c(X) = (1-(1-\epsilon)^c)M - s \tag{1}$$

The expected attack value increases with c, the number of concurrent relay instances spawned. For every ϵ, there exists a c such that $E_c(X) > 0$. This is a clear attack on the crypto-economic soundness of Π_{Int}.

4.1 Counter-measure: Dynamically Increasing Signature Checks

To counter the concurrency issue described above, we propose Π_{Dyn}, an extension of Π_{Int}. In Π_{Dyn}, the security parameter (signatures sampled) dynamically increases based on the number of relay instance backed by the same *backing* validator. The light client (verifier) now keeps a counter $u_{e,v}$ for each *backing* validator v in epoch e that increments by 1 whenever there is an initial claim made by a relayer. For any further relay instances by a relayer (or a set of relayer) using the same backing validator v, the number of signatures sampled during challenge phase is increased by $1 + 2 * \lceil log_2(u_{e,v}) \rceil$. If c instances of Π_{Dyn} are initiated concurrently using the same *backing* validator, the probability of attack succeeding is summation over all the c instances:

$$\sum_{i=1}^{c} \frac{1}{2^{m+1+2*\lceil log_2 i \rceil}} \leq \frac{1}{2^m} \tag{2}$$

This dynamic increase in the number of signature checks ensures that the probability of successful attack is bounded irrespective of the number of concurrent Π_{Dyn} initiated, and the advantage gained by using the same *backing* validator multiple times is neutralized.

The setup for Π_{Dyn} is very similar to Π_{Int} except that the verifier ($\mathcal{LC}$) maintains an additional mapping $u : (\texttt{Epoch}, PK_{\mathcal{V}_B}) \rightarrow \mathbb{N}$, where $u(e, v)$ captures the number of time a validator has been used for the backing signature in the commit-phase. Here, the *backing* validator is identified by its public-key. We do not keep track of relayers as they are permissionless and any attempt to keep track is not sybil resistant. Intuitively, the dynamic random sampling described in Protocol 2 ensures the adversary does not increase probability of successful attack without increasing its slashing exposure. Importantly, note that the security parameter remains unchanged if no concurrency attack is launched.

Theorem 3. ***Dynamic Random Sampling Mitigates Concurrency Attacks*** *Let $\Pi_{dyn,\epsilon}$ denote an instantiation of Π_{Dyn} (Protocol 2) where security parameter m is set s.t $\epsilon = 1/2^m < s/\mathcal{M}$. For any PPT adversary $\mathcal{A}$ controlling at most $n/3$ validator on $\mathbb{S}$, the expected incentive for an attacker $\mathcal{A}$ is negative.*

Proof. We first compute the probability of successful attack by defining the following events:

- $W_{\mathcal{A}}$: adversary convinces the verifier an un-finalised block B without *justifications*, i.e., the adversary successfully attacks Π_{Dyn}.
- $W_{\mathcal{A},v,k}$: the adversary succeeds and $u(e, v) = k$ in the verifiers state.
- $T_{\mathcal{A},v,k}$: the adversary attempts using v as the backing validator in *commit* phase for the k^{th} attempt, i.e., $u(e, v) = k$ after the attempt.

The success probability for an adversary is computed as follows:

$$\mathbb{P}(W_{\mathcal{A}}) = \sum_{v \in V} \sum_{k=1}^{\infty} \mathbb{P}(W_{\mathcal{A},v,k}) \tag{3}$$

$$= \sum_{v \in V} \sum_{k=1}^{\infty} \mathbb{P}(W_{\mathcal{A},v,k} | T_{\mathcal{A},v,k}) \cdot \mathbb{P}(T_{\mathcal{A},v,k}) \tag{4}$$

$$\leq \sum_{v \in V} \mathbb{P}(T_{\mathcal{A},v,1}) \cdot \sum_{k=1}^{\infty} \frac{\epsilon}{2^{1+2\lceil logk \rceil}} \tag{5}$$

$$\leq \frac{3\epsilon}{4} \cdot \sum_{v \in V} \mathbb{P}(T_{\mathcal{A},v,1}) \tag{6}$$

Protocol 2 Π_{Dyn}: Dynamic Random Sampling

1. **Commit:** The prover $\mathbb{R}$ sends a tuple $(\mathcal{P}_B, \mathit{Claims}, \sigma_j, op_j)$ to the verifier, where:
   - $\mathcal{P}_B$: Payload for the block B.
   - $\mathit{Claims} = [b_1, \cdots, b_n]$: a bitvector of length n. An honest prover sets $b_i = 1$ iff they possess the signature σ_i on $\mathcal{P}_B$ that verifies against pk_i.
   - σ_j: signature of the backing validator v_j.
   - op_j: opening of $\mathcal{C}$ to pk_j (i.e., the Merkle co-path).
2. **Challenge:** Verifier increments $u(e, pk_j) := u(e, pk_j) + 1$ corresponding to the current epoch e and validator with public-key pk_j. Verifier checks if *Claims* has at least $n - f$ indices set to 1s, else it terminates. If the signature op_j opens to C and σ_j is valid, then the verifier samples $m' = m + 1 + 2 * \lceil log_2(u_{e,v}) \rceil$ indices $i_1, \ldots i_{m'}$ where each i_k is chosen uniformly at random (using random function $\mathcal{R}$) from the positions of bits in *Claims* set to 1s. Here m is a statically chosen security parameter.
3. **Response:** Identical to Π_{Int}, the prover sends signatures, public keys and openings for the m' random indices requested in *challenge*-phase.
4. **Verify:** Identical to Π_{Int}, the verifier checks the signatures and validates the openings for the response. If all checks pass, the state is updated:
   - $\mathit{latestSyncedBlockHeight} := \mathit{height}(B)$
   - $\mathit{latestSyncedPayload} := \mathcal{P}_B$
   - delete entries in u for all epochs preceding B's epoch

   Else, B is stale and verifier's state remains unchanged.

Figure 2: Π_{Dyn}: dynamically increases the number of signature checks based on the number of relay instances backed by the same validator.

$$\leq \frac{3\epsilon}{4} \cdot |\{v \in V : \mathcal{A} \text{ uses } v \text{ as backing validator}\}| \tag{7}$$

Hence, for any adversary $\mathcal{A}$, the expected incentive is:

$$E_{\mathcal{A}} = \mathbb{P}(W_{\mathcal{A}}) \cdot \mathcal{M} + SE_{\mathcal{A}} \cdot (-s) \tag{8}$$

$$\leq |\{v \in V : \mathcal{A} \text{ uses } v \text{ as backing validator}\}|(\frac{3s}{4} - s) \tag{9}$$

In fact, the expectation decreases linearly w.r.t the number of backing validators used by the adversary. □

4.2 Griefing Attacks are Expensive

Since initiating Π_{Dyn} is permissionless, an adversary can attempt to grief honest relayers by intentionally inflating the dynamic security parameter. This results in increased network workload and added costs, however, we show such griefing attacks are expensive and hence impractical for a rational adversary. Assuming the mapping $u(e, v)$ on the destination chain is publicly observable, an honest relayer in the *commit* phase can always pick a

validator with the least usage, i.e., $\text{argmin}_v\, u(e,v)$ in the epoch e. Thus, an adversary with objective of griefing honest relayers has to uniformly increase the *usage counter* (u) across all possible *backing* validators for the epoch. Let C_{init} be the cost for a relayer to initiate Π_{Dyn} and C_{ver} be the cost of checking each additional signature in the *verify* phase. $\mathcal{O}(2^{x-1} \cdot |V|)$ need to be initiated by the adversary for increasing the security parameter by only x. Recall that the security parameter grows by $2\lceil log_2 u(e,v) \rceil$. We compute the griefing factor GF [But18] (ratio of additional costs on victims to cost incurred by the attacker) for the above optimal strategy.

$$GF(x) = \frac{x \cdot C_{\text{ver}}}{2^{x-1} \cdot C_{\text{init}} \cdot \frac{2}{3}|V|} \tag{10}$$

Therefore, the griefing factor asymptotically drops exponentially for as the attack prolongs. In most practical scenarios, $C_{\text{ver}} \approx C_{\text{init}}$ if not $C_{\text{ver}} \ll C_{\text{init}}$, resulting in low griefing factor for even small values of x. Moreover, the griefing attacks do not extend beyond the epoch in which they are launched, as the usage counter u is reset every epoch.

4.3 Safety and Liveness

Theorem 1 proves that Π_{Int} is ϵ-sound for a single interaction and theorem 3 shows that Π_{Dyn} is unconditionally safe with soundness error ϵ within the same epoch. We do not make any trust assumptions on the relayer for the safety results. However, for liveness, Theorem 2 in conjunction with one honest relayer ensures liveness. Note that, our soundness depends on validators being slashed for signing malicious payloads. Similar to Π_{Int}, soundness of Π_{Dyn} is affected in the highly unlikely case where blocks relayed for a period longer than the unbonding period (described in Section 3.3).

5 BEEFY: Polkadot-Ethereum Bridge

The protocol Π_{Dyn} has been instantiated to implement a trustless and decentralized bridge BEEFY from Polkadot to Ethereum and is currently live. We outline the key design decisions and security considerations.

5.1 Light-weight Succinct Finality on Polkadot

The Polkadot relay chain [BCC+20] uses GRANDPA [SK20], a deterministic finality gadget, for finalising blocks. GRANDPA is designed for secure and fast finalisation for the network, but is not suitable for light clients. In particular, finality proofs for GRANDPA are large (votes on forks not blocks) and light clients are required to maintain forks to follow finality. We design BEEFY [bee23] as an additional light-weight finality layer on GRANDPA such that:

1. BEEFY justifications satisfy the properties in Section 2.1
2. BEEFY uses the ECDSA (`secp256k1` [W+14]) signature scheme which is efficiently verifiable on-chain on Ethereum.
3. The validator sets for BEEFY and GRANDPA are identical for each epoch.

In our current design, BEEFY finality lags GRANDPA finality by a few seconds. Note that it is not necessary for every Grandpa finalised block to be BEEFY finalised. The payload is root of an MMR (Merkle Mountain Range) capturing the state of Polkadot, allowing more efficient append operations as the chain and state grow [Tod16]. MMRs also allow more efficient block inclusion and ancestry checks on the verifier side [BKLZ20].

We now describe at a high-level the BEEFY protocol in a partially synchronous setting. The protocol is designed to be modular and can be plugged into PoS chains with any deterministic finality gadget. We extend the consensus mechanism with an extra voting round. Each validator has a local belief of the following:

1. Highest GRANDPA finalized block number (G).
2. Highest BEEFY finalized block number (B).
3. Last block of the previous epoch $n-1$ (E_s^{n-1}).

Initially, they are all set to the genesis block. We assume that GRANDPA is independently finalising blocks and BEEFY's objective is to produce justifications for a subset of those blocks. Since GRANDPA does not allow forks, BEEFY validators are essentially voting on a single chain. Hence, the crucial part of the protocol is to determine the block height to vote on. The protocol runs in rounds. A *BEEFY round* is an attempt by validators to produce a BEEFY Justification. Round number is defined as the block height of a GRANDPA finalised block that the validators vote for. The protocol expects honest validators to gossip their vote. From the local view of a validator, a round ends if either of the following events occur:

1. Validator collects $2n/3+1$ valid votes for the current round, i.e. the block obtains BEEFY justifications.
2. Validator receives a BEEFY Justification for a block higher than the currently known highest BEEFY block.

In both cases the validator proceeds to determine the new round number. As a strawman approach, BEEFY validators could just pick the next GRANDPA finalised block. However, this would be inefficient as BEEFY finality is not required for every GRANDPA finalised block. Moreover, if for unknown reasons (e.g. network latency) the two mechanism drift apart, then there is a risk of BEEFY significantly lagging behind GRANDPA. To address this, we introduce a more sophisticated *round selection* mechanism which ensures the gap between GRANDPA and BEEFY finalised blocks is bounded, even if the rate at which GRANDPA finalises block is exponentially more than the rate at which BEEFY finalises blocks.

Round Selection: We define two kinds of blocks from the perspective of the BEEFY validators:

1. Mandatory Blocks: ones that MUST have BEEFY justification. Validators are required to mandatorily provide BEEFY justifications for these blocks. Last block in each epoch is defined to be a mandatory block.
2. validators are encouraged to finalize as many blocks as possible to enable lower latency for light clients and hence end users, without significantly lagging behind GRANDPA finalised blocks.[2]

The next round number r to participate is determined by BEEFY validator based on its local view of $\mathsf{B}, \mathsf{G}, \mathsf{E}_s^n$ as follows:

$$r = (1-M) * \mathsf{E}_s^{n-1} \tag{11}$$

$$+ M * min(\mathsf{E}_s^n, (\mathsf{B} + \mathfrak{P}((\mathsf{G}-\mathsf{B}+1)/2))) \tag{12}$$

where:

[2]Since the BEEFY authority set is the same as the GRANDPA authority set for any GRANDPA finalized block, the epoch boundaries for BEEFY are exactly the same as the ones for GRANDPA.

- M is 1 if the mandatory block in the previous epoch is already finalized or 0 otherwise.
- $\mathfrak{P}(x)$ returns the smallest number greater than or equal to x that is a power of two.
- min is the minimum of its arguments.

Intuitively, the next round number selected is either the mandatory block (with least height) without a BEEFY justification, or the highest GRANDPA-finalized block whose block number difference with B block is a power of two. The mental model for round selection is to first finalize the mandatory block and then to attempt to pick a block taking into account how fast BEEFY catches up with GRANDPA. In case GRANDPA makes progress, but BEEFY lags behind, validators are changing rounds less often to increase the chances of BEEFY catching up with GRANDPA[3].

5.2 Slashing for BEEFY misbehaviors

Slashing on-chain for BEEFY participants signing is crucial for security guarantees of Π_{Int} and Π_{Dyn}. To this end, we store recent payloads on-chain, however, a slash reporter can always generate an MMR ancestry proof [Tod16] to show that a block (not stored on-chain) was the prefix of a recent block. The slashing conditions are straight-forward: validators in BEEFY are slashed for signing a block that is not in the current chain (GRANDPA finalised). This includes blocks with height less than or equal to the head of the current chain but are not in the chain, and blocks with a higher block number. As long as GRANDPA is safe, validators can only be slashed for voting for blocks they do not see as finalised by GRANDPA, which honest validators will never do. In theory the full-stake can be slashed but the validator can go rogue until the slash is enforced. Hence, we settle on slashing only half the stake, keeping room for slashing misbehaviors in other subsystems of the Polkadot protocol.

5.3 Verifier Accessing RANDAO on Ethereum

We describe an implementation of a light-client verifier, specifically, the *challenge* phase in Π_{Int}: The prover sends a transaction including the commit message to a smart contract which stores the message and the block number n_{init} in which the transaction is included. The verifier's challenge is derived from the RANDAO randomness from some block with number $n_{\text{challenge}}$ in the range $n_{\text{init}} + b_{\text{delay}} \leq n_{\text{challenge}} \leq n_{\text{init}} + b_{\text{delay}} + b_{\text{window}}$ for some parameters $b_{\text{delay}}, b_{\text{window}}$. b_{delay} represents the number of slots to be waited to ensure the RANDAO value is unpredictable, while b_{window} represents the number of slots of opportunity provided to the relayer to respond to the challenge after the delay. A smart contract call made by the prover included in block $n_{challenge} + 1$ records this challenge. Then the prover can send a final transaction including the response to the smart contract, which verifies according to the interactive protocol. We note that smart contracts on Ethereum have access to the RANDAO randomness from the previous block as well as the block number, as well as the slot numbers since the merge of EIP-4788 [SDR+22] as part of Dencun upgrade [PST24] on Ethereum Mainnet [4].

[3] Once a validator picks a new round r (and the validator casts a vote), it ends the previous one, no matter if finality was reached (i.e. the round concluded) or not. Votes for an inactive round are not propagated.

[4] The analysis is more involved if the smart contract has only access to block numbers. Block producers are assigned to slot numbers in an epoch, not block numbers. Some slot numbers will have more adversarial slots before them and hence more choices for the randomness. By choosing whether or not to produce blocks in earlier slots, which do not affect the number of choices for the randomness, the adversary may be able to ensure that the sampling block occurs at their choice of slot number.

5.4 RANDAO Bias Analysis

We instantiate the random function $\mathcal{R}$ (used in the *challenge*-phase) in Π_{Dyn} with the RANDAO beacon on Ethereum. While we assumed $\mathcal{R}$ is uniformly random and unpredictable, it is well-known that RANDAO is biasable [Edg23, AW24]. We focus on quantifying the bias our specific objective: an attacker who wants to bias an interactive protocol (e.g., Π_{Dyn}) that uses RANDAO. We analyze the bias and how it affects our security parameter.

In this section, we consider public-coin protocols with *Commit-Challenge-Response* phases and abstract away from Π_{Dyn}. In a public coin protocol, the verifier challenge is chosen uniformly at random from some challenge set S_c, however if the verifier is implemented on a blockchain, adversarial participants can introduce bias in the randomness. We quantify this bias as follows:

Definition 3. A verifier V of a public-coin protocol is μ-biasable if for any adversary, for any challenge $c \in S_c$, $\Pr[\text{V produces } c] \leq \mu/|S_c|$.

Assumptions: We assume that at least 2/3rd validators are honest, and the adversary cannot forge signatures or predict honest validator's randomness contributions (for Ethereum, both covered by the unforgeability of BLS signatures). As in the previous analyzes, though they don't make it explicit, we assume that an attacker is unable to prevent an honest block producer's block from being included in the chain. Though unlikely, such attacks are feasible using the attack on LMD Casper outlined in [NMRP21, SNM+22] or by performing denial of service attacks on the block producers whose identity is public, however we exclude it from our analysis. Under the above assumptions, we derive a μ s.t the interactive protocol $\Pi_{\text{Int}}^{\text{Eth}}$ (Π_{Int} instantiated with Ethereum as the target chain) has a μ-biasable verifier protocol.

The key quantity is the tail length T, the number of slots with adversarial block producers in sequence before the RANDAO value is used at the end of the epoch or the challenge block for BEEFY. The last honest block producer before this point produces a block that must be included, whose contribution to the randomness is random and unknown in advance. The adversarial contributions the randomness are fixed by this point, so the adversary has the choice of publishing a block or not. This gives them 2^T choices of randomness. We build upon the TAIL-MAX strategy described in [AW24], and modeled as a Markov Decision Process M'_G in [AW24].

Adversarial Strategy: The adversary can employ TAIL-MAX continuously and wait until the current or next epoch has an exceptionally high T_n. After b_{delay} blocks for b_{delay} longer than two epoch lengths, an adversary before committing waits until the current epoch n_{commit} (or a close epoch) has many adversarial validators at the end. Then b_{delay} blocks later, the current epoch $n_{challenge}$ may still be biasable, allowing the adversary to have a more than usual chance to get many adversarial blocks before the trigger block. We can bound this bias by calculating how much the adversary can bias by running TAIL-MAX (synonymously TM) until epoch $n_{challenge} - 2$ which gives the optimal biasability.

Analysis: We denote st_{commit} as the state at the time of commit and write $\Pr_{A,st_{\text{commit}}}[E]$ as the probability that E happens with adversary (or a policy in the MDP) A. We denote by $T_n^{(s)}$, the number of adversarial slots before a slot s occurs in epoch n. Maximising this is very similar to T_n, except that the sequence of consecutive adversarially controlled blocks may extend into the previous epoch. The optimal policy $T_n^{(s)}$-MAX for the MDP M'_G for maximising $T_n^{(s)}$ runs TAIL-MAX until epoch $n-2$.

We denote $T_n'^{(s)}$ as the length of the sequence which is in epoch n so we have

$$T_n^{(s)} = \begin{cases} T_n'^{(s)} \text{ when } T_n'^{(s)} \leq s-2 \\ s-1+T_{n-1} \text{ when } T_n'^{(s)} = s-1 \end{cases} \tag{13}$$

The distribution of $T_n'^{(s)}$ under $T_n^{(s)}$-MAX is similar to a truncated version of that of T_n under TAIL-MAX.

Concretely, we have:

$$\Pr_{\text{TAIL-MAX},st_{\text{commit}}}[T_n] \geq k] = \Pr_{T_n^{(s)}-MAX,st_{\text{commit}}}[T_n'^{(s)} \geq k] \tag{14}$$

for $k \leq s-1$, since both require a particular k slots to be adversarially controlled. We hence have the following bound:

$$E_A[2^{T_n'^{(s)}}] \leq E_{T_n^{(s)}-MAX}[2^{T_n'^{(s)}}] \leq E_{TAIL-MAX}[2^{T_n}] \tag{15}$$

Note that for the policy $T_n^{(s)}$-MAX, $T_{n_{\text{challenge}-1}}$ and $T'^{(s)}$ are independent because the Markov chains for odd and even epochs are independent.

We now compute a reasonable bound on the number of slots at the end of an epoch that it is feasible to wait for. From our simulations in [Art25], we obtained from the stationary distribution, i.e. under the continuous attack above, tail lengths of $15, 16, 17$ occur in expectation once in every 8,24.1 and 72.3 years respectively. It seems rather expensive in terms of missed block rewards to carry out the attack for that long. Without the continuous attack, tail lengths 15,16,17 only occur in expectation every 26.2,78.6,235.8 years respectively. We fix $T_{init} = 16$ as the maximum feasible tail length for the adversary to wait.

The adversary could feasibly know the block producers in epochs $n_{\text{commit}} + 1$ and $n_{\text{commit}}+2$ if the commit slot is close to the end of epoch n_{commit}. We assume that both have tail length at most T_{init}. If there is a tail length of 16 in future epoch n, then the distribution of T_{n+2d} is that of d transitions of the Markov chain from the state corresponding to tail length 16. Concretely we compute that e.g. for $d = 2$, $E_{\text{TAIL-MAX}}[2^{T_{n+4}}]$ is 172.8. Note that this decreases in d.

Putting It All Together: The adversary can take T, the number of adversarially controlled slots before the RANDAO randomness is sampled for the challenge, to be the maximum value of $T_n^{(s)}$ over b_{window} slot numbers. They have 2^T choices of RANDAO samples. Each of these choices is uniformly distributed and random (though they are not independent). Thus by a union bound for a particular challenge value $c \in S_{challenge}$, they have at most $E_A[2^T]/|S_{challenge}|$ probability of getting c as the challenge. So the verifier is μ-biasable for $\mu = \max_A E_A[2^T]$. We use union bounds to bound $E_A[2^T]$ over the sequence of adversarial blocks in each epoch, detailed further in Appendix **??**. Plugging in the parameters for $\Pi_{\text{Int}}^{\text{Eth}}$, we obtain that $\Pi_{\text{Int}}^{\text{Eth}}$ is at most 864-biasable[5]. This results in an additional $\lceil log_2\mu \rceil = 10$ signature checks in the challenge phase of Π_{Int} to negate the bias in RANDAO induced by adversarial validators on Ethereum in collusion with relayers.

5.5 Gas costs for Verification

Gas costs for verifying ECDSA signatures on Ethereum using the `ecrecover` precompile is ≈ 3500 Gwei. We compare gas costs for verifying BEEFY finality justifications using 3 different approaches across validator set sizes (assuming Polkadot's total stake rate of $\approx 50\%$, i.e., the ratio of tokens staked to total issuance):

1. Π_{Int} with no concurrency attack, but including the extra signatures required for negating RANDAO bias.

2. Naive approach of verifying all signatures.

3. SNARK circuits for verifying ECDSA signatures using UltraPlonk [bar24].

$\|V\|$	$log_2(M/s)$	Π_{Int}	Naive	UltraPLONK
10	20	44811	24129	797216
10^2	$2 * 10^2$	55152	230919	816107
10^3	$2 * 10^3$	65493	2299148	811825
10^4	$2 * 10^4$	79281	22981129	819116
10^6	$2 * 10^6$	99963	2298001141	943720

Figure 3: Comparison of isolated gas costs (in Gwei) for verifying the ECDSA `secp256k1` signatures in the three approaches. Here $|V|$ is the validator set size of the source chain. We only consider the signatures required for accepting BEEFY finality proofs. The gas costs for UltraPLONK also include membership proofs. The values for Naive approach are computed by multiplying the gas costs for verifying a single signature by $2|V|/3 + 1$.

As evident from the table, the costs for random sampling logarithmically increase with $|V|$ and are an order of magnitude less than the SNARK verification even for $|V| = 10^6$. For $|V| \geq 10^4$, the gas requirement for Naive approach already exceeds the gas limit per block in Ethereum mainnet, making it infeasible. Further, such bulk usage for verifying signatures drives up the price of gas. As of 27/11/2024, each BEEFY consensus update on Ethereum mainnet costs 0.014 ETH ($\approx$ 50 USD).

6 Applying Fiat-Shamir Heuristic

As a natural extension, the Fiat-Shamir heuristic can be applied to transform the interactive protocol Π_{Int} into a non-interactive proof of knowledge protocol Π_{FS}, enabling the generation of a compact certificate that convinces the verifier that at least one honest validator has signed the payload. Albeit in a more general setting, Compact Certificates for Collective Knowledge introduced in [MRV+21] tackles a similar problem. Their non-interactive protocol is in spirit similar to our work but caters to a different setting with weighted votes and does not focus particularly on bridge applications. In the compact certificates approach, the randomness sampled in *challenge* phase of Π_{Int} is replaced by the prover computing a hash on all publicly know data to the verifier. Π_{Int} can similarly be transformed such that the prover uses a cryptographic hash function h over $\mathcal{C}$, $\mathcal{P}_B$, *Claims*, R_σ (Merkle Root of a tree whose leaves are claimed signatures). The security parameter (number of signatures in the certificate to be checked by verifier) in Π_{FS} purely depends on the assumed hash power of the adversary to break h. Assuming that an adversary can query $Q = 2^q$ hashes, Theorem-1 from [MRV+21], shows that $m + q$ signatures are required to be checked in the certificate for Π_{FS} to achieve a soundness error of 2^{-m}. In contrast, Π_{Int} requires only m signature checks for the same soundness error 2^{-m}.

To ensure 256-bit security for Π_{FS}, $m + q$ can be set to 256. However, can we have a more realistic bound on the hash power of adversary? Interestingly, we can rely on the rationality of the adversary. We assume that the amount of value gained in attacking the bridge (Π_{FS}) is less than the market cap of Bitcoin. If the number of hashes in expectation required to mine a bitcoin block, should if used to brute force Π_{FS} gives no high a probability for an attack than the fraction of all bitcoin given as a mining reward currently, it is rational to use any hash power to mine bitcoin rather than attack the bridge. Since mining a block requires getting a 256 bit hash lower than the CurrentTarget, we can bound the maximum hash power Q of an adversary (see Equation 16). Plugging in parameters with the values at the time of writing, we obtain $q \leq 101$, where $2^q = Q$.

$$Q \leq \frac{2^{256}}{\text{CurrentTarget}} \cdot \frac{\text{CurrentBTCSupply}}{\text{BlockReward}} \tag{16}$$

[5]The analysis in this section is backed by numerical computations in [Art25].

Figure 6 compares the signature checks required by verifier for Polkadot and Ethereum for the various versions of our protocol and altair.

Network	$log_2(M/s)$	Π_{Int}	Π_{FS}^{256}	$\Pi_{\text{FS}}^{\text{฿}}$	Altair
Polkadot	576	10	256	111	201
Ethereum	3761875	21	256	122	512
Algorand	460	9	256	110	61
Binance SC	260	9	256	110	30

Figure 4: Comparison of Signature checks required in Π_{Int} with Fiat-Shamir versions and Altair for PoS networks with varying M/s values. M denotes the market cap and s denotes the minimum stake of validator. Comparison of signature checks required by the protocols Π_{Int}, Π_{FS} with 256-bit security, and $\Pi_{\text{FS}}^{\text{฿}}$ with hash power bounded via rationality assumption relative to bitcoin mining.

7 Related Work

While there has been advances in application specific (e.g. token swaps and payment channels) bridging solutions [SAB+24, MMS+19, ZHL+19, LUTZ21], we focus on approaches supporting functionality of following finality for PoS networks. Most bridge architectures [ron24, axe21, wor24] involve a centralised and trusted intermediary (via a multi-sig) that run full-nodes of the bridged chains. Time and again, the trusted centralised entities have been compromised [ABC+24] resulting in massive financial implications.

Comparison with Altair. Ethereum's Altair upgrade [alt24] introduced a light-client protocol based based on *sync-committees* [Edg22], a significantly smaller subset of validators (currently 512) responsible for attesting the finality of blocks. The sync-committee is randomly chosen for each epoch ($\approx$ 27 hours) using RANDAO beacon, and remains unchanged for a given epoch. Similar to Altair, Π_{Dyn} does not require any custom crypto or SNARK primitives on the prover or verifier side, hence can be integrated readily on existing blockchains, and particularly efficient for large validator sets. We skip the already know issues with Altair [GM23, Pre23] and focus on the advantages of Π_{Dyn} specifically for bridge applications which demand higher security guarantees and efficiency requirements:

1. For the same soundness error, the signature checks required in Π_{Dyn} are significantly lower than Altair. The current sync-committee size of 512 guarantees soundness error of $8 * 10^{-54}$ and requires 342 signature checks, while Π_{Dyn} requires only 172. Detailed further in Appendix A.

2. Π_{Dyn} is more resilient to adaptive adversaries. The sync-committee for the upcoming epoch is known in advance, hence the adversary can use the whole epoch duration to adaptively corrupt members. In contrast, the adversary has a much shorter duration, the difference between the *challenge* and *response* phase (configured per scenario), in Π_{Dyn} to corrupt adaptively.

3. All bridges would be affected if the sync-committee is compromised, while bridges using Π_{Dyn} do not have a single point of failure.

4. Π_{Dyn} lets the verifier configure the security parameter enabling it to trade-off security and efficiency depending on the crypto-economic setting.

Optimistic Techniques With Fishermen. Several bridges like Nomad [nom21] and Near's Rainbow [nea24] are examples of optimistic protocols (aka Claim and Challenge schemes) that leverage fraud-proofs [CRR13]. Typically, the light client optimistically accepts a state without verifying and relies on economically incentivised Fishermen/WatchDogs to detect invalid updates. Unlike our approach, the relayers and fishermen need to be staked to avoid spamming and forcing all signatures being verified by making false positive challenges. Further, the security of such bridges depend on the censorship resistance of target chain during the challenge period. Dynamic transaction fees on the target chain worsen the issue, resulting in high challenge rewards and stakes. Moreover, the challenge period needs to be long trading off latency and security. In comparison, our solution requires less data since all signatures do not need to be published on chain and our bridge has lower confirmation latency. If the target chain is censored, our protocol loses liveness rather than safety but optimistic protocols lose safety.

SNARK based bridges. Recently, embedding SNARK-based on-chain light-clients has been quite popular approach for trustless bridges [BMRS20, VGS+22, XZC+22]. Accountable Light Client system introduce in [CSSV22] guarantees that a large number (e.g.1/3rd) misbehaving validators can be identified when a light client is misled, a crucial property for PoS blockchains. These solutions require custom SNARKs and cryptographic primitives like aggregatable signatures which are difficult to implement in existing networks. Our approach is simpler and secure implementation is less involved [CET+24].

Threshold Signatures. An alternative approach is using a threshold signature scheme with a single public key for all validators, as adopted by Dfinity [Gro21]. These schemes typically use secret sharing for the secret key, which has two main drawbacks. First, they require a communication-intensive distributed key generation protocol for setup, which is challenging to scale for large validator sets (even with 100 validators). Despite recent advancements [GJM+21, Gro21, GHL21], implementing such setups across a large peer-to-peer network remains difficult and may need repeating when the validator set changes. Second, secret sharing-based threshold signatures do not reveal which subset of validators signed, lacking accountability. Dfinity [Gro21] employs a re-shareable BLS threshold signature, maintaining the same public key even with validator set changes. This provides a constant-size proof for the verifier but fails to identify misbehaving validators or the specific validator set responsible, as the signature remains the same for any threshold subset.

Randomness Beacons: There is a series of work on randomness beacons for blockchains [CMB23]. Our work directly benefits from advances in unbiasabale and unpredictable randomness generation by blockchains. Schemes based on Publicly Verifiable Secret Sharing [BSKN20, SJSW20] and Verifiable Delay Functions [LW15, SJH+20] are immune against *last-revealer attacks* directly improves the efficiency of our random sampling protocol, since we do not need to increase the security parameter to negate the bias.

8 Conclusion and Future Work

We presented an interactive light-client protocol Π_{Dyn} for PoS blockchains using on-chain randomness. Π_{Dyn} leverages crypto-economic arguments to drastically improve efficiency (on-chain computation usage) without compromising security. We demonstrated the practicality of our protocol by instantiating a trustless and decentralized bridge between Polkadot and Ethereum. As future work, we plan to design incentive mechanism's for the relayer market. We are interested in settings where the relayers are incentivised by *public good funding* (DAO treasuries) [LZHZ21, VVJ23, pol24] as well as fees generated by users of the bridge.

Acknowledgements.

We thank Aidan Musnitzky, Alistair Singh, and Vincent Geddes from Snowfork and the Bridge team at Parity Technologies for fruitful discussions on practical challenges in implementation. We also thank Jeffrey Burdges, Robert Hambrock, and Syed Hosseini for useful feedback and for reviewing drafts of this work. Lastly, we thank Alfonso Cevallos and Handan Kilinç Alper for their involvement in the early stages of the project.

References

[ABC+24] André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. Sok: Security and privacy of blockchain interoperability. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3840–3865, 2024.

[alt24] Altair light client – sync protocol. https://ethereum.github.io/consensus-specs/specs/altair/light-client/sync-protocol/, 2024.

[Art25] Anonymised Artifact. Randao analysis. https://anonymous.4open.science/r/ccs25-470-randao-analysis-F9F4/, 2025.

[AW24] Kaya Alpturer and S. Matthew Weinberg. Optimal randao manipulation in ethereum, 2024.

[axe21] Axelar: Connecting applications with blockchain ecosystems. https://www.axelar.network/whitepaper, 2021.

[bar24] Ultraplonk verifier by aztec. https://github.com/AztecProtocol/barretenberg/tree/master/sol/src/ultra, 2024.

[BCC+20] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinç Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. Overview of polkadot and its design considerations. *CoRR*, abs/2005.13456, 2020.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016.

[bee23] Polkadot specifications: Beefy. https://spec.polkadot.network/sect-finality#sect-grandpa-beefy, 2023.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[BKLZ20] Benedikt Bunz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 928–946. IEEE, 2020.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *IACR Cryptol. ePrint Arch.*, page 352, 2020.

[BSKN20] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. RandPiper – reconfiguration-friendly random beacons with quadratic communication. Cryptology ePrint Archive, Paper 2020/1590, 2020.

[But18] Vitalik Buterin. A griefing factor analysis model. https://ethresear.ch/t/a-griefing-factor-analysis-model/2338, 2018.

[CET+24] Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. Sok: What don't we know? understanding security vulnerabilities in snarks. *CoRR*, abs/2402.15293, 2024.

[Cha22] Chainalysis. Vulnerabilities in cross-chain bridge protocols emerge as top security risk. https://www.chainalysis.com/blog/cross-chain-bridge-hacks-2022/, 2022.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493 – 507, 1952.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[CMB23] Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 75–92, 2023.

[Cra96] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *Ph. D.-thesis, CWI and U. of Amsterdam*, 2, 1996.

[CRR13] Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Information and Computation*, 226:16–36, 2013. Special Issue: Information Security as a Resource.

[CSSV22] Oana Ciobotaru, Fatemeh Shirazi, Alistair Stewart, and Sergey Vasilyev. Accountable light client systems for pos blockchains. *IACR Cryptol. ePrint Arch.*, page 1205, 2022.

[Edg22] Ben Edgington. One page annotated spec. https://eth2book.info/capella/annotated-spec/, 2022.

[Edg23] Ben Edgington. *Upgrading Ethereum*, chapter RANDAO Biasability. Ben Edgington, 2023.

[ell22a] Nomad loses $156 million in seventh major crypto bridge exploit of 2022, 2022. https://hub.elliptic.co/analysis/nomad-loses-156-million-in-seventh-major-crypto-bridge-exploit-of-2022/.

[ell22b] Over $1 billion stolen from bridges so far in 2022 as harmonys horizon bridge becomes latest victim in $100 million hack, 2022. bit.ly/3fvlIME.

[GHL21] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. ePrint 2021/1397, 2021.

[GJM+21] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. ePrint 2021/005, 2021.

[GM23] Vincent Geddes and Aidan Musnitsky. Snowfork's analysis of sync committee security. https://forum.polkadot.network/t/snowforks-analysis-of-sync-committee-security/2712/10, 2023.

[Gro21] Jens Groth. Non-interactive distributed key generation and key resharing. ePrint 2021/339, 2021.

[HB24] Mor Harchol-Balter. Introduction to probability for computing. https://www.cs.cmu.edu/~harchol/Probability/chapters/chpt18.pdf, 2024.

[LUTZ21] Rongjian Lan, Ganesha Upadhyaya, Stephen Tse, and Mahdi Zamani. Horizon: A gas-efficient, trustless bridge for cross-chain transactions. *CoRR*, abs/2101.06000, 2021.

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366, 2015.

[LZHZ21] Lu Liu, Sicong Zhou, Huawei Huang, and Zibin Zheng. From technology to society: An overview of blockchain-based dao. *IEEE Open Journal of the Computer Society*, 2:204–215, 2021.

[MMS+19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[MRV+21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 626–641. IEEE, 2021.

[nea24] What is the rainbow bridge? https://doc.aurora.dev/bridge/introduction/, 2024.

[NMRP21] Michael Neuder, Daniel J. Moroz, Rithvik Rao, and David C. Parkes. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders. *CoRR*, abs/2102.02247, 2021.

[nom21] Nomad protocol. https://docs.nomad.xyz/the-nomad-protocol/overview, 2021.

[pol24] Polkadot Wiki: Treasury. https://wiki.polkadot.network/docs/learn-polkadot-opengov-treasury, 2024.

[Pre23] James Preswitch. Altair has no light client. https://prestwich.substack.com/p/altair, 2023.

[PST24] Ethereum Foundation Protocol Support Team. Dencun mainnet announcement. https://blog.ethereum.org/2024/02/27/dencun-mainnet-announcement, 2024.

[rek24] Rekt leaderboard. https://rekt.news/tr/leaderboard/, 2024.

[ron24] Ronin bridge documentation. https://docs.roninchain.com/apps/ronin-bridge, 2024.

[SAB+24] Giulia Scaffino, Lukas Aumayr, Mahsa Bastankhah, Zeta Avarikioti, and Matteo Maffei. Alba: The dawn of scalable bridges for blockchains. *IACR Cryptol. ePrint Arch.*, page 197, 2024.

[SDR+22] Alex Stokes, Ansgar Dietrichs, Danny Ryan, Martin Holst Swende, and light-client. Eip-4788: Beacon block root in the evm. https://eips.ethereum.org/EIPS/eip-4788, 2022.

[SJH+20] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. Cryptology ePrint Archive, Paper 2020/942, 2020.

[SJSW20] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89, 2020.

[SK20] Alistair Stewart and Eleftherios Kokoris-Kogia. GRANDPA: a byzantine finality gadget. *CoRR*, abs/2007.01560, 2020.

[SNM+22] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 560–576. Springer, 2022.

[sno24] Snowbridge: A trustless bridge between polkadot and ethereum. https://github.com/snowfork/snowbridge/, 2024.

[Tod16] Peter Todd. Making utxo set growth irrelevant with low-latency delayed txo commitments. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-May/012715.html, 2016.

[VGS+22] Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An ultralight blockchain client. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 597–614. Springer, 2022.

[VVJ23] Paul Van Vulpen and Slinger Jansen. Decentralized autonomous organization design for the commons and the common good. *Frontiers in Blockchain*, 6:1287249, 2023.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[wor24] Wormhole architecture overview. https://wormhole.com/docs/learn/fundamentals/architecture/, 2024.

[XZC+22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 3003–3017. ACM, 2022.

[ZHL+19] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 193–210. IEEE, 2019.

A Efficiency Comparison with Altair

Let X be the random variable denoting the number of malicious validators being picked in the Sync-Committee. Since the validator set on Ethereum is $\geq 10^6$, we can approximate the hypergeometric distribution of X to $X \sim Bin(512, p)$, where $p = 1/3$ is the probability of picking a malicious validator from byzantine assumptions. The probability that a majority of malicious validators comprise the sync-committee can be bounded using Pretty Chernoff's inequality [Che52, HB24] as follows:

$$Pr(X - E[X] \geq \lambda) \leq e^{-\frac{2\lambda^2}{n}}$$

Substituting $\lambda = n/3$ and $E[X] = n/3$, we obtain soundness error bound

$$Pr(X \geq 2n/3) \leq e^{-2n/9}$$

Plugging the values for Ethereum, we obtain a bound on probability the a super-majority malicious validators were elected in the sync-committee (soundness error) $Pr(X \geq 342) = 8 \times 10^{-54}$. To obtain the same soundness error via Π_{Dyn}, the number of signature checks (security parameter) required is 176 (i.e., $log_2(1/Pr(X \geq 342))$). In fact, the result can be generalised: the ratio of signature checks required in Π_{Dyn} to Altair for achieving the same soundness errors is $(log_2\ e)/3$.