HyperLoop: Rationally secure efficient cross-chain bridge

Aniket Kate *† Easwar Vivek Mangipudi*, Charan Nomula*, Raghavendra Ramesh*,
Athina Terzoglou† and Joshua Tobkin*
*Supra Research
Email: {e.mangipudi, c.nomula, r.ramesh, j.tobkin}@supraoracles.com
†Purdue University
Email: aniket@purdue.edu, athina.terzoglou@gmail.com

***Abstract*—Cross-chain bridges, realizing the transfer of information and assets between blockchains, form the core of blockchain interoperability solutions. Most existing bridge networks are modelled in an honest-malicious setting, where the bridge nodes are either honest or malicious. Rationality allows the nodes to deviate from the protocol arbitrarily for an economic incentive. In this work, we present HyperLoop, an efficient cross-chain multi-signature bridge and prove that it is safe and live game-theoretically, under the more realistic rational-malicious model.**

As rational bridge nodes are allowed to deviate from the protocol and even collude, a monitor mechanism is necessitated, which we realize by introducing whistle-blower nodes. These whistle-blowers constantly check the operations of the bridge and raise complaints to a complaint resolution network in case of discrepancies. To enforce punishments, it is necessary for the nodes to stake an amount before participating as bridge nodes. Consequently, a cap on the volume of funds transferred over the bridge is established. We describe a sliding window mechanism and establish a relation between the stake and the sliding window limit necessary for the safety of the bridge.

Our design yields an economic, computation, and communication-efficient bridge. We realize and deploy our bridge prototype bridging Ethereum and Polygon chains over testnets. For a 19-node bridge network, each bridge node takes an average of only 3 msec to detect and sign a source chain request, showing the highly efficiency and low-latency of the bridge.

I. Introduction

Cross-chain bridge protocols provide the bridging functionality to transfer financial assets across blockchains. As per DefiLlama on 18th Dec 2024, around USD 930 million of value is bridged in the preceding 24 hours. As the bridge protocols gain popularity, the attacks on them have also seen a rise. Since May 2021 as of Aug 2024, these attacks have accounted for approximately USD 3.2 billion in stolen assets [17]. The attacks on bridges show that the bridges form the weakest link in the ecosystem making the security of the bridge paramount.

Like most cryptographic and multiparty computation schemes [23], [24], [19], [25], [33], [20], bridge protocols [5], [15], [16] have been proposed whose security has been discussed and argued in what we call the 'honest-malicious' model where a subset of the parties are always assumed to be honest. However, with the increasing total value being withheld in the blockchains, assuming some nodes handling this large value to behave honestly is not realistic. The reality is that nodes may interact arbitrarily for economic gains. This particular behaviour makes many existing bridge mechanisms [5], [15], [16] extremely vulnerable to large volume of funds being usupred. Hence there is a dire need to analyze bridges in a more realistic rational setting, which is the focus of this work.

Our multi-signature bridge protocol is simple. The bridge nodes watch for bridge requests from users on the source chain. Then they submit transactions to the destination chain endorsing the requests observed. Bridge smart contracts on the destination chain wait for a threshold number of endorsements and then they do the required response. Typically, these requests are in the form of locking a certain volume of assets on the source chain, and responses are in the form of minting or releasing the corresponding volume of assets on the destination chain. Since all the honestly behaving nodes have the same view of the source chain, they sign same set of transactions and blocks, this removes the need for consensus or interaction among the bridge nodes.

Rational setting entails that the bridge nodes participate correctly in such a bridge protocol only if it is profitable to them. Towards that goal, we design an incentive-penalty mechanism that yields a game-theoretically stable bridge protocol.

- To enforce penalties on the bridge nodes, they need to stake a certain amount first. Then the maximum penalty that can be imposed is the staked amount. Then the bridge nodes could collude and place fraud endorsements on the destination chain for a bridging amount greater than their cumulative stake of the bridge nodes. The important thing to note is that the maximum penalty is not a deterrent here as the profit gained by the bridge nodes is more than the loss owing to their penalty. So, naturally we need to limit the bridged amount of transfers.
- Note that it is still allowed by the protocol for the bridge nodes to collude and fraudulently place a bridge response transaction on the destination under the limit. Hence, we need *whistle-blowers* to constantly monitor the source and destination chains and raise complaints whenever a source request - destination response mismatch occurs.
- To resolve the complaints, we need another entity since rational whistle-blower nodes can themselves collude

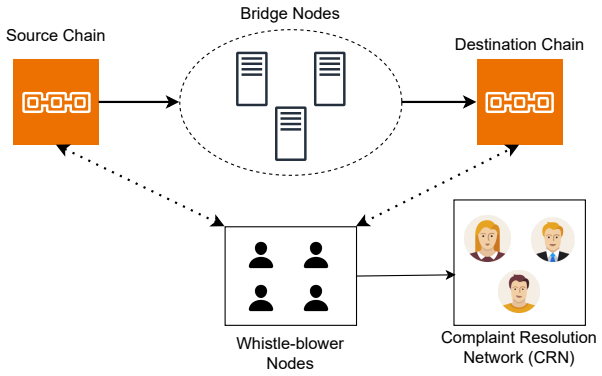

Fig. 1: Architechture of the HyperLoop. Whistle-blower nodes follow the transactions on both chains and raise a complaint to CRN when there is a discrepancy.

with the bridge nodes. We use a Complaint Resolution Network (CRN) network of nodes for this purpose. If a whistle-blower's complaint is true, then the colluding bridge nodes' stakes are confiscated. To prevent false alarms by whistle-blowers, we mandate a minor stake on the whistle-blower nodes whenever they are raising complaints. When found that the complaint is false, then the complaining whistle-blower's stake is confiscated.

A pertinent question is: - if you have a set of CRN nodes to behave honestly as a network, why can't they offer the bridge node services themselves? It is to be noted that the CRN network is external to the bridge and does not participate in the regular mechanism at all. They are mostly dormant and are approached only in extreme cases of discrepancy in transactions by the bridge nodes. Penalization ensures the security of the bridge, and hence, bridge nodes are not incentivized to collude, making collusion/discrepancy a very rare event. Thus, the honest-malicious nature of the network is limited to the CRN network, which does not participate in the network and appears only on a rare event of an active attack. Moreover, it is quite challenging to totally avoid such honesty assumption, without making any extra assumptions on the functionality of the chains than what they traditionally offer. This is especially the case when *both* the bridge nodes and whistle-blower nodes are allowed to be rational and collude. Without a resolving entity, all the rational entities will simply collude to perpetually usurp funds. Refer Section H for a comment on how to avoid the CRN with extra assumptions on the chains.

Thus, our HyperLoop comprises three sets of actors: staked bridge nodes, monitoring whistle-blower nodes, and a Complaint Resolution Network (CRN). The architecture of the bridge is provided in Figure 1.

It must be noted that even with whistle-blower nodes, the bridge nodes may collude and sign off large value transaction before it can be caught by the whistle-blower nodes. To prevent this, a limit on the total value that can be transferred in a window should be limited. We realize this limit using a sliding window mechanism to maximize the value bridged in a given time. See Section V for further details. There is a natural turnaround time for whistle-blowers to detect a collusion and report to the CRN network that validates and resolves the complaint. This turnaround time would be the sliding window length.

Non-interactivity. Whenever a cross-chain request or event appears on the source chain, each of the bridge nodes independently generates a signature on the request and forwards it to the destination chain. Bridges like CCIP [5] use a consensus protocol among all bridge nodes to agree on the source transaction(s) and generate a signature. We leverage the fact that all the honestly behaving bridge nodes have a consistent view of the source chain. The advantage of this is twofold: first, a simple majority of the number of signatures from the bridge nodes is sufficient to prove the validity of the source chain events. This makes the bridge network operate in the $2f+1$ model requiring simple majority instead of the $3f+1$ model requiring super majority of signatures for f Byzantine nodes in the bridge network. Second, it completely avoids interaction among the bridge nodes, improving computational and communication efficiency. This also allows supporting higher transfer values for the same stake of bridge nodes.

Additional features of HyperLoop include batching requests for efficiency, allowing reverts of requests to ensure quality of service. See the full version of this paper [9, Appendix C] for further details. HyperLoop allows both permissioned and non-permissioned nodes to offer the services.

Towards proving the security of the protocol, we propose an ideal functionality of the protocol that explicitly allows for collusion among rational parties. Then, using game theoretic and cryptographic analyses, we build a simulator that provides an indistinguishable view in the ideal world-real world paradigm.

We implement the HyperLoop protocol and deploy between Ethereum and Polygon testnet for different numbers of bridge nodes and transactions. Each bridgenode takes $\sim$ 3msec to detect a request event and generate the signature on the transaction. The total end-to-end delay from the point of emitting the event on the source chain till the finalization of the transaction on the destination chain is dependent on the chain finality times, and for the Ethereum-Polygon bridge, it is ~ 5 sec for a $19-$node bridge setup.

Contributions.

- We construct HyperLoop, a bridge protocol that requires a simple majority of bridge node signatures (under honest-malicious setting) in a non-interactive way.
- Using game theoretic analysis, we derive an incentive-penalty mechanism and show that the non-collusion is the dominant strategy for the bridge nodes so that the crypto-economic security of the bridge protocol is assured. As far as we know, we are the first ones to analyze a bridge protocol in the much-needed rational-malicious setting.
- We realize a decentralized collusion detection mechanism by introducing incentivized whistle-blowers, who can also

be rational.

We also introduce the sliding window limit on high-value transfers so that the loss due to collusion is capped. We then realize different batching strategies, through which we make our bridge design gas-efficient and scalable.

II. Related Work

Initially, approaches like wrapping[22] to use assets like Bitcoin on other blockchains have been developed, especially to tap the economic potential of Bitcoin on different DeFi applications on other chains. The standard way was to deposit some Bitcoins in a custodian's account on the Bitcoin chain, and then the custodian is to mint the same amount of *wrapped Bitcoin (wBTC)* on the destination chain. These wBTC can then be used for different applications and, when needed, exchanged back to the original Bitcoin on the Bitcoin network, typically in a $1:1$ ratio. This involved trusting the custodian. Our focus is to avoid centralized custody. This trust can be decentralized by using a threshold-distributed version of the custodian.

Trustless Bridges. Trustless bridges work without adding any extra trust assumptions. Examples include bridges based on zero-knowledge proofs ([35]), atomic swaps[29], [34], and on-chain book-keeping of validator committees [10], [7].

- ZK bridges require at least one honest node to provide the proof of correctness of requests. However, they suffer from high proof generation times — for example, zkBridge takes 20 sec to generate the proof (using multicores and multithreading), whereas our HyperLoop takes 3 msec to the bridge.
- Atomic swap mechanisms [29], [34] offer a bridging functionality, but they require another entity on the destination chain to set up keys with the requesting client and publish transactions. This process can not be automated through smart contracts.
- Checking the set of validator signatures [12] of the source chain at the destination chain requires validating the signatures of a typically high number of validators (the number of Ethereum validators is of the order of a million). It also involves updating the validator information continuously at the destination chain. All these approaches lead to high latency. This latency becomes a bottleneck especially with the latest chains [2], [11] which achieve fast finality, even sub-second finality times.

Recent bridges like XCLAIM [36] and Across [1], assume the existence of players like liquidity providers who constantly monitor the source chain for requests and respond. While XCLAIM has been shown to be crypto-economically secure, it (still) requires a proof of honoring the request by the liquidity provider. This results in high latency for the bridging operation.

Trustless bridges are fine for use-cases that tolerate high latencies. The focus of this paper is towards a low latency decentralized multi-signature based bridges.

HyperLoop is a multi-sig chain or a 'middle chain' where confirmation from a group of staked bridge nodes is sought for an event communicated between chains. Some popular examples in the space are Wormhole [16] and Axelar [14]. The main distinguishing feature is HyperLoop is focused mainly on achieving low latency while allowing rational behavior of nodes. It essentially tries to achieve a sweet spot between the latency and trust-assumption trade-off. HyperLoop is a pairwise bridge solution and differs from interoperability-focused systems like Cosmos [4] and Polkadot [8]. A detailed discussion of the comparison of HyperLoop with the latest famous chains is presented in Appendix A.

Rationality and Collusion. Multiple attempts have been made to study cryptographic protocols in the rational setting over the period of time. Gordon et al. [28] propose a rational secret-sharing scheme initially. Beyond rationality, study of collusion [30], [31], [32] has seen an uptick in the recent times with the economic aspect of blockchains taking prominence. Significant number of existing mechanisms and protocols remain insecure against colluding parties in the multi-party setting. Mangipudi et al. [31] have studied and proposed a collusion deterrent scheme for providing escrow services to clients. Gong et al. [27] study collusion prevention schemes in the private information retrieval mechanism. However, since the protocols differ significantly from one another, the mechanism design differs for each application. In this work we design a mechanism to deter collusion in cross-chain bridge application.

Recently, a new framework for rational protocol design (RPD) was proposed by Garay et al. [26]. In contrast to previous works in rational cryptography, they consider a zero-sum two-player game between the protocol designer and the attacker, where the designer selects a protocol and the attacker (with knowledge of the protocol) selects his strategy. In this framework, both the designer's and the attacker's objective is to maximize their utility. Following work by Badertscher et al. [18], showed that the Bitcoin protocol is secure against rational adversaries implying that no coalition has an incentive to deviate from the protocol given that the remaining parties follow it.

III. System Setup

In this section, we set up the preliminaries and definitions required for the rest of the paper.

HyperLoop is a pairwise bridge protocol that connects two blockchains, enabling information and asset transfer between them. There are three actors in our protocol – a network of bridge nodes, a set of whistle-blowernodes, and a Complaint Resolution Network (CRN) network. The bridge nodes run the clients of both the source and destination chains to follow the events on them. The whistle-blower nodes also run both the clients and constantly monitor and verify the actions of the bridge nodes. They complain about discrepancies to the CRN network. The protocol's security ensures that the rational bridge nodes do not deviate.

The nodes (Bridge and Whistle-blower) can be of the following kind:

- Rational node that always takes rational actions for economic incentive. The node may deviate from the expected actions or responses prescribed by the protocol. The node has a (rational) strategy to maximize the utility obtained by such a deviation. The strategy space may be known to other nodes in the protocol. However, the parties' exact strategy is unknown to the other parties.
- Malicious node that can arbitrarily deviate from the protocol. The actions taken by the node need not maximize the node's utility nor result in an economic incentive. Nodes compromised by the adversary can act maliciously and are called Byzantine.

We analyze the protocol in the rational-malicious model. The nodes may behave honestly for economic incentives. We use the word *node* throughout the paper to mean a bridge node unless otherwise explicitly stated.

Network Model. The bridge network is completely connected and asynchronous [21]. An asynchronous network entails that the messages sent by the honest parties are delivered to other honest parties eventually, and none of the messages are dropped. There is no bound on the message latency. Bridge nodes may have different views of the chain at any given instant due to the asynchrony of the network. All the bridge nodes are connected by authenticated point-to-point links and have access to a reliable broadcast channel.

Threat Model. We consider a rational-malicious model in which the protocol tolerates up to f of malicious nodes. The rest are rational. The malicious nodes can deviate arbitrarily from the protocol. In this work, we consider static corruptions, where the adversary corrupts up to f nodes at the beginning of the protocol when the network is initialized and controls all communication between the parties.

Notion of Time. Time is needed for handling revertable bridge requests (See Appendix B). Blocks of many blockchains typically carry Wall-clock time-stamps; these approximated Wall-clock block time-stamps can be used to specify the time factor for revertable requests. When the block time-stamps are not available, we use the number of blocks on the (destination) chain as the factor of time for reverts.

Staking. The bridge nodes are required to stake a predetermined amount S. The external whistle-blower nodes are also required to include a stake amount at the time of raising a complaint. Upon successful complaint by the whistle-blower node, the stake of the bridge nodes is slashed and is paid as a reward to the whistle-blower node. The whistle-blower nodes are also paid a periodic reward for the verification process. The stake amount of complaints is withheld when the complaint is determined to be false. Section V expands on the staking requirements.

The properties expected from the bridge are provided in Appendix B.

IV. HyperLoop

We first present the ideal functionality $\mathcal{F}_{\texttt{bridge}}$ of the bridge and then describe the HyperLoop protocol that realizes it.

Functionality. The functionality $\mathcal{F}_{\texttt{bridge}}$ is provided in Appendix Figure 8. The functionality interacts with the source chain $\mathcal{SC}$ and destination chain $\mathcal{DC}$, n bridge nodes and m whistle-blower nodes. The adversary can corrupt up to f bridge nodes and f' whistle-blower nodes. The functionality interacts with the corrupted nodes through the simulator $\mathcal{S}$. Here $n > 2f + 1$ and $m > f' + 1$. The non-corrupted bridge and whistle-blower nodes are rational.

All the messages for a session are identified with the session id sid. The simulator forwards the `keygen` message along with the total sets of bridge and whistle-blower nodes and the sets of corrupted parties. The functionality checks if the number of corrupted parties is within bounds and proceeds to generate key pairs for all the non-corrupted parties and forwards the `keygen` message to all the parties. All the bridges send deposit (stake) of value S. The functionality accepts requests only after all the bridge nodes have forwarded the deposits. The functionality transfers the deposits to an address which it controls.

The functionality receives the requests from the source chain through the `req` message. It checks whether the cumulative value of requests in the last window (sliding window) of size T to be less than the sliding window limit L. If it does not, a `revert` message is sent to $\mathcal{SC}$. The accepted request is forwarded to all the bridge nodes. The simulator forwards a signature on the requests on behalf of the corrupt nodes. The functionality verifies these signatures and if verified, records the index. The non-corrupt nodes just forward a sig message and the functionality locally computes the signature since it holds the secret keys pertaining to the honest bridge nodes. If the total number of signatures is greater or equal to $f + 1$ for a request, the list of signatures is forwarded to the destination chain and the whistle-blower nodes.

The destination chain forwards an `ack` to the functionality, which stores the time t' at which the message is received. This message is immediately forwarded to the whistle-blower nodes. Then the functionality checks if there is a matching source chain request. If such a request does not exist, it does nothing. This precisely captures the collusion scenario in which a destination chain transaction forwarded by the functionality is finalized without a valid source chain request.

If there is a matching source chain request, the functionality checks if the revert time window has passed. If it has, a `revert` message is forwarded and the request is reverted. However, the functionality also checks if any source chain requests have not been acknowledged within the revert time window. All such requests are reverted.

Any whistle-blower node can raise a complaint to the functionality. However, they have to deposit a minor stake before raising a complaint. This is withheld incase of false complaints and hence prevents them. If the complaint is valid, all the bridge nodes who have signed an invalid request are penalized by withholding their stake. The minor stake and a reward are forwarded to the complaining whistle-blower node upon valid complaint.

Protocol $\pi_{\texttt{bridge}}$

The system consists of two chains $\mathcal{SC}, \mathcal{DC}$, and n bridge nodes realizing the bridge functionality and m whistle-blowernodes. Each bridge node deposits value S before offering the service. It also consists of a CRN to resolve any complaints.

Setup:

- Each bridge node B_i has a secret key pair (sk_i, pk_i). Two smart contracts $\texttt{sc}_s$ and $\texttt{sc}_d$ are deployed on the source and the destination chains.
- $\texttt{sc}_d$ accepts signed messages from the bridge nodes. The public keys pk_i are registered with $\texttt{sc}_d$.
- All the parties have access to a global clock $\mathcal{T}$.

Source chain smart contract $\texttt{sc}_s$

- $\texttt{sc}_s$ accepts requests from the clients along with a value payment $\texttt{val}$.
- It computes the value $\texttt{val}_{\texttt{cum}}$ of total requests within the last time window w. Checks if $\texttt{val}_{\texttt{cum}} + \texttt{val} < L$. If yes, it accepts the request. Else, it rejects.
- Accepts signatures for $\texttt{rx}$ transactions form the bridge nodes. If a certain transaction has more than $f+1$ signatures, $\texttt{rx}$ is considered authorized and posted onto the chain.

Bridge nodes

- Each bridge node observes the events on the smart contract $\texttt{sc}_s$. An event is a request transaction transferring a value $\texttt{val}$ to sc_s.
- The bridge node awaits the finality of the transaction and a pre-determined B number of blocks on the source chain after the block in which the transaction appears on the source chain $\mathcal{SC}$.
- After waiting for the finality, each B_i, checks for the sliding window condition:
  - If the total requested value in the last time window w is less than L, admit the transaction
- After admitting the transaction, B_i generates $\sigma_i = \sigma(\texttt{dx}, sk_i)$, a signature on the destination chain transaction $\texttt{dx}$ and forwards it to the destination chain smart contract sc_d.
  - For batching, check different batching criteria to form the batch and then generate the signature.

Reverts

- Each bridge node observes the events on the destination chain for the forwarded signatures
- If the transaction $\texttt{dx}$ does not appear with in a time τ_r, the bridge node generates a signature

Destination chain smart contract $\texttt{sc}_d$

- $\texttt{sc}_d$ receives the message $(\texttt{dx}, \sigma_i)$ from the bridge nodes. It verifies each signature $\top \leftarrow Ver_1(\sigma, pk)$. If not verified, the signature is dropped. It collects the signatures to form a Quorum Certificate. The QC and the request are considered valid if the number of valid signatures exceeds the threshold $f + 1$.
- Computes the value v'_r of total requests within the last time window w.
- Checks if $v'_r + \texttt{val} < L$. If yes, accept the transaction $\texttt{dx}$ and the value transfer. Else, reject.
- If accepted, $\texttt{sc}_d$ transfers value $\texttt{val}$ to the address $\texttt{add}$ specified by the client in the transaction $\texttt{tx}$.

Complaint

- A whistle-blowernode can raise a complaint against any destination transaction $\texttt{dx}$ to the CRN by depositing a minor stake S_{wb}.
- CRN verifies the complaint, if valid, withholds the stake of all the applicable bridge nodes. Else withholds the stake of whistle-blower node.

Fig. 2: HyperLoop protocol

Protocol. In Figure 2, we present the protocol that realizes the functionality discussed above. In the protocol, we focus on value transfer requests, though any kind of request can be supported by the protocol. n bridge nodes realize the bridge between the source and destination chains. A smart contract, $\texttt{sc}_s$ is deployed on the source chain $\mathcal{SC}$; a client C forwards a request to this smart contract with the required value $\texttt{value}$. The client expects an equivalent value on the destination chain $\mathcal{DC}$ on the address $\texttt{add}$ owned by the client. Each bridge node $B_i, i \in [n]$ generates a secret key - public key pair (sk_i, pk_i) [1]. The bridge nodes act as full clients of both chains. Each bridge node B_i watches for the value request event on the source chain smart contract $\texttt{sc}_s$. They check for the requests to reach finality, with B blocks appearing after the block where the request occurs on the source chain. After confirming the finality of the request (by waiting for a certain number of blocks, etc), each bridge node individually fetches the request and generates an equivalent destination chain transaction $\texttt{dx}$. Each bridge node generates a signature $\sigma_i(\texttt{dx}, sk_i)$ using the secret key sk_i. All the honest bridge nodes have a consistent view of the source chain. All the parties in the system have access to a global clock, typically achieved using the time-stamping of blocks on the chains.

The bridge nodes admit the source chain transactions only if they agree with the sliding window value limit. The source chain, bridge nodes, and destination chain independently check this limit for redundancy. However, it is sufficient if the destination chain smart contract checks if the admitted requests satisfy the sliding window limit. The destination chain smart contract, $\texttt{sc}_d$, verifies each signature received from the bridge nodes. It collects the verified signatures for each request and forms a quorum certificate when a sufficient threshold of signatures is obtained. It independently checks if the validated

[1] In case of threshold signature, these would be key shares instead of independent keys

transactions satisfy the sliding window limit. The corresponding transfer is initiated once a request transaction achieves a valid multi-sig.

Each value request on the source chain is handled individually. The bridge nodes generate signatures $\sigma_i(\mathtt{dx}, sk_i)$ using the secret key shares sk_i. These signatures are forwarded to the destination chain smart contract sc_d which collects them to form the quorum certificate.

Rational-Malicious Setting

We consider the setting where the bridge nodes are rational and work to maximize their gains. Upto f nodes can be malicious, a single adversary controls all the malicious nodes. The rational bridge nodes can collude, illegally sign off a non-existent bridge request, and mint money on the destination chain. To prevent this, we introduce an incentive-penalty mechanism. We require the nodes to *stake* an amount S, to implement slashing and penalization. Depending on the nature of the deviation from the protocol, the penalization could include - suspension from participation for a time period, banning completely, or legal recourse in case of a permissioned setup. For the malicious node to succeed in posting arbitrary transactions, at least one rational node should collude with them. Even if f malicious nodes publish their secret information, at least one more signature from a rational node is required for the multi-sig validation.

Sliding time-window transfer limit. Staking does not fully solve the problem as the rational bridge nodes can collude, as mentioned earlier, and mint an amount greater than the staked amount on the destination chain. The penalization from the staked amount cannot act as a deterrent here, as the gained illegitimate amount could be way higher. Hence, a limit on the number of bridge requests is necessary. HyperLoop imposes a value limit of L for a time-window of length T. This time-window is implemented as a sliding window instead of the non-sliding approach. If a value v has been approved for a source chain in a time t, the HyperLoop disallows authorizing a transfer of value greater than $L - v$ in the next $T - t$ time. The limit or restriction is realized as a sliding window where the limit is imposed over the continuous time domain. The relation between L and the stake S is presented in Section V.

HyperLoop enforces this sliding window limit simultaneously at 3 places: bridge smart contracts on the source chain, the bridge client, and at the bridge smart contracts on the destination chain. This layered security avoids exploiting the loop-holes (software vulnerabilities) in one place. This is also better in terms of UX (user experience) because if a bridge request is not admitted owing to the crowded sliding window pipeline, then the client's request on the source chain itself fails. The client then needs to resubmit this request perhaps with a higher transaction fee on the source chain.

Whistle-blowers & CRN. As mentioned, the whistle-blower nodes are introduced to prevent collusion among bridge nodes. The bridge nodes could potentially keep generating fraudulent transactions that mint tokens on the destination chain. These tokens may be equally shared among the colluding nodes. By appropriately selecting the parameters of the system, i.e. the stake S, the fee f, and the complaint reward r_c, we can show that in the presence of an honest whistle-blower a bridge node is incentivized to act as a whistle-blower node and 'snitch' on the coalition. Simultaneously, the whistle-blower nodes prefer to complain compared to a promise of future payoffs by the bridge.

The whistle-blower nodes constantly monitor the source and destination chains and the signatures generated by bridge nodes and raise complaints to the Complaint Resolution Network (CRN) network in case of any discrepancy. The first node that generates a valid complaint is rewarded. This reward includes the sum total stakes of all the colluding bridge nodes. However, the whistle-blower nodes need to deposit or stake a certain value S_{wb} before forwarding the complaint. This prevents any of them from generating invalid complaints and false alarms.

The CRN network [6] is assumed to always have the threshold number of honest nodes required to correctly assess complaints and resolve them. The CRN network consists of independent nodes with access to the source and destination chain states and the transactions signed by the bridge nodes. Upon a valid complaint from a whistle-blower, the CRN network may request each bridge node for the response. If no valid response is obtained within a time window, the stake of the bridge nodes is confiscated, and the bridge nodes are publicly banned from the system. The first whistle-blower node who raises a valid complaint is then rewarded.

Collusion is an unusual event among the bridge nodes and may not occur too often, especially with a banning policy imposed on misbehaving nodes. To motivate the external nodes to validate constantly, we reward them periodically. While this periodic reward motivates them, it may also make them 'lazy' without verifying the transactions and collecting the reward. To prevent such a scenario, we request that the whistle-blowers submit proof of validation before the periodic reward. The mechanism for the periodic reward is presented in Section F.

V. Game-theoretic reward and stake analysis

In this section, we design the rewards and penalties for the different parties to prevent collusion. We first consider a purely rational model and then extend it to a rational-malicious model. After discussing the collusion scenarios, we present the analysis for setting the rewards for each party.

A. Considerations and assumptions

Before we analyze the system, we present the assumptions under which the protocol is practical.

- When multiple nodes raise a complaint to claim the reward, each node wins the race with equal probability. This is for the ease of the analysis. If we indeed consider different probabilities, it deters the nodes further to collude and snitch since they do not know if they have a higher/lower probability of success.

- The permissioned bridge nodes provide verifiable physical identities before offering the service. This ensures that the ban can be implemented and is effective. This shall also ensure that the banned bridge node can not offer the service again.
- All the bridge nodes are similar. Also, no nodes or agent(s) can impose negative payoffs on any other parties outside the protocol, such as using blackmail. However, they may offer positive pay-offs like bribes to encourage a certain strategy.
- γ is the expected reward of the bridge nodes for offering the service in the future, at any time. γ is not just the receivable payments from future payments of the *current* service offering but any other future business offering by the agents. This is because identities are physically verifiable, and any ban is public.

Game Theory Definitions. Before we proceed with the analysis of the protocols, we will define the necessary notions from game theory. Since the parties are interacting simultaneously on rounds we model the games as a repeated game. Infinitely repeated games represent games where the participants do not know when is the final round, even though the game is finite. It is important to note that for infinite horizon games, the players value the utility gained in the present higher than in future rounds. This is modeled with the discounted rate assumption. Formally,

A strategy of the player i is a function ϕ_i that assigns to each history $(a^1, a^2, \ldots, a^{s-1})$ an action $a_i^s \in A_i$. In repeated games, we are interested in subgame perfect equilibrium, defined below.

Definition 1 (Strategy)**.** *A strategy of player i in a repeated game with perfect information is a function s_i that assigns a history $(a^1, a^2, \ldots, a^{t-1})$ to an action $a_i^t \in A_i$.*

Definition 2 (Subgame perfect equilibrium (SPE))**.** *The strategy profile $\phi = (\phi_1, \ldots, \phi_N)$ in a repeated game with perfect information is a subgame perfect equilibrium if, for every player i, every history h, and every strategy ϕ_i' of player i, discounting reward generated by ϕ_i after the history h is at least as good according to player i's preferences as discounted reward generated by the strategy profile (ϕ_i', ϕ_{-i}) in which player i chooses s_i' while every other player j chooses ϕ_j.*

The following principle gives a condition for a strategy to be an SPE. Informally, a strategy s is an SPE if, and only if, no player can gain by deviating from s in a single stage and conforming to s thereafter.

Definition 3 (One-stage deviation principal)**.** *Consider an infinite-horizon games, G^∞, that is continuous at infinity. Then, the one-stage deviation principle holds. The profile s^* is an SPE if, and only if, for all i, h^t, and t, we have that $u_i(s_i^*, s_{-i}^*|h^t) \geq u_i(s_i, s_{-i}^*|h^t)$, for all s_i that satisfies $s_i(h^t) \neq s_i^*(h^t)$ and $s_i|h^t(h^{t+k}) = s_i^*|h^t(h^{t+k})$*

Collusion and its prevention. The (rational) bridge nodes may collude at any point among themselves for an economic incentive i.e., if the payoff from a deviation from the protocol is higher. They may collude to endorse non-existent bridge requests on the source chain and mint money on the destination chain essentially breaking the Validity property stated in Appendix B. However, since we enforce the sliding window limit, the bridge nodes can not transfer a value more than L in a time window of size T. Before offering the services, each bridge node stakes a deposit of value S. It is withheld in case of deviation from the expected honest behavior in the protocol. Any deviation from the protocol by the bridge nodes is publicly visible. We consider two scenarios for the bridge nodes:

- *Permissionless* nodes may join the bridge network and offer services by depositing a stake of value S. Any node may offer services as a permissionless node as soon as the stake is deposited.
- *Permissioned* nodes submit publicly verifiable identities along with the stake deposits. They enter into legal contracts before offering services. In case of dishonest behavior, the nodes are penalized by confiscating the stake, banned from offering services, and a legal course is taken if needed. Each node incurs an additional negative pay-off of value γ due to banning. This is the total pay-off they may obtain from future service offerings and the cost of repercussions owing to legal recourse, etc.

B. Design of the stake and reward value

We now compute the bounds on the rewards and stake of the nodes to join the bridge network. Due to the sliding window value limit enforcement, no collusion event can extract more than value L. Here, we assume that the whistle-blower nodes are guaranteed to detect fraud and raise a complaint to the CRN and get it resolved within the sliding window time limit of T from the start of the malicious event.

The protocol introduces a repeated game between the bridge nodes with a strategy space of three strategies - act *honest*-ly, *collude*, and collude but *snitch* on the other colluding nodes, for the stage game. Any node can either follow the protocol honestly or collude with other nodes to transfer a value of L and share the transferred value. However, each node may participate in collusion and get their share, but it may also proceed to act as a whistle-blower to snitch on the other colluding nodes. This way, the node that acts as a whistle-blower will also obtain an extra reward, thereby improving the pay-off from the system.

System setting. Each node, upon offering services honestly, receives a pay-off f for each window of time T. Each bridge node has computation cost c_b for following the protocol honest and communication cost $\bar{c}_b$ for colluding with the other bridge nodes. Any dishonest behavior is assumed to be caught and reported by the whistle-blower nodes in a time limit of T upon the transaction appearing on the destination chain. The maximum value that can be authorized by the bridge nodes in any window of time T is L. Each dishonest node loses the stake S upon being caught. All permissioned nodes, upon dishonest behavior, incur an additional negative pay-off

Stake value of the bridge node	S
Fee for following the protocol	f
Penalty for breaking Liveness	θ
Penalty for breaking Validity	$S+\gamma$
Bridge node cost per round	c_b
Bridge node cost per round (collusion)	$\bar{c}_b$
Whistle-blower stake while raising a complaint	S_{wb}
Whistle-blower reward for a successful complaint	r_c
Whistle-blower periodic reward	r_p
Whistle-blower cost per round	c_{wb}
Whistle-blower cost per round (collusion)	$\bar{c}_{wb}$
Complaint resolution turnaround time	T
Revert window time limit	τ_r
Transfer limit in a sliding time-window of duration T	L

TABLE I: Variables in the incentive-penalty system

of γ once caught. The client requesting may request a revert of the request if not honored within a time window of τ_r from the time the request is finalized on the source chain. Each bridge node receives a fee of value f for offering services honestly for every time window of size T.

We first illustrate the pay-offs and strategies using a two-node example for the forward case where the bridgenodes generate signatures on the source chain transactions and extend it to a general case of n nodes for the permissionless and permissioned node cases. We then discuss the game induced during reverts for the revertable transactions.

Two node example. Let the bridge network consist of two nodes $\{\mathsf{A}_1, \mathsf{A}_2\}$, wherein both nodes are required to endorse a bridge request for the corresponding action to be taken on the destination chain. Either party can choose a strategy among the three strategies of honesty, colluding, and snitching.

Permissionless setting: Table II presents the pay-offs each node will obtain for the different strategies for a permissionless node case in the stage game. If one of the nodes is honest, collusion does not occur, and they obtain a positive pay-off f. Under collusion, they can transfer a maximum value L together and, assuming they divide it equally, can obtain a pay-off of $\frac{L}{2}$. However, once they collude, the whistle-blower nodes raise a complaint, and both will get caught. Then, both the nodes lose their stake of S, making the pay-off $\frac{L}{2} - S - \bar{c}_b$ for collusion. However, after collusion, any node can act as a whistle-blower themselves and raise a complaint. If only one node raises the complaint, they obtain a reward r_c. If both the nodes play the *snitch* strategy and raise a complaint on the collusion, we assume each of them would win with half the probability and hence, in expectation, would receive a reward of value $\frac{r_c}{2}$. When a deviation is caught the game stops, and a new instance starts from the setup phase.

Now, we will show that it is in the best interest of the bridges to follow the protocol honestly. Assume that the bridge nodes collude at some epoch $\mathsf{e} \geq 1$, after which the game stops. For the first $\mathsf{e}-1$ epochs their utility is f for following the protocol honestly. In epoch e their utility from colluding is $\frac{L}{2} - S - \bar{c}_b$. Notice that for $r_c > 0$ a node can unilaterally increase their pay-off by playing the strategy *snitch* instead of *collude* irrespective of the other player playing *collude* or *snitch*. Hence, we assume that when there is collusion among the bridge nodes they play the *snitch* strategy. For honest behavior to be SPE, we would need the discounted utility of the *honest* strategy, that continuous indefinitely, to be greater than *snitch* strategy, therefore we need,

$\sum_{s=1}^{+\infty} \delta^{s-1}(\mathsf{f} - c_b) > \sum_{s=1}^{\mathsf{e}-1} \delta^{s-1}(\mathsf{f} - c_b)$
$+ \delta^{\mathsf{e}-1}\left(\frac{L}{2} - S + \frac{r_c}{2} - \bar{c}_b\right)$

From rearranging, we get that the fee f must satisfy,

$$\begin{aligned}\mathsf{f} &> \frac{\delta^{\mathsf{e}-1}}{\sum_{s=\mathsf{e}}^{+\infty}\delta^{s-1}}\left(\frac{L}{2} - S + \frac{r_c}{2} - \bar{c}_b\right) + c_b \\ &= (1-\delta)\left(\frac{L}{2} - S + \frac{r_c}{2} - \bar{c}_b\right) + c_b\end{aligned} \tag{1}$$

The fee f is typically collected as a percentage of the total value requested to be bridged by the client. The minimum value of f is zero. This is because it is possible that during some intervals of time, the bridge nodes do not get any fee owing to the lack of requests. This modifies the above relation to $S > \frac{L+r_c}{2} - \bar{c}_b + \frac{1}{(1-\delta)}c_b$.

In the setting with *permissioned* nodes, the nodes being caught upon collusion will lose an additional utility of γ in the *collude* and *snitch* strategies. The pay-off matrix is depicted in the Table III. The pay-off matrix is similar to the non-permissioned case except for the loss of future pay-off γ. Similar to the previous case, for honesty to be the dominant strategy in the permissioned node case, from Table III, we have the following relation $\mathsf{f} > (1-\delta)\left(\frac{L}{2} - S - \gamma + \frac{r_c}{2} - \bar{c}_b\right) + c_b$

Taking the minimum value of f to be 0 gives
$S > \frac{L+r_c-2\gamma-2\bar{c}_b}{2} + \frac{1}{1-\delta}c_b$

General case with n bridge nodes. In a scenario with n bridge nodes, a request is authorized if $f+1$ or more nodes generate a signature on the source chain transactions. Hence, the collusion scenario needs at least $f+1$ nodes to cooperate. In the non-permissioned bridge node case, the pay-off would be f for each node for honest behavior. However, when at least $f+1$ nodes collude and make a (maximum) transfer of value L, the transferred value is divided equally among all the colluding nodes with each obtaining a value of $\frac{L}{f+1}$. In a simple collusion scenario, a whistle-blower node would raise a complaint and all the colluding nodes would lose their stake S. Thus the pay-off of each node of the $f+1$ colluding nodes is $\frac{L}{f+1} - S$. All the other nodes would obtain a value f if they follow the protocol honestly.

If any colluding node plays the *snitch* strategy, they obtain a pay-off of $\frac{L}{f+1} - S + r_c$. The other f colluding nodes would obtain a pay-off of $\frac{L}{f+1} - S$. If k nodes play the snitch strategy, we assume that each of them wins the reward with equal probability and the expected reward for any of the k nodes is $\frac{L}{f+1} - S + \frac{r_c}{k}$.

Following the two node example, given a choice between a *collude* strategy and *snitch* strategy, a node will always play the *snitch* strategy. This is because the node can have a better pay-off from the system under collusion irrespective of what other colluding nodes are playing. Thus, all $f+1$ nodes would play the *collude* strategy, with each of them obtaining a pay-off of $\frac{L}{f+1} - S + \frac{r_c}{f+1}$. For the *honest* behavior to be

A_1 \ A_2	*honest*	*collude*	*snitch*
honest	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$
collude	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$\left(\frac{L}{2}-S-\bar{c}_b, \frac{L}{2}-S-\bar{c}_b\right)$	$\left(\frac{L}{2}-S-\bar{c}_b, \frac{L}{2}-S+r_c-\bar{c}_b\right)$
snitch	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$\left(\frac{L}{2}-S+r_c-\bar{c}_b, \frac{L}{2}-S-\bar{c}_b\right)$	$\left(\frac{L}{2}-S+\frac{r_c}{2}-\bar{c}_b, \frac{L}{2}-S+\frac{r_c}{2}-\bar{c}_b\right)$

TABLE II: Pay-off matrix in permissionless setting

A_1 \ A_2	*honest*	*collude*	*snitch*
honest	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$
collude	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$\left(\frac{L}{2}-S-\gamma-\bar{c}_b, \frac{L}{2}-S-\gamma-\bar{c}_b\right)$	$\left(\frac{L}{2}-S-\gamma-\bar{c}_b, \frac{L}{2}-S-\gamma+r_c-\bar{c}_b\right)$
snitch	$(\mathsf{f}-c_b, \mathsf{f}-c_b)$	$\left(\frac{L}{2}-S-\gamma+r_c-\bar{c}_b, \frac{L}{2}-S-\gamma-\bar{c}_b\right)$	$\left(\frac{L}{2}-S-\gamma+\frac{r_c}{2}-\bar{c}_b, \frac{L}{2}-S-\gamma+\frac{r_c}{2}-\bar{c}_b\right)$

TABLE III: Pay-off matrix in permissioned setting

a dominant strategy, a node should obtain a higher pay-off than any collusion strategy. This would give us the following relation $\mathsf{f} > (1-\delta)\left(\frac{L}{f+1}-S+\frac{r_c}{f+1}-\bar{c}_b\right)+c_b$.

Similarly, for the permissioned node case, we have the following relation to make honesty the dominant strategy $\mathsf{f} > (1-\delta)\left(\frac{L}{f+1}-S-\gamma+\frac{r_c}{f+1}-\bar{c}_b\right)+c_b$. Taking the minimum value of f to be zero gives the following relations:

$S > \frac{L+r_c}{f+1}-\bar{c}_b+\frac{1}{1-\delta}c_b$,

$S > \frac{L+r_c}{f+1}+\gamma-\bar{c}_b+\frac{1}{1-\delta}c_b$

Theorem 1. *With the proposed reward mechanism in Section V, with at least one honest and available whistle-blower node, under the Nash Equilibrium of the induced Bayesian game, the rational bridge nodes do not collude in both the permissionless and permissioned bridge node settings. Under such a scenario, the HyperLoop protocol of Figure 2 that includes the enforcement of the limiting sliding-time-window satisfies all the correctness properties of Section B.*

C. Collusion scenarios of bridge nodes with whistle-blower nodes

The stake value settings to prevent collusion, through which the bridge nodes can transfer a total value of L are described in Section V-B. If there is at least one honest whistle-blower node in the network, they will always raise valid complaints when needed. However, when all the whistle-blower nodes are rational, they have an incentive to collude with the bridge nodes and not raise complaints during protocol deviation. Here, we analyze the reward of the rational whistle-blower node so that they are always incentivized to raise valid complaints.

Rational whistle-blower. In this model, the bridge nodes and any external node that can act as a whistle-blower is considered rational. Let the whistle-blower nodes be denoted as $P_i, i < m$. To improve the pay-off from the system, any whistle-blower node that finds a malicious transaction can approach the bridge nodes to obtain a share in the total malicious transaction value.

Each whistle-blower has a set of actions for each epoch (stage game), that is $A = \{lazy, honest, collude, snitch\}$. We assume that the whistle-blower nodes incur zero cost when they choose the *lazy* strategy, c_{wb} when playing *honest*, and $\bar{c}_{wb}$ when colluding or snitching. For epoch e, let $a^{\mathsf{e}} = (a_1^{\mathsf{e}}, \ldots, a_m^{\mathsf{e}})$ be the action selected by the whistle-blowers. We model the game as a game with perfect information since, within the coalition, all identities are known. Also, the reward of the honest behavior does not depend on the actions taken by the other whistle-blower nodes. Notice that since the whistle-blowers are identical, their utility depends only on the number of honest and colluding/snitching whistle-blowers. Table IV presents the pay-off of any whistleblower i with respect to the action vector a_{-i}. It is important to note the bridge may deviate from the protocol even if there is no collusion with whistle-blower nodes, hence for the stage game utility, we multiply the complaint reward r_c with an indicator that is 1 if there is a deviation by the bridges and 0 otherwise. Finally, we denote by r_p the expected periodic reward given to whistle-blower nodes per round when they submit the proof of validation. We refer the details of the reward mechanism for the periodic reward to Appendix F.

	One honest in a_{-i}	All colluding in a_{-i}
lazy	0	0
honest	$r_p + r_c \cdot \mathbb{1}\{\text{deviation}\} - c_{wb}$	$r_p + r_c \cdot \mathbb{1}\{\text{deviation}\} - c_{wb}$
collude	$r_p - \bar{c}_{wb}$	$\frac{L}{m+f+1} - \bar{c}_{wb}$
snitch	$r_p + r_c \cdot \mathbb{1}\{\text{deviation}\} - \bar{c}_{wb}$	$\frac{L}{m+f+1} + r_c - \bar{c}_{wb}$

TABLE IV: Pay-off matrix of a whistle-blower i

First, to ensure that the period reward is sufficient to incentivize honest behavior we need $r_p > c_{wb}$. Otherwise, the cost of monitoring is higher than the reward and the whistle-blowers would prefer the *lazy* strategy over the *honest* strategy.

If we blindly follow the intuition from the analysis of the bridges, we would get that in a single epoch the strategy *snitch* is preferred over *collude*, and therefore any deviation would be caught. However, the bridge nodes may promise the whistle-blower nodes a share among the total value transferred through collusion from the upcoming epochs as well. In this case, the whistle-blower nodes do not have any incentive

to raise a complaint. The attack would be undetected and perpetual.

Assume that at epoch $\mathtt{e}$ the whistle-blower nodes collude with the bridge nodes. In order for the collusion to be caught, the pay-off of the strategy *snitch* and then *honest* for all rounds to be preferred over *collude* the utility of the first strategy must be greater than the second. The utility of *snitch* & *honest* strategy is $\delta^{\mathtt{e}-1}\left(\frac{L}{m+f+1}+r_c-\bar{c}_{wb}\right)+\sum_{s=\mathtt{e}+1}^{+\infty}\delta^{s-1}(r_p+r_c\cdot\mathbb{1}\{\text{deviation}\}-c_{wb})$. However, in the worst case, if there are no complaints in the future epochs, and the periodic reward is almost equal to the cost the utility from the *honest* strategy is almost zero. Therefore, we need the pay-off of epoch $\mathtt{e}$, $\delta^{\mathtt{e}-1}\left(\frac{L}{N+f+1}+r_c-\bar{c}_{wb}\right)$, to be greater than the utility of the *collude*-ing strategy. Therefore, we get,

$$\delta^{\mathtt{e}-1}\left(\frac{L}{m+f+1}+r_c-\bar{c}_{wb}\right)>\sum_{s=\mathtt{e}}^{+\infty}\delta^{s-1}\left(\frac{L}{m+f+1}-\bar{c}_{wb}\right)$$

Rearranging the above inequality, we get the complaint reward r_c must satisfy

$$r_c>\frac{1}{\delta^{\mathtt{e}-1}}\sum_{s=\mathtt{e}+1}^{+\infty}\delta^{s-1}\left(\frac{L}{m+f+1}-\bar{c}_{wb}\right)$$
$$=\frac{\delta}{1-\delta}\left(\frac{L}{m+f+1}-\bar{c}_{wb}\right)$$

For extension of the protocol with general discount functions besides exponential, see Section D-A. We further extend the analysis of the protocol to allow the adversary to have a bribing budget, see Appendix D.

Theorem 2. *The HyperLoop protocol $\pi_{\mathtt{bridge}}$, with the rational-malicious model on the bridge nodes and whistle-blower nodes, with the reward mechanism described in Section V, realizes the functionality $\mathcal{F}_{\mathtt{bridge}}$.*

The proof of the theorem has been provided in Appendix E.

Batch size	Time	Gas units	Tx fee-POL	Tx fee (ethereum)
5	1 ms	210,717	0.00895264	21 USD
10	3 ms	281,212	0.01124848	27 USD
15	3 ms	351,831	0.01407324	33 USD
20	5 ms	422,623	0.01690492	40 USD
25	5 ms	493,551	0.01974204	47 USD
30	7 ms	564,616	0.02258464	54 USD

TABLE V: Time to sign and gas costs along with transaction fee for different batch sizes for a bridge network with 19 nodes

VI. Experimentation and Performance Analysis

We implement [2]the HyperLoop protocol to bridge Ethereum and Polygon by realizing the source and destination chain smart contracts in `Solidity` along with the bridge node logic in `Rust`. We deploy the one smart contract each on `Sepolia Testnet` and `Polygon Amoy` with the respective block explorers `sepolia etherscan` and `oklink amoy` to log the different parameters of the blocks of the chains. The bridge nodes have been deployed as virtual machines on a Mac OSX with M3 chip at 2.75GHz and 8GB RAM. We time the protocol for different bridge sizes and report the gas costs for different batch sizes.

[2]https://anonymous.4open.science/r/hyperloop–3172/

Each bridge node keeps track of the source chain events. After detecting the an emitted request, generates an ECDSA signature on the constructed destination chain transaction and forwards it to the destination chain. Before joining the bridging service, each bridge node should register with the destination chain. The node registers by forwarding the public key and proving in zero knowledge, using a Schnorr-based proof that it knows the secret key of the forwarded public key. The destination chain keeps track of the public keys, anytime a bridge node leaves, the corresponding public key is removed and vice versa.

The sliding window logic is applied in each of the three places, the source, destination chains and the bridge nodes for defence in depth. The destination chain smart contract maintains a key value pair, with the key as the hash of the block and the different signatures from different bridges nodes as a list. Each signature is verified before being added to the list. When a threshold number of verified signatures are received, a transfer event is triggered and the corresponding payment is transferred to the specified destination chain public key. However, even after the threshold is reached, the bridge node values are still buffered if valid. This is essential to pay the reward to the bridge nodes for the signatures.

Batching. We consider different batching criterion – the bridge node waits for a certain fixed time period before forming the batch and forwarding. If the value of requests already fills up the sliding window limit, a batch is formed and signed.

We deploy different sets of bridge nodes with multiple transactions being batched. Each bridge node hashes the set of transactions and generates a signature on it. Hence with increasing batch size, the change in delay on the bridge node is only due to the hashing which is typically minimal compared to other operations. However, a large batchsize results in larger gas cost on the destination chain since the whole batch needs to be buffered and the respective signatures of bridge nodes collected. Hence the total number of transactions that can be batched is only limited by the block gas cost on the destination chain. Table V shows the time taken by the bridge nodes to sign the batches and the gas costs for the verification on Ethereum. The bridge nodes time includes the time to hash the transactions and generate a signature on the hash. They take 3 msec for a batch of 10 transactions and 20 msec for a batch of 200 transactions. The total gas units and the corresponding USD price for Ethereum for the verifying the transactions on destination chain is also presented in the Table V.

End-to-end delay. We time the bridging path by using two time measurements. First is the end to end delay. This is computed for a request as the difference between blockstamp time on the source chain when the request appeared and blockstamp time on the destination chains when the corresponding response transaction is finalized. The second one is with reverts. If the request is not honoured within a specified amount of time, the bridge nodes sign a revert transaction and it is posted on the source chain if threshold number of bridge nodes sign it. To compute this, we include the forward end to

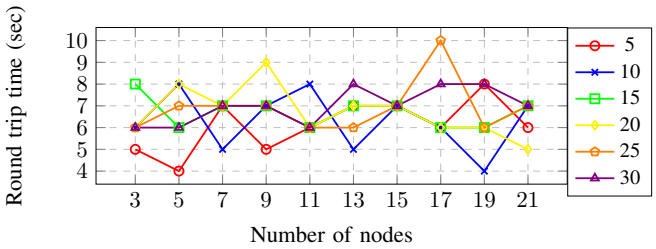

Fig. 3: End to end delay for different batch sizes

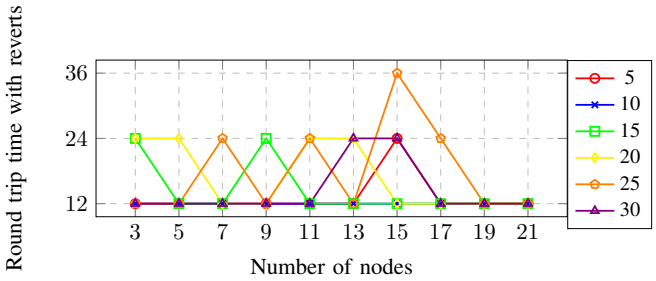

Fig. 4: End to end delay with reverts for different batch sizes

end time and the time taken to generate a revert transaction and have it finalized on the source chain.

Table V also indicates the gas costs required for fulfilling the response. The depicted gas cost is the cost when the threshold is reached and the response transaction is posted on the destination chain. For the signatures received before the threshold is hit, the gas cost would be lesser since the smartcontract just verifies the signature and adds them to a list. Also, once the the threshold is reached and the request is fulfilled, the smartcontract may still buffer the signatures from the bridges so that they can be rewarded later, depending on the reward policy for the bridge nodes. Table V also presents the cost of verification and posting the response on Ethereum.

Figure 3 and Figure 4 indicate the end to end delay for transactions for different batch sizes from 5 to 30. The timings indicate the variation with which each transaction is included in a block and finalized on the destination chain. This is not deterministic and can vary with the particular instant. However, it might be noted that all the batches have been finalized within two blocks on the destination chain. Figure 6 and Figure 5 indicate the average gas and gas when the

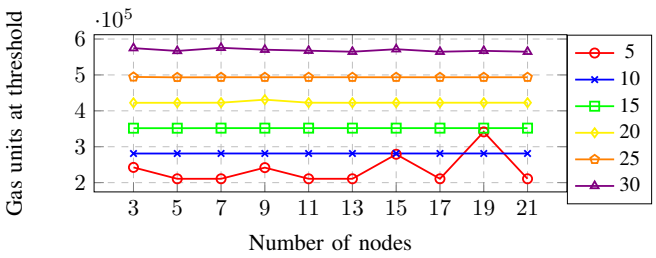

Fig. 5: Total gas units (threshold) for different batch sizes

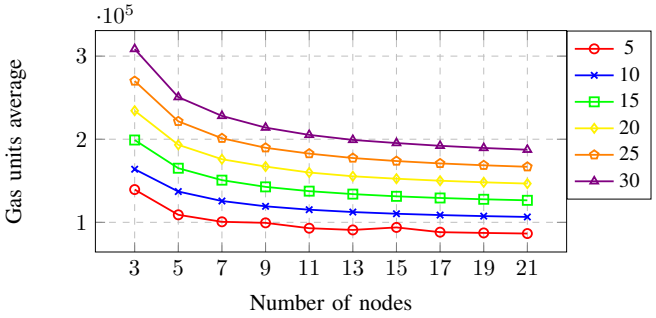

Fig. 6: Average gas units (including threshold tx) for different batch sizes

threshold is reached for the destination smart contract. When the threshold is reached, a transfer is initiated, taking higher gas than the usual signature message being forwarded to the smart contract. A detailed discussion on why honest majority is more beneficial, why multisig is better for this particular application than threshold signature and the different uses and applications of HyperLoop are found in Appendix C.

REFERENCES

[1] Across bridge. https://across.to/across-bridge.
[2] Aptos network. https://aptosfoundation.org/.
[3] Chainlink Oracle Network. https://chain.link/education/blockchain-oracles#what-is-an-oracle-network.
[4] Cosmos network. https://cosmos.network/.
[5] Cross chain interoperability protocol. https://chain.link/cross-chain.
[6] Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example. https://www.investopedia.com/tech/what-dao/.
[7] Hyperbridge. https://hyperbridge.network/.
[8] Polkadot network. https://polkadot.network/whitepaper/.
[9] Rationally secure efficient cross-chain bridgel. https://drive.google.com/file/d/1FLUK2mjR6bFUVmmT8vA5UoDp3L-OJE8_/view?usp=sharing.
[10] Succinct. https://www.succinct.xyz/.
[11] Supra. https://supra.com/.
[12] Sync committee protocol. https://docs.telepathy.xyz/telepathy-protocol/sync-committees.
[13] Tendermint. https://tendermint.com/.
[14] Axelar Network. https://axelar.network/axelar_whitepaper.pdf, 2022.
[15] Layer Zero. https://layerzero.network/, 2022.
[16] Wormhole. https://wormhole.com/, 2022.
[17] André Augusto, Rafael Belchior, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. Xchainwatcher: Monitoring and identifying attacks in cross-chain bridges, 2024.
[18] Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? a rational protocol design treatment of bitcoin. In *Advances in Cryptology–EUROCRYPT 2018*.
[19] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *ACM CCS 2008*.
[20] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography, 2002*.
[21] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM CCS 2022*.
[22] Giulio Caldarelli. Wrapping trust for interoperability: A preliminary study of wrapped tokens. *Information*, 13(1):6, 2021.
[23] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.
[24] Wenliang Du and Mikhail J Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Workshop on New security paradigms, 2001*.

[25] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.
[26] Juan Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *Symposium on Foundations of Computer Science 2013*.
[27] Tiantian Gong, Ryan Henry, Alexandros Psomas, and Aniket Kate. More is merrier: Relax the non-collusion assumption in multi-server pir. *arXiv preprint arXiv:2201.07740*, 2022.
[28] S Dov Gordon and Jonathan Katz. Rational secret sharing, revisited. In *Security and Cryptography for Networks, 2006*.
[29] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.
[30] Jonathan Katz. Bridging game theory and cryptography: Recent results and future directions. In *Theory of Cryptography Conference*, pages 251–272. Springer, 2008.
[31] Easwar Vivek Mangipudi, Donghang Lu, Alexandros Psomas, and Aniket Kate. Collusion-deterrent threshold information escrow. In *Computer Security Foundations Symposium (CSF) 2023*.
[32] Thibault Schrepel. Collusion by blockchain and smart contracts. *Harv. JL & Tech.*, 33:117, 2019.
[33] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology—EUROCRYPT 2000*.
[34] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *IEEE Conference on Security and Privacy (SP) 2022*.
[35] 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 *Proceedings of the 2022 ACM SIGSAC CCS*.
[36] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*.

APPENDIX A
COMPARISON OF HYPERLOOP WITH OTHER WELL-KNOWN CHIANS

Cosmos IBC [4] is of hub-and-spoke architecture, with Cosmos Hub in the center and zones as the spokes. All the zones are independent and run Tendermint [13] consensus protocol and communicate with other zones via Cosmos Hub using Inter Blockchain Protocol (IBC). The nodes of the zones also run the Tendermint light clients of the Cosmos Hub so that all the communication is bridged via the Cosmos Hub. The communication between zones works on the principles of relaying rather than a multi-sig approach of HyperLoop. Since Tendermint light clients cannot be run on the validators of external (to Cosmos) chains like Ethereum, it requires another network, a specialized Cosmos zone called *peg zone*, to bridge. Then this peg zone serves as multi-sig bridge between an external chain and Cosmos zones. Because this peg zone runs an instance of Tendermint Consensus protocol in a partially synchronous networking model, they need a super-majority rather than HyperLoop's simple majority. Another distinguishing factor is latency. Communication from a Cosmos zone to an external chain has to go through the relay process of the Cosmos Hub, followed by the consensus protocol of the peg zone, to reach the external chain. In contrast, there is no distributed protocol in the HyperLoopbridge design, and hence faster, apart from these factors of honest simple majority and latency.

Polkadot [8] is also of the hub-and-spoke model with a Relay chain at the Hub. The Relay chain does not host any user transactions. Only the parachains (spokes) contain the user transactions. The relay chain maintains the state and the corresponding Merkle root of all the parachains. It also provides finality to the blocks of the parachains.

The communication between parachains happens through the principles of relay bridging. Similarly, the communication between Polkadot and external chains follows the relay bridging approach.

LayerZero The central thesis of LayerZero[15] is to require two *separate* entities: oracles and relayers, necessarily to effect a transfer from a source chain to a destination chain. They claim that as long as the interests of oracles and relayers conflict, collusion attacks on any transfers using this pair of entities are not possible. LayerZero claims the selection of oracles and relayers to be configurable, though Chainlink Oracle [3] appears to have been hardwired for the Oracle role for all practical purposes. As mentioned before, we analyze any Bridge protocol from a natural rational-malicious model that has not been done before. In a rational model, the relayer and the oracle committees can simply collude and convince the destination chain of any transaction, minting currency. This leads to an undetected and perpetual attack on the system.

Also, a two-entity model yields significant power to each of the groups in terms of affecting the ordering, yielding to MEV attacks, and mounting delaying (leading to front running), back running, sandwiching and censorship attacks. In LayerZero model, a dapp is motivated to run a relayer on his own and retain control to play these attacks. Thus it is possible for gullible users of a dapp to be oblivious of such attacks by malicious dapp developers.

Chainlink's Cross-Chain Inter-operability Protocol (CCIP) [5] protocol consists of three actors: a *committing* Distributed Oracle Network (DON), an *executing* DON, and a *Risk Management Network (RMN)*. The committing DON monitors the events of an *OnRamp* smart contract on the source chain. It then bundles the bridge requests generating those events and computes a Merkle Root of this bundle. This Merkle root is then posted on to the destination chain along with a Quorum Certificate (QC) of the committing DON showing its Byzantine Fault Tolerant (BFT) consensus. The nodes in RMN also follow the bundle of bridge requests on the source chain, compute the Merkle root independently, and validate the committing DON posted Merkle root on the destination chain. Upon success, the RMN node 'blesses' this Merkle root, else it 'curses'. The nodes in the executing DON post these bridge requests along with its Merkle proof against the blessed Merkle root on the destination chain, and then they get processed on the destination chain.

Mainly the protocol has 3 steps: Merkle root posting by the committing DON after a consensus inside the committing DON, Merkle root endorsement by the RMN, relay of the bridge request and its Merkle proof by an executing DON node. In comparison, HyperLoop has just one step without any interaction in the distributed protocol during normal (non-

collusion/malicious) operations and, hence is faster.

APPENDIX B
CROSS-CHAIN BRIDGE - PROPERTIES

We define the properties we expect the bridge protocol to satisfy in this section.

We use bridge-*requests* for the transactions on the source chain asking for a message transfer or an asset transfer or requesting for some service from the bridge network. The corresponding transaction posted on the destination chain by the bridge nodes is termed bridge*response*. The Bridge requests could come with an optional *revert* period τ_{rev}, indicating that the submitter of the request expects his/her transaction on the source chain to be reverted in case an appropriate Bridge response is not posted on the destination chain within τ_{rev}. τ_{rev} is specified either as Wall-clock time approximated by block time stamps or by the number of blocks on the destination chain. Such requests are termed *revertable*, and the corresponding reverting transactions on the source chain are termed *reverted bridge requests*.

We use the following notation and definitions in describing the properties expected from the bridge:

- $\langle \texttt{req}_1, \texttt{req}_2, \ldots, \texttt{req}_k \rangle$ indicates the transactions on a chain respecting the total order from $\texttt{req}_1$ to $\texttt{req}_k$.
- $\texttt{valid}(\texttt{req}, \texttt{res})$ is a relation that holds on a tuple – a Bridge request $\texttt{req}$ and a Bridge response $\texttt{res}$, only if they are valid and successful.
- $\texttt{validRevert}(\texttt{req}, \texttt{rev})$ is a relation that holds on a tuple – a Bridge request $\texttt{req}$ and a transaction on the source chain that reverts the request $\texttt{rev}$.
- $\texttt{limit}(\texttt{req}_{i+1}, \texttt{req}_{i+2}, \cdots, \texttt{req}_{i+k})$ is a relation that holds on a sequence of Bridge requests only if they satisfy a limit condition. Typically, this condition is used to limit the total value of the amount transferred in a time duration.

We now present the properties we expect from the bridges and group them as the classical safety and liveness properties.

Safety Properties. We expect the bridge to satisfy the following validity , ordering and sliding window limiting properties for the safety of the bridge.

- *Validity*
  - Every pair of non-revertable bridge request $\texttt{req}$ and its bridge response $\texttt{res}$ must satisfy $\texttt{valid}(\texttt{req}, \texttt{res})$.
  - Every pair of revertable bridge request $\texttt{req}$ and its bridge response $\texttt{res}$ satisfies $\texttt{valid}(\texttt{req}, \texttt{res})$ if and only if $\texttt{res}$ happened within τ_{rev}.
  - Every pair of revertable Bridge request $\texttt{req}$ and its reverted Bridge request $\texttt{rev}$ satisfies $\texttt{validRevert}(\texttt{req}, \texttt{rev})$if and only if there is no Bridge response $\texttt{res}$ within τ_{rev} such that $\texttt{valid}$ ($\texttt{req}$,$\texttt{res}$) holds.
- *Ordering*
  - Let the ordered sequence of Bridge requests be $\langle \texttt{req}_1, \texttt{req}_2, \cdots \rangle$, and let the ordered sequence of Bridge responses be $\texttt{resp} = \langle \texttt{res}_1, \texttt{res}_2, \cdots \rangle$. Then, for every $1 \leq i$, if $\texttt{res}_i \in \texttt{resp}$ then $\texttt{valid}(\texttt{req}_i, \texttt{res}_i)$ holds, otherwise $\texttt{req}_i$ is a revertable Bridge request.
  - Let the ordered sequence of reverted Bridge requests be $\langle \texttt{res}_1, \texttt{res}_2, \cdots \rangle$. Then there must be an ordered subsequence of revertable Bridge requests $\langle \texttt{req}_1, \texttt{req}_2, \ldots, \rangle$ such that $\texttt{validRevert}(\texttt{req}_i, \texttt{rev}_i)$ with $1 \leq i$.
- *Sliding Window Limiting.* Let $\langle \texttt{req}_1, \texttt{req}_2, \ldots, \texttt{req}_k \rangle$ be the sequence of Bridge requests in any duration of predetermined length, say T. Then $\texttt{limit}(\texttt{req}_1, \texttt{req}_2, \ldots, \texttt{req}_k)$ holds.

Liveness Properties. The protocol needs to satisfy the following properties to ensure that the bridge is live.

- For every non-revertable Bridge request there must exist a Bridge response.
- For every revertable Bridge request, there exists exactly one of the following (XOR relation):
  - Bridge response $\texttt{res}$ within τ_{rev}, (XOR)
  - reverted Bridge request.

APPENDIX C
BATCHING AND REVERTS

Bridging every bridge request separately and individually could be inefficient especially when these are of small value transfers or of custom messages. An efficient, scalable approach is to batch multiple requests together. To improve efficiency and latency, the bridge nodes batch the value transfer requests and generate signatures on the batches. The bridge nodes form batches based on the time or volume limits of the number of transactions. Since each honest node forms the batch according to the same criteria, the batches forwarded by each honest bridge node which achieve a valid multi-sig at the destination chain $\texttt{sc}_d$ node would be the same and consistent. Each time a batch is formed and signatures are generated, the elapsed-time clock is reset locally at the bridge nodes, and the elapsed time is counted from then on afresh. Time is measured as Wall-clock time when available or in terms of the number of blocks of the destination chain. This realizes the functionality of Figure 8. Batching helps amortize the costs of multi-sig verification on the destination chain among multiple bridge requests, lowering the bridging fees.

A. Reverts

The distributed protocol of HyperLoop is set in the asynchronous networking setting. There is no theoretical time bound on the submission of the corresponding response transactions on the destination chain. There are scenarios where the user expects the response transaction on the destination chain to happen within a certain time, and if not s/he prefers reverting the transaction rather than to wait. To cater to such scenarios, HyperLoop offers optional *revertable* bridge requests, wherein bridge requests come with a time parameter τ_{rev}. Here we require the bridge smart contracts on the destination chain to ensure that the bridge requests are not honoured past their τ_{rev}. In other words, if the response transaction posted by the bridge nodes lands in a block of

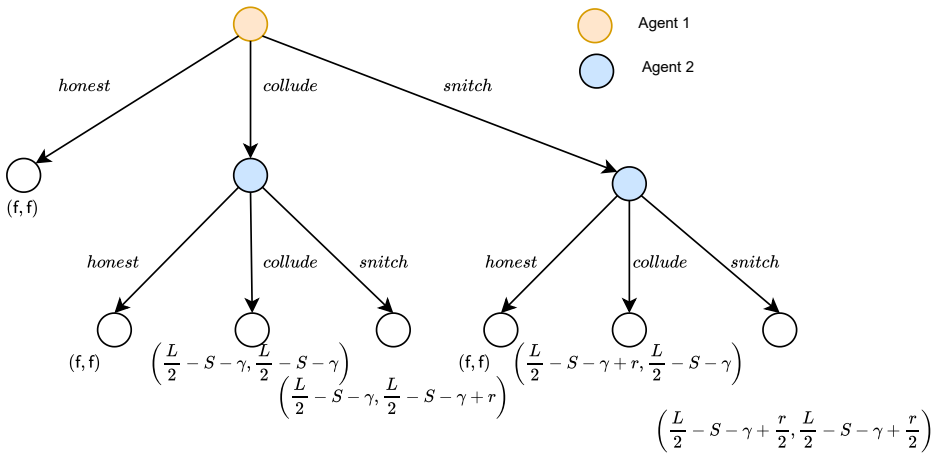

Fig. 7: Game progression among the two bridge nodes for the permissioned setting. For non-permissioned setting, consider $\gamma = 0$. With $\mathsf{f} > \frac{L}{2} - S - \gamma + r_c$, honesty would be the dominant strategy of any player.

the destination chain with a timestamp value more than τ_{rev} then they fail.

Once a requested transaction does not appear by time τ_r on the destination chain (from the time of appearing on the source chain), the bridge nodes generate a signature on a transaction reverting the request. However, the bridge nodes may collude and delay posting the revert transaction. Nevertheless, it can be observed that they do not have any economic incentive to do so. The colluding nodes can only affect the liveness of the network by delaying or withholding the revert transactions. We now show that generating the signature without delay and posting the transaction is a dominant strategy for the rational nodes.

Once again, we consider a two-node bridge network and depict the pay-offs of the nodes when both should generate a signature for the revert in Table VI. The nodes can either *delay* the transaction or *post* it immediately. The pay-offs of the two nodes under delay and posting immediately $(delay, post)$ strategies are depicted in Table VI. In the case when both play the *delay* strategy, both will lose their stake leading to the total pay-off of value $-S$. If only one of them plays the *delay* strategy, that node obtains a pay-off of $-S$, and the other node obtains the fee for honest behavior. If both the nodes *post* the revert, they will both obtain the fee f. It can be observed that whoever plays the *delay* strategy obtains a negative pay-off of value $-S$ and hence *post* is the dominant strategy of the game induced during reverting the transaction requests. To penalize the delaying nodes, the whistle-blower nodes need to impose a time limit on the delay before they expect the bridge nodes to post the revert transaction. This would be a system parameter indicating a large time window. It should be noted that the protocol itself offers only eventual revert guarantee in the revert path of the transaction.

TABLE VI: Pay-offs of rational permissionless bridge nodes under different strategies during reverts.

A_1 \ A_2	*delay*	*post*
delay	$(-S, -S)$	$(-S, \mathsf{f})$
post	$(\mathsf{f}, -S)$	(f, f)

APPENDIX D
ADVERSARY WITH A BRIBING BUDGET

In this model, all bridge nodes except the corrupted bridge nodes are considered rational. Initially, we consider at least one honest whistle-blower node. The adversary has a bribing budget of β to induce the rational parties to collude. The adversary can corrupt up to f parties; for collusion to occur, the adversary should induce (one more) the $f + 1^{st}$ party by offering a bribe of up to β. The bribe budget β is over and beyond the total stake that the adversary is willing to lose. Along with the bribe, the adversary may also forego all the transferred value L to the induced node. The rational node will accept the bribe and collude if the total gain is more than the total loss, which would be the stake withheld. In this case, the total gain for the node is $L+\beta$, and the total loss would be S. Hence, the rational node would not collude with the adversary as long as $S > L + \beta$.

In the case of rational whistle-blower nodes, the adversary may try to bribe and corrupt these nodes as well. The corrupt whistle-blower node either does not raise a complaint or raises false complaints. However, before raising a complaint, the whistle-blower nodes should deposit a minor stake, which will be withheld in case of false alarms. The only advantage the adversary gains by spending his budget on the whistle-blower nodes is briefly pausing the bridge.

In the case of rational whistle-blower nodes, the adversary induces them to collude with $f + 1^{st}$ node to perform the attack. The value $L+\beta$ that the adversary provided the $f+1^{st}$ bridge node before should be divided among the bridge node

Functionality $\mathcal{F}_{\texttt{bridge}}$

- The functionality $\mathcal{F}_{\texttt{bridge}}$ interacts with the source chain $\mathcal{SC}$ and the destination chain $\mathcal{DC}$, n bridge nodes and m whistle-blower nodes. The adversary can corrupt up to a maximum of f bridge nodes and f' whistle-blower nodes. The functionality interacts with the corrupted nodes through the simulator $\mathcal{S}$.
- The functionality controls an address pk_{sid} to which it can forward funds when necessary.
- The functionality $\mathcal{F}_{\texttt{bridge}}$ generates the secret keys locally for the honest nodes and maintains lists $SKeys$, $Keys$ for the keys and `eventList, confirms, Deposits` for the source chain and destination events. It also maintains the variables `currentTime, startTime, timeLimit, batchsize` etc. All the variables and lists are initialized to ϕ or 0 appropriately.

Key Generation Upon $(\texttt{keygen}, S, W, C_S, C_W, \{pk_i\}_{i\in\mathcal{I}_S}, \{wpk_j\}_{j\in\mathcal{J}_W}, sid)$ from $\mathcal{S}$

- Define $H_S := S \setminus C_S$ and $H_W := W \setminus C_W$ set $n := |S|$ and $m := |W|$.
- If $C_S > f$ mark S as `Corrupt` and exit. Else if $C_W > f'$, mark W as `Corrupt` and exit the procedure.
- Else sample key pairs $\{(sk_i, pk_i)\}_{i\in H_S}$ and set $Keys[i] := pk_i \forall i \in S$. Set $Skeys[i] := sk_i \forall i \in H_S$.
- Send $(\texttt{keygen}, sid)$ to all $\{P_i\}_{i\in S}$.

Deposits Upon receiving $(\texttt{deposit}, S, i, pk_i, sid)$ from either $\mathcal{S}$or party P_i, transfer the deposit value S to the public key pk_{sid} using a `transfer` subroutine and append (i, pk_i) to the list `Deposits`. If the number of deposits received is n, set `alldeposits` to $\top$.

Event requests Upon $(\texttt{req}, \texttt{val}, rid, sid)$ from $\mathcal{SC}$ at time t, if `alldeposits` is $\top$, compute the cumulative value of requests $\texttt{val}_{\texttt{cum}}$ in the time window $[t-w, t]$. Else, do nothing.

- If $\texttt{val}_{\texttt{cum}} \leq L$,
  - Record $\texttt{rec}_{\textsf{req}} := \langle \texttt{val}, rid, t, sid\rangle$ and forward it to all $\{P_i\}_{i\in H_S}$ and $\mathcal{S}$.
- Else, send $\langle \texttt{revert}, rid, sid\rangle$ to $\mathcal{SC}$.

Signature Evaluation

- Upon $(\texttt{sig}, \sigma_i, pk_i, i, \texttt{val}, rid, sid)$ from $\mathcal{S}$,
  - If $Signature.\texttt{verfiy}(\sigma_i, pk_i) = \top$, append (i, σ_i, pk_i) to the list $Sig[rid, sid]$. Else, do nothing.
- Upon $(sig, i, \texttt{val}, rid, sid)$ from $P_i \in H_S$,
  - Generate signature, $\sigma_i := Signature.\texttt{sign}((rid, sid), Skeys[i])$ and append $(i, \sigma_i, Keys[i])$ to list $Sig[rid, sid]$.

  If $|Sig[rid, sid]| \geq f + 1$ forward $(Sig[rid, sid], \texttt{val})$ to $\mathcal{DC}$and $\{P_j\}_{j\in W}$ and $\mathcal{S}$.

Destination chain finality Upon $(\texttt{ack}, rid, sid)$ at t' from $\mathcal{DC}$, record $\langle \texttt{ack}, rid, t', sid\rangle$ and forward it to $\{P_j\}_{j\in H_W}, \mathcal{S}$. Then check for the record $\langle \texttt{val}, rid, t, sid\rangle$. If it exists, continue. Else, do nothing.

- If $t' > t + \tau_{\texttt{rev}}$, then forward $(\texttt{revert}, rid, sid)$ to $\mathcal{SC}$.
- Else, do nothing.

Revert At any given time t'' if a record $\langle \texttt{val}, rid, t, sid\rangle$ exits such that $t < t'' - \tau_{\texttt{rev}}$, send $\langle \texttt{revert}, rid, sid\rangle$ to $\mathcal{SC}$.

Complaint Upon $(\texttt{complaint}, s, rid, sid,)$ from $P_j \in H_W$ or $\mathcal{S}$, check if matching records $\langle \texttt{val}, rid, t, sid\rangle$ and $\langle \texttt{ack}, rid, t', sid\rangle$ exist for the same rid.

- If yes, send $(\texttt{penalize}, sid)$ to P_j.
- Else, send $(\texttt{penalize}, sid)$ to all P_i for all $(i, \sigma_i) \in Sig[rid, sid]$, transfer the reward and stake value $s + r$ to P_j using the `transfer` subroutine.

Fig. 8: Ideal functionality of cross-chain bridge with batching and reverts

and the whistle-blower nodes. Thus, the total number of nodes sharing the value $L+\beta$ is $N+1$; since the adversary controls f corrupt nodes, they do not expect any pay-off. Assume a discount factor δ as before. The reward that needs to be paid to the rational whistle-blower node to raise a valid complaint is

$$r_c > \frac{\delta}{1-\delta}\left(\frac{L+\beta}{(m+1)} - \bar{c}_{wb}\right) \tag{2}$$

System's point of view. It should be noted that even if the system (or the firm implementing the bridge) sets the stake to be $S > \frac{L}{t+1}$, it will not lose any currency. Under collusion, the total loss to to the system is L and the total stake withheld is $> (t+1)S$. The system does not lose currency if $(f+1)S > L$ implying $S > \frac{L}{f+1}$. This is because the bribe β is being transferred between the adversary and the bridge node, and the security of the system ensures that it does not incur any loss due to the same.

Preventing false alarms from whistle-blowers. Any node that wishes to raise a complaint against the Bridge nodes first must provide a stake of value S_{wb} before making the complaint. If the complaint is valid, the sum total of all the stakes of the Bridge nodes is given as a reward along with the stake S_{wb}. However, if the whistleblower node is rational,

they may not raise a complaint; they may instead approach the Bridge nodes to share the transferred value. We resolve this in the following analysis.

A. General discounting functions

We can extend the previous results to general discounting functions. Let $\delta : \mathbb{N} \to \mathbb{R}$ be a decreasing function such that $\sum_{s=1}^{+\infty} \delta(s) u_i(a^s) < +\infty$.

Similar to Equation (1) and Section V-B, for non-permissioned and permissioned setting respectively, we get that f must satisfy:

$$\mathsf{f} > \max_{\mathsf{e}} \frac{\delta(\mathsf{e})}{\sum_{s=\mathsf{e}}^{+\infty} \delta(s)} \left(\frac{L}{2} - S + \frac{r_c}{2} - \bar{c}_b \right) + c_b \tag{3}$$

$$\mathsf{f} > \max_{\mathsf{e}} \frac{\delta(\mathsf{e})}{\sum_{s=\mathsf{e}}^{+\infty} \delta(s)} \left(\frac{L}{2} - S - \gamma + \frac{r_c}{2} - \bar{c}_b \right) + c_b \tag{4}$$

For the whistle-blower nodes the complaint reward must satisfy the following two inequalities for the rational setting and the rational-malicious model respectively(equivalent to Section V-C and Equation (2)):

$$r_c > \max_{\mathsf{e}} \frac{1}{\delta(\mathsf{e})} \sum_{s=\mathsf{e}+1}^{+\infty} \delta(s) \left(\frac{L}{m+f+1} - \bar{c}_{wb} \right) \tag{5}$$

$$r_c > \max_{\mathsf{e}} \frac{1}{\delta(\mathsf{e})} \sum_{s=\mathsf{e}}^{+\infty} \delta(s) \left(\frac{L+\beta}{(m+1)} - \bar{c}_{wb} \right) \tag{6}$$

APPENDIX E
POSTPONED PROOFS

Theroem-1

Proof. The proof of the theorem is canonical. Since at least $f+1$ signatures are required to form the Quorum on the destination chain, at least one honest node should provide a valid signature. Given a signature scheme that is secure against existential forgery, no adversary can forge the signatures of the honest nodes. No honest node would generate signatures on transactions that are out of order or have invalid requests. We assume the honest bridge nodes to be continuously available, and hence, liveness is not affected as at least $f+1$ nodes generate valid signatures for every request. The honest nodes also generate destination chain transaction requests with matching values, check the sliding window limits, and drop the requests if they exceed the limit. The adversary can corrupt a maximum of f parties and can never achieve a Quorum set of signatures on its own. Hence, all the correctness properties of the bridge are naturally achieved in an honest-malicious model. □

Theorem-2

Proof. First, we show that honesty is a dominant strategy, and then the correctness properties of the HyperLoop protocol follow. Then we should that the protocol is cryptographically secure by showing a simulator that provides an indistinguishable view of the real and ideal worlds.

When the number of bridge nodes is two, if one of the nodes plays the *honest* strategy, no matter what the other node plays, both the nodes obtain a pay-off of f. If both the nodes play the *collude* strategy, they would share the maximum value transfer L among each other, each obtaining $\frac{L}{2}$. However, with at least one honest and available whistle-blower node, the value obtained by each node would be $\frac{L}{2}$ the maximum expected pay-off from collusion for any of the nodes is $\frac{L}{2} - S$ in the permissionless setting. If the fee f and stake are designed such that Equation (1) is met, the pay-off $\frac{L}{2} - S < \mathsf{f}$. Any agent can deviate unilaterally to play the *honest* strategy and improve their pay-off to f. Thus, the strictly dominant strategy of any node is to follow the protocol honestly. This is true even if one player is playing the *collude* strategy and the other player wishes to play the *snitch* strategy. The game progression is depicted in Figure 7.

If the node wishes to play the *snitch* strategy, the second node can play any strategy. If the second node plays *honest*, both the nodes obtain f. If the second party plays either the *collude* or the *snitch*strategy, the first node obtains $\frac{L}{2} - S + r_c$ and $\frac{L}{2} - S + \frac{r_c}{2}$ respectively. With 1 is met, then f would be greater than either values. The second node can unilaterally change their strategy to *honest* to obtain better pay-off, making *honest* the dominant strategy. This is evident from the extensive form game depicted in Figure 7.

The argument extends to a n bridge node scenario where at least $f+1$ nodes are required to authorize a transaction on the destination chain. As long as Section V-B and Section V-B hold, *honest* would be the dominant strategy in the Bayesian game induced among the nodes. When players are colluding and if one of the agents decides to *snitch* on others, every other rational agent would attempt to *snitch*; they end up with every node under collusion receiving a lower pay-off than *honest* behaviour.

Simulator $\mathcal{S}$.

- The simulator $\mathcal{S}$ interacts with the functionality $\mathcal{F}_{\texttt{bridge}}$, and the corrupted bridge and whistle-blower nodes. The adversary can corrupt up to a maximum of f bridge nodes and f' whistle-blower nodes. The simulator has the set of bridge nodes and whistle-blower nodes that the adversary has corrupted. It has access to the function $Signature$ with methods `sign` and `verify`.
- During key generation, the simulator does the following: For $i \in C_S, j \in C_W$, obtain the public keys of the corrupted parties and forward $(\texttt{keygen}, sid, S, W, C_S, C_W, \{pk_i\}_{i \in C_S}, \{wpk_j\}_{j \in C_W})$ to the functionality.
- To forward the deposit amounts, for $i \in C_S$ forward the value S to the functionality by sending the $(\texttt{deposit}, S, i, pk_i, sid)$ message to the functionality.
- To handle the requests, upon receiving $\langle \texttt{val}, rid, t, sid \rangle$ from the functionality, for each party $i \in C_S$, forwards the partial signature generated through

$\sigma_i = Signature(\langle \texttt{val}, rid, t, sid \rangle, sk_i)$ as $(sig, \sigma_i, pk_i, i, \texttt{val}, rid, sid)$ to the functionality.

- The simulator can also raise complaints on behalf of the corrupted whistle-blower nodes by forwarding the message $(\texttt{complaint}, s, rid, sid)$

The simulator controls the view of the corrupted parties. The simulator forwards the public keys of the corrupted parties to the functionality, which upon receiving the these, populates the key pairs along with the key pairs of the honest parties. The simulator also forwards the initial deposit amount of behalf of the corrupted parties to the functionality. The simulator also forwards the signatures of the requests obtained from the corrupted parties to the functionalities. It can also raise complaints on behalf of the corrupted parties. The interacting corrupted parties have a consistent view in the real and ideal world since they forward deposits and the signatures on the received requests to the simulator just as in the real world. □

APPENDIX F
PERIODIC REWARD FOR WHISTLE-BLOWERS

As we mentioned in the introduction, collusion among the bridge nodes is a rare event. Therefore, periodic reward is necessary to mitigate the computational cost incurred by the whistle-blower nodes. While this periodic reward motivates them, it may also make them 'lazy', i.e. collect the reward without verifying the transactions. Thus, the whistle-blower nodes must submit proof that they were actively monitoring the system to get rewarded. In this section, we are going to describe the periodic reward mechanism.

At a high level, the reward mechanism will randomly challenge a whistle-blower node to provide proof that they were actively monitoring for a random interval in the past. To submit the proof, they have to post a bridge request in the $\mathcal{SC}$ with the hash of the pairs in the interval. The bridge nodes can efficiently verify the proof's validity since they have access to the pairs of requests and responses. If the proof is valid the bridge nodes sign off the transactions and the whistle-blower node is rewarded R_p from a smart contract in the $\mathcal{DC}$. Note that if the bridge nodes collude and refuse to sign off the transaction the honest whistle-blower can raise a complaint to the CRN network.

To avoid overloading the system with periodic reward challenges, instead of checking every epoch, let $\{\mathtt{e}_1, \mathtt{e}_2 \ldots\}$ be a set of randomly selected epochs, or checkpoints, where the reward mechanism runs. Let $\bar{\mathtt{e}}$ be the expected distance between checkpoints. At a checkpoint $\mathtt{e}_i$ an active whistle-bloweris selected uniformly at random to provide a proof for the interval $[r_{\mathtt{e}_i}, \mathtt{e}_i - 1]$ in order to get reward R_p, where $r_{\mathtt{e}_i}$ is selected at random. The selected whistle-blower node must post the proof as a bridge request in the next epoch. To bypass the issue of whistle-blowernodes only checking the interval they have been requested the endpoint $r_{\mathtt{e}_i}$ is selected at random hence the whistle-blower cannot be prepared in advance. We also require that $r_{\mathtt{e}_i} < \mathtt{e}_{i-1}$ in order to guarantee that all epochs until the checkpoint $\mathtt{e}_i$ have been verified at least once. Furthermore, the interval $[r_{\mathtt{e}_i}, \mathtt{e}_i - 1]$ is large enough that retroactively checking cannot be done between in time to be posted in the next epoch.

The parameters of the mechanism that we need to decide on are $\bar{\mathtt{e}}$, R_p, and $r_{\mathtt{e}_i}$. In Section V-C, we showed that the expected periodic reward per round (r_p) must satisfy the following, $r_p > c_{wb}$, therefore, we have that the expected reward per round $r_p = \frac{R_p}{m \cdot \bar{\mathtt{e}}} > c_{wb}$.

Notice, that a whistle-blower node can copy the solution from another whistle-blowerin exchange for part of the reward. It is important to point out that even in this scenario, the system is secure since at least one active whistle-blower is honestly monitoring the bridges. However, it is in the best interest of any whistle-blower node to monitor the bridge nodes since the whistle-blower nodes enter and exit the system freely. This makes coordinating with the rest of the nodes and finding another node that can provide the random interval requested much harder. Finally, to disincentivize copying the proofs we can increase the number of random intervals requested per checkpoint.

APPENDIX G
DISCUSSION

Honest simple majority and honest super majority. We observe that in almost all of the existing bridge networks a super majority threshold is employed, and the assumption there is that there exists at least super majority of honest nodes in the bridge network. We feel that it is an inspiration from *Byzantine Agreement* and *Byzantine Fault Tolerant State Machine Replication (BFT SMR)* protocols.

We remark that this need not be the case, mainly because the bridge requests are already ordered owing to the execution of BFT SMR protocol on the source chain. These bridge requests also have a unique place on the source blockchain – typically the block number and the transaction index inside the block. When we have this unique placement of the requests on the source chain, a Byzantine Agreement or a BFT SMR protocol is an overkill. We observe that there is no possibility of equivocation in the bridge network. All that is required is a confirmation or an endorsement by a simple majority threshold set of nodes in the bridge. The bridge nodes just relay the requests on the source chain to the destination chain while preserving the unique indexing information. The bridge smart contracts on the destination chain then needs to receive such relay messages only from a simple majority of bridge nodes, before taking the corresponding action on the destination chain. The simple majority requirement then guarantees inclusion of one honest node's endorsement (in the honest-malicious model), which is sufficient to take the required response on the destination chain.

By doing away with the roles of proposers and aggregators, the bridge protocol yields to a simpler game-theoretic analysis, as presented in Section V.

Threshold signature. For the purposes of paying rewards for good behaviour and penalizing for bad behaviour we need

accountability of bridge nodes. As the standard designs of threshold signature cryptography do not provide accountability we do not use it. Also, we do not have any aggregators in the bridge protocol itself. The signatures are individually verified on the destination chain until the simple majority threshold is reached.

An argument can be made that this approach of validating individual signature is not scalable. Consider the alternative. The alternate solutions typically include a role of an aggregator in the protocol. Here the aggregator receives the signed endorsements of bridge requests from the bridge nodes, aggregates them into a multi-sig along with a bit-map or a threshold signature, and submits to the destination chain as one single transaction. As mentioned above, owing to accountability, we need to go with a multi-sig approach. The validation on the destination chain smart contract would typically involve aggregating the public keys (using the bit-map) and then validating the aggregate multi-sig with this aggregated public key.

We reckon that this alternate approach is expensive in terms of gas costs as the public key aggregation and aggregated signature validation is happening on the destination blockchain virtual machine. In HyperLoop protocol the signature validation happens as part of the transaction validation outside of the blockchain virtual machine, and is hence cheaper.

This alternate approach also yields to higher latency as we now have an interactive protocol with aggregators in the play.

Use-cases and architecture. Hyperloop is essentially a cross-chain message-passing protocol. Many kinds of cross-chain applications may be built on top of this message-passing layer. These applications include:

- cross-chain smart contract function calls,
- lock-and-mint wrapped asset bridging,
- bridging with liquidity pools,
- cross-chain DeFi apps like lending, derivatives etc.

A. Criterion for batching

Criterion: We use three configurable parameters for batching the request transactions:

`timeLimit` We batch requests for the duration set by `timeLimit`. This avoids waiting unnecessarily for inordinately long times.

`valueLimit` We batch the bridge requests up to or immediately greater than the value of `valueLimit`. If the sum of the values of the requests is greater than or equal to the `valueLimit` then those requests are batched and forwarded. If a request is of value greater than `valueLimit` then it is still admitted as long as the total set of requests satisfy the sliding window limit.

`numLimit` We cap the number of bridge requests to be batched by `numLimit`.

APPENDIX H
AVOIDING CRN IN THE SYSTEM

If we can assume that the events of the source and destination chains can be verified on each other, then one may attempt to remove a trusted CRN from the system. This information transfer may be achieved by protocols like checking the source chain validator multisig on the destination chain and vice versa. However, this capability is difficult to assume among different chains especially if chains like Bitcoin are involved. In the case when it is possible, the whistle-blower nodes can raise complaints at both the chains and pause the bridge. Upon resolution, they can transfer the result to the other chain while proving that the complaint has been made and the bridge is paused. The analysis of this approach is beyond the scope of this paper.