Zeko: Fractal scaling of ZK applications using a Shared Sequencer L2 Stack

Robert Kornacki
rob@milkomeda.com

Nicolas Arqueros
nico@milkomeda.com

Sebastien Guillemot
seba@milkomeda.com

Brandon Kase
bkase@o1labs.org

Florian Kluge
florian.kluge@o1labs.org

October 5, 2023

1 Introduction

With the latest innovations in the blockchain space pushing towards a rollup-dominant future, Mina has found itself to be positioned in an ideal place to capitalize on all of its upfront ZK work, but now in a new direction. At this point in time ZK Rollups have reached significant mind-share in the blockchain industry as the ideal scaling solution, yet due to the inherent complexity at play, are up till now a nascent and burgeoning field with no clear winner.

A ZK Rollup on top of Mina unlocks the best of what Mina's model has to offer while maintaining a competitive edge with the likes of Ethereum and other leading chains with growing L2 ecosystems. Furthermore, this project seeks to strengthen Mina's strong points by fulfilling the following goals:

- Increasing the throughput of Mina L1
- Unlocking new possibilities for dApps by offering a DA Layer as a part of the L2
- Implementing reusable rollup architecture that can be expanded upon by future innovative Rollups (both onto Mina and in the future to Ethereum as well)
- Improve UX by supporting faster block times at the L2 level

2 Motivation

Mina Protocol is a layer one blockchain that aims to be the privacy and security layer for web3 through utilization of zero knowledge proofs. Mina itself is powered by a multi-tiered recursive zkSNARK proof that in a small, constant, size (succinctness) stands in for the full blockchain. Developers write smart contracts on top of Mina by tapping into this proof layer: The zkApps protocol extends the potential of zero-knowledge cryptography by enabling the following characteristics while preserving the succinctness of Mina:

- **General programmability** - able to execute and settle arbitrary programs, not constrained to a particular VM model, but instead is flexibly designed to support any number of execution models, VM-like or otherwise
- **Programmable privacy** - the privacy of inputs to smart contracts and their state can be programmed by their developers
- **Constant verification time** - individual transactions are executed, or more accurately "proven", asynchronously off-chain and verified on-chain in time proportional to the account updates, independent of computational complexity of any individual proof inside of each account update.

Mina's recursive layer, Pickles, on top of its proof system, Kimchi, supports arbitrary infinite recursion with no trusted setup. This is the key to unlocking an isomorphic Mina L2 ZK Rollup as this paper describes. Moreover, recursion allows private computation to be broken up into pieces, even ones that

can be completed by different people at different times who don't trust one another. Mina aims to be a very decentralized system, as all L1s should, but this, coupled with heavy reliance on ZK infrastructure, means that throughput and finality times are not desirable for many kinds of user interactions.

An L2, such as this one, can tap into the decentralized base layer of Mina, but dramatically improve throughput and perceived settlement time for user interactions. Since Mina itself is almost like a ZK Rollup-based validium for any base-chain it bridges to, many parts of Mina's infrastructure can be reused to create a ZK Rollup on itself. Moreover, this design is structured in such a way that each of the core components can be reused in other contexts.

The L2 system presented here actually describes a system of replaceable modules that can create highly configurable ZK Rollups, similarly to the OP Stack. In the same way that the OP Stack enables builders to lift any MIPS logic into an L2 blockchain (like Minecraft, Gameboy, Base, Optimism), this system enables lifting any arbitrary state machine powered by Pickles-wrapped Kimchi-compatible proofs into an L2 blockchain. This L2-stack by-default settles to Mina rather than Ethereum and by way of a ZK Rollup. This gives us a more flexible environment for applications with different sorts of trade-offs improving performance, for example, when compared to a Mina L1 application. Eventually, the proof-system backend can be hot-swapped to kzg-bn254 Kimchi proofs and the ZK stack can be configured to settle to Ethereum directly as well.

Similar to the OP Stack, it is made up of a stack of modular components that can be swapped in-and-out by other projects, but core pieces can be reused at will. Not only are core pieces like the DA layer useful for other zkApp builders, but building a custom L2 can involve reusing the DA layer, prover layer, and even sharing the same sequencer set across ZK L2s similar to Interchain Security in Cosmos and Parachains in Polkadot. All that's needed is to describe custom logic.

Compositional properties are innumerable due to Pickles/Kimchi's recursion capabilities. Proofs on these custom L2s and even *the full L2 state themselves* similar to Mina's full blockchain can be folded inside a Mina (L1 or L2) zkApp as well.

3 Requirements

In order to fulfill the above stated goals in the Introduction, this ZK Rollup project intends to adhere to the following requirements:

1. Minimize missing rollup blocks: Mina has a very long block time, and so missing a rollup block (ex: no sequencer elected for that slot) has a large impact on the L2 throughput
2. Minimize competing rollup blocks per slot: the cost of generating a proof for a rollup block is high. Therefore, we want to avoid cases where there are 2 sequencers elected for the same block which would double the proof generation cost
3. No new cryptography: the rollup should be implementable, secure and achieve reasonable performance under the existing cryptography available in Mina
4. Secure under partial synchrony: Ouroboros is a partially synchronous protocol and so our model needs to be secure under these assumptions
5. Faster blocks: the L2 has to support a faster block time than the L1. This means multiple L2 blocks need to be compressible into a single L1 rollup block submission
6. Composability: the rollup should, at least, achieve composability between different L2 deployments. Composability with the L1 is ideal, but not required. If composability with the L1 is not achieved, there should some mechanism to leverage any Mina L1 specific bridges (for EVM or otherwise) for the L2 format
7. Data Layer Reuse: The data layer solution should be reusable by L1 apps or L3 apps
8. Native zkApp execution environment: Developers should be able to reuse zkApp native functionalities and tooling, as both Mina and the rollup should have equivalent execution environments and not break native smart contracts.

4 Protocol Summary

Unlike existing rollup implementations which are typically deployed on L1's with on-chain storage as a fee-based native primitive, our rollup requires a novel design work in order to run synergistically with Mina (L1) and take into account its storage model. Due to the limited L1 on-chain storage capacity, this requires relying more deeply on the DA (Data Availability) layer for propagation of nearly all data, in addition to new mechanisms to ensure that liveness and security is maintained for both the state of the L2 itself, and the L1→L2/L2→L1 bridges.

In this section we will briefly touch upon the full cycle of L1→L2 bridging, submitting L2 transactions, bridging back from L2→L1, and finalizing a L2 batch end-to-end. The goal here is to provide a high level overview of how the different parts of the system interact together to function as a full-fledged ZK Rollup, in order to provide the reader with an intuition for the following sections.

4.1 User Submits L1→L2 Bridge Transfer Request (Deposit Request)

Users on Mina L1 have the ability to move their funds onto the L2 by submitting a bridge transfer request to our bridge smart contract. When submitting this request they specify the amount/what address they are sending the funds to on the L2.

The bridge smart contract emits an event which reveals the inputs they submitted with their transaction.

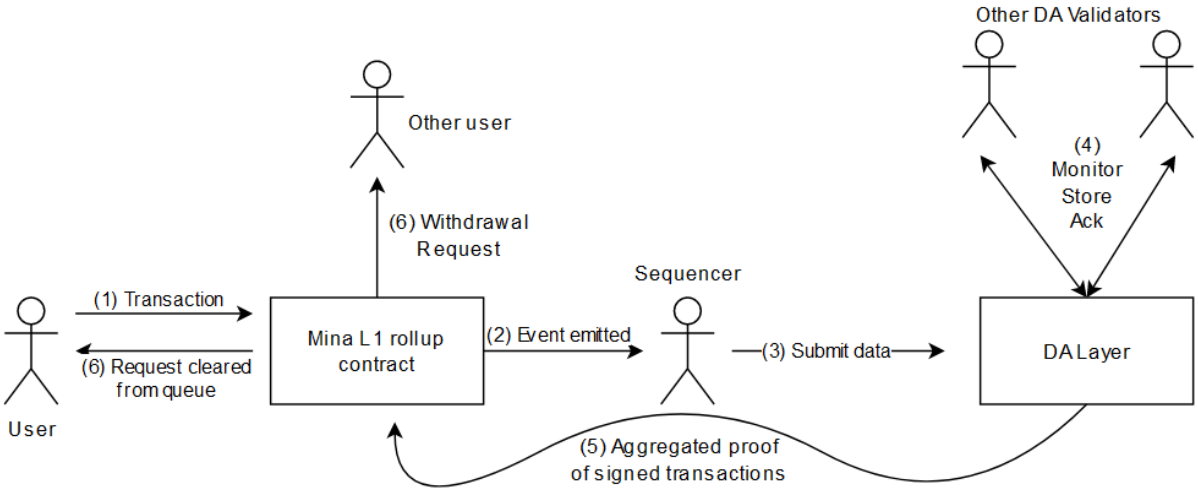

Figure 1: Successful L1 → L2 transaction

4.2 Transfer Request Relayed To DA Layer

The sequencers of the L2 track all of the events emitted by the bridge smart contract, verify that the inputs emitted in the event are correct, and then proceed forward with relaying the transfer request to the DA layer (only a single sequencer needs to submit a given request, and users will be able to re-emit events from previous requests from their wallet in case of events being missed).

The data of the event is posted in a "bridge transfer request pool" smart contract on the DA layer together with a proof of the transaction being included in a Mina L1 block. As this is the DA layer, this pool in and of itself contains no finalization logic, but can be instead understood as a persistent pool for all potential L1→L2 transfers. All sequencers refer to this pool for finding all historical transfer requests (enabling bootstrapping/resyncing of new sequencers).

4.3 Sequencer Generates Queue Of Pending Transfer Requests

From the pool smart contract on the DA layer, each sequencer locally generates a queue of pending (to be applied) transfer requests. This is done via a process where the sequencer locally verifies each of the proofs attached to the transfer requests, and cross-references which ones have already been committed

in previous finalized batches by checking the receipts posted to the DA layer (from the previous batches, more on this later).

This local generation of the transfer request list, rather than attempting to perform some form of consensus on the DA layer, ensures that all sequencers will be always able to maintain an up-to-date view no matter if rollbacks take place on Mina L1.

4.4 Sequencer Creates L2 Transactions Which Fulfill Bridge Transfer Requests

From the list of pending transfer requests, the sequencer (if they are selected to be the batch producer at the given block height) selects one or more and adds them into his/her batch by aggregating the proofs of all of the transactions.

Furthermore, in this process adding a transaction which fulfills a bridge transfer request will also require the sequencer to produce a "L1→L2 transfer receipt". This is a piece of data attached to the rest of the batch which specifies which transfer requests are included (receipts enable the L1 bridge smart contract to check for coherence of bridge transfers before finalizing a batch).

4.5 Sequencer Includes Other Transactions In Batch

Sequencers have an HTTP API for users of the L2 to submit transactions directly, in order for them to be included in an upcoming batch. The sequencer adds these L2 transactions together into the same batch that they were creating for the bridge transfer requests.

In the future when the system upgrades from single to multi sequencer, the DA layer will be used to also act as a mempool for L2 transactions. Users will be able to either directly submit their L2 transaction, or sequencers will be able to push extra transaction submitted by users which they do not have capacity to include in their batch.

4.6 Users Submit L2→L1 Transfer Requests (Withdrawal Request)

Users on the L2 can likewise submit transactions to the rollup bridge smart contract on the L2. In other words, users can create L2→L1 bridge transfer requests.

These transfer requests are reasonably similar to the L1→L2 variants in regards to inputs to the smart contract, however due to the fact that this is done on the L2 back to the L1, we do not require relaying these transfer requests to the DA layer pool before including them in a batch.

Instead, because the L2 uses the L1 for sequencing (and the sequencer is guaranteed to be online when the transfer requests are committed to the batch, meaning the events are guaranteed available to the sequencer), the protocol can enforce that the sequencer must include all L2→L1 bridge transfer requests in their batch.

As such, the sequencer produces a L2→L1 transfer receipt for each transaction that submits a bridge transfer request in the batch.

4.7 Sequencer Posts Batch + Receipts To The DA Layer

Once the sequencer has fully finished building the batch (by including bridge transfer requests in both directions, aggregating proofs, and generating all receipts), then he/she is required to post the batch to the DA layer and have consensus reached on its availability.

Of note, the posting of receipts onto the DA layer also allows sequencers to more easily keep track of which L1→L2 transfer requests have been fulfilled (if the batch posted in fact gets accepted in the rollup smart contract on Mina L1 and finalized as a part of L2 history). This thus allows sequencers to update their local pending transfer request queue.

Once consensus has been reached on the DA, then the sequencer can finally post the batch to Mina L1 to finalize it together with the DA consensus proof.

4.8 Finalizing The Batch On Mina L1

When the batch is posted to the rollup smart contract on Mina L1, the following takes place:

1. The DA consensus is checked to ensure the batch has been made available.
2. The aggregated proof of all transactions of the batch is verified.
3. The L1→L2 receipts are verified to correspond to existing pending bridge transfer requests in the L1→L2 bridge smart contract (and those requests are thus moved from pending to confirmed in the contract).
4. The L2→L1 receipts are verified to match all of the transactions in the L2 which submitted bridge transfer requests.

If all of the above validates properly, then the batch is considered valid and becomes a part of the accepted L2 history (to be clear, if Mina L1 rolls back, then the L2 will roll back with it maintaining bridge security). If the batch is accepted, then all of the L2→L1 receipts are also fulfilled with users on Mina L1 receiving tokens from the bridge smart contract corresponding to the receipts.

4.9 ZK Rollup State Transition

As such, the act of transitioning our ZK Rollup from state A→B requires:

1. Interacting with the DA layer twice (once for the bridge transfer request pool, once for DA consensus of the batch)
2. Selecting and aggregating a series of bridge transfer requests and their receipts (for both L1→L2 and L2→L1)
3. Aggregating user-submitted L2 transactions
4. Finalizing the batch by posting it + the DA consensus to Mina L1, thereby officially processing all included bridge transfer requests.

By putting all of this together, we are able to build a functioning ZK Rollup on top of a L1 which does not natively provide arbitrary-sized on-chain storage as a native primitive.

4.10 Supporting Faster L2 Blocks

Going further, it is also possible to move past the fixed ratio of one batch representing one L2 block. Instead we can consider a batch to be made of a fixed ratio of blocks per batch (ex. 10), and thus enforce that every L2 batch posting moves L2 state +10 blocks forward (note this does not mean that every L1 block necessarily will include an L2 batch posted, meaning the L2 will not constantly progress forward at a 10x pace compared to the L1).

The L2 sequencer will have the ability to commit/aggregate L2 transactions as they come in to the current block in the batch, and will move to building the next block based off of his/her local clock (it is possible for no transactions to be submitted by users, and thus some blocks may be created empty). From the user perspective, this provides a significantly faster experience when interacting with the L2 compared to Mina L1, and provides dApps more open design space to work with.

Of note, when the sequencer aggregates the proofs for a specific block part of a batch, this is only a "soft-finalization". From the perspective of the ZK Rollup contract on Mina L1, this block still doesn't exist in the official L2 history. Only when the full batch has been posted to the rollup contract and each block within verified to be valid, is the batch (and thus all blocks within) actually added to the L2 official history (though the L1 itself may rollback, meaning for users technically they must wait for the L1 block to finalize before assuming the L2 has as well).

5 Data Availability (DA) Layer

Data Availability guarantees that a piece of data has been broadcast (either whole or in parts) to a sufficient number of participants in a protocol such that the data (including historic data) is recoverable by all. DA is a vital part of all blockchains, but this is multiplied many-fold in the context of deploying

a rollup on top of an L1 with very limited on-chain storage available. In other words, a DA layer is a requirement for a ZK Rollup to be deployed on top of Mina.

As we will touch in greater depth in future sections, this DA layer is not only used for the posting of batches, but also serves a key role in storing L1→L2 bridge transfer requests. Additionally, the DA layer will also be used in the future for any new features/functionality which require multiple actors to coordinate together before acting on the L1 or the L2 (ex. acting as a mempool for L2 transactions in a multi-sequencer setup).

Of note, the DA layer performs no sequencing/finalization of L2 batches. It is entirely possible for batches to be posted to the DA layer by a sequencer, which either are explicitly invalid at posting (from the perspective of the L1 rollup smart contract/official L2 history) or which become invalid after Mina L1 goes through a rollback.

5.1 DA Layer Option Analysis

5.1.1 Cryptography background

Data availability proofs generally leverage KZG commitments to provide data availability. This is because they are the most efficient solution not only for single queries, but also support aggregating multiple proofs in a single query of constant size. KZG commitments are generally implemented over BLS curves, which are not available to Mina smart contracts.

There exists data availability constructions that only require IPA (inner-product argument) or FRI (hash function) assumptions which can be implemented in Mina contracts, but they are not explored in-depth compared to the well studied KZG approach, and state-of-the-art performs worse both on proof size as well as verification time.

5.1.2 Arweave or IPFS

These solutions are not designed to support data availability sampling over large sets of data and therefore are not well suited for our use-case.

5.1.3 Celestia

Celestia does not have Poseidon hash/pasta curves (uses SHA2 and SHA3 / secp256k1) and cannot implement them as it has no execution layer, and thus no smart contracts either. It also does not support KZG, instead opting for 2D Reed-Solomon based fraud proofs with the intention of adding KZG support in the future. It uses the Cosmos SDK so it can be modified. Additionally, it uses Tendermint, which is advantageous due to it being fork-free one a block is finalized.

5.1.4 Polygon Avail

Avail is a platform built on Polkadot Substrate, and although it does not have Poseidon Hash/Pasta Curves, it does have support for Ed25519. This means that it may be usable with a performance hit once the Ed25519 precompile is live on Mina. Furthermore it also has full KZG support. It uses GRANDPA from Polkadot, which may have forks.

Unfortunately, it appears that Polygon Avail is on pause. They were working with the Nomad team on the implementation and development stopped after the Nomad hack.

5.1.5 Chia

Chia does not support Poseidon hash or pasta curves, but instead uses BLS12-381 and SHA256 hashing algorithms. It also provides programmability with its smart contract-enabled Chialisp language. Additionally, it has its own DataLayer solution, which has many of the same functionalities we would need, such as pub-sub for passing data around.

Unfortunately, Chia lacks a few essential features that would be necessary for our use case — namely KZG, deterministic finality, trustless proof of finality to external chains, and it also has a slow block time of 52 seconds. However, Chia is also working on a state channel solution, which would allow us to bypass the long finality and long block times by having a dedicated state channel for Mina state proofs. Having

a dynamic validator set (closing and reopening the state channel) would likely require building a Mina light client smart contract inside the Chia blockchain. The Chia team have shown potential interest in making the changes needed to unblock us if we decide to go this direction.

5.1.6 Ethereum

Ethereum has plans to implement KZG in the coming future in addition to adding support for pasta curves, so it may be possible to use as a DA layer eventually. However, at present this is not available, and given previous road map timelines of Ethereum, it is unlikely a wise decision to plan for these updates releasing in the near future. Ethereum support would also be a prerequisite for Eigenlayer DA support as EigenLayer requires specifying slashing conditions for malicious behavior in Solidity which can only be done if Ethereum supports the required cryptogragphy.

5.1.7 Algorand

Algorand does not have Poseidon hash or pasta curves, but has a smart contract language (TEAL) which enables implementing the needed requirements. Its inclusion of Curve25519, secp256k1, BLS12-381, BN254 and Falcon Signatures shows they are willing to also include various primitives at the VM level as well.

Algorand has a 3s block time with instant finality together with threshold signature checkpointing. Algorand also includes a storage layer which allows contracts to save theoretically up to terabytes of data if they are willing to pay the price, though it does not have KZG support.

The Algorand team are willing to potentially help out with making changes necessary to support new primitives we require, though it is not geared as an explicit DA layer at this point in time.

5.1.8 Polkadot

Currently Polkadot requires projects to be a parachain in order to get access to its DA layer. They are implementing an on-demand space feature where you can pay for space allocation - although this is not released yet. Available primitives are: curve25519, ristretto25519, secp256k1 and blake2b. They are not too open to add newer crypto primitives at the VM layer without showing adoption by implementing them as a WASM smart contract.

5.2 Initial DA Layer Implementation

The primary goal of the initial implementation of the DA layer is to build something that works in the immediate future that suffices our core requirements, while leaving the door open to future improvements towards a truly robust & decentralized DA solution.

This initial implementation will be built as an Etheremint chain (Cosmos), thereby providing us with instant finality, fast blocktime (1-2s), and access to the EVM. The EVM speeds up development time by allowing us to reuse previously written smart contracts from Milkomeda to support the coordination required for the DA of L2 batches.

The initial DA protocol will simply require a majority of validators to sign off on the batch data (using their Mina keypair). The validators of both the chain and the DA smart contract will be a predetermined set comprised of the Mina Foundation/O(1) Labs/Milkomeda. Once the signatures are aggregated, they can be carried over to the rollup smart contract on Mina L1 by the sequencer as an attestation of DA for the batch they are posting.

Of note, updates to the set of validators necessitates updating them on both Mina L1 and the DA layer chain. This can be coordinated purely on Mina L1 and bridged over trustlessly (using the Nil Foundation bridge) to update the list in the DA layer EVM smart contract.

Lastly, this DA layer will use the L1 token as its native asset (in other words for gas/transaction fees). Using the Nil Foundation bridge, we will support 1-way bridging of the rollup project token from Mina L1 to the DA layer chain for anyone (eventually other projects or users) but starting with the sequencer(s) to pay for transactions & attestation fees on the DA layer.

Thanks to the fact that the DA layer chain will be based on Ethermint, it also has access to the Cosmos ecosystem's IBC messaging protocol. This will make it easier for DA validators to redeem

the profit they made from fees and cash them out (more important in the future when the DA layer decentralizes). This path offers greater simplicity compared to building a trusted 2-way bridge between Mina L1 and the DA layer

5.3 Upgrading To A Long-Term Solution

The initial DA layer implementation will be sufficient in the short term for bootstrapping the rollup, however in the medium-long term we have one of two options:

1. Improve the Ethermint chain by implementing DA sampling and building it out into a full fledged DA solution for the Mina ecosystem as a whole.
2. Integrating with an established chain for DA.

Based off of our initial implementation the first option offers us the ability to reuse much of the same infrastructure, however introduces significant engineering complexity. No chain live on mainnet to date has currently implemented DA sampling (Celestia is slated to launch this year), and additionally adding it on top of Ethermint specifically will likely run into several integration problems (as it was not designed with this in mind).

For example, the dynamics of how light nodes performing DA sampling interact with the standard block production model of the EVM, among several other issues, will likely accumulate into becoming a large-sized project of its own to make everything working seamlessly. As other teams are already working on solving the DA problem themselves, it generally makes more sense to reuse their solutions rather than building everything from scratch.

In this view, Celestia is currently the DA project that is closest to mainnet launch, and as such would be a good target for integrating with our rollup. However as mentioned previously, Celestia does not have support for Poseidon hash / pasta curves nor does it support KZG commitments at the moment.

This unfortunate lack of direct compatibility is not addressable by simply writing a smart contract to add support either, as Celestia by design has no execution and thus no smart contracts. Though native integration into Celestia is an appealing option, it is not directly viable in the short term.

That being the case, one potential mid-term option is to use Celestia as a DA layer, but reuse our set of L2 sequencers to act as validators which bridge the DA attestation themselves. In other words, we can deploy an EVM sovereign rollup on top of Celestia L1 which reuses all of the same architecture we initially implemented on top of our Ethermint chain, but now with a solid DA layer underneath all of the data.

This acts as a solid step forward in robustness, as the chance of data being lost compared to a custom Ethermint deployment is significantly minimized. However the one tradeoff to note with this approach is that our set of L2 sequencers will be security bearing for attesting to the validity of the DA. As such this approach is more robust against adversaries who would try to delete/corrupt data held by the small set of Ethermint DA layer validators, but does not improve safety against attacks where a rogue majority of sequencers try to falsely assert that DA has been made for a batch that was never posted to the DA layer.

Other risks associated with a potential deployment on top of Celestia include: delays in the launch of Celestia mainnet would postpone the launch of our L2; unknowns related to Celestia's tooling which may be lacking or buggy on launch; Celestia has yet to prove itself with a steady track record of months/years of stability and reliability under its belt.

Over the long-term as DA layers mature within the wider blockchain space, the landscape for which solution will be best to settle on for our ZK Rollup will become clearer. At this point due to the lack of any options being production ready/released, the focus will be primarily on our initial implementation while building connections with all up-and-coming DA solutions. You can read more about potential long-term R&D options in section 12.1

6 L1→L2 Bridge (Deposit)

When designing the L1→L2 Bridge, there is a key requirement which must be taken into account; we cannot enforce that a new L2 batch which gets posted will include every pending L1→L2 bridge transfer

request. This is the case for two main reasons:

- Mina L1 events are temporal/possible to miss (ex. all L2 sequencer are offline when a transfer request is submitted on the L1)
- It is possible for a pile-up of bridge transfer requests to accrue (ex. if sequencers are offline for a prolonged time) which may be too large to process in a single batch (either on the DA layer due to transaction size, or on Mina L1 due to circuit size)

In the prior case, someone could potentially perform a DoS attack on the sequencers forcing them offline for a short period of time (or due to wide server outages) and submit a bridge transfer request which has its event (with the clear text inputs of the transfer request) completely missed by all sequencers. Without care, this would cause the L2 to lock, due to the lack of data availability of the transfer request, leaving it to be impossible to fulfill.

In the latter case (which is less likely, but still a valid edge case to be wary of), the L2 could arrive at a state which would be impossible to transition from because of the large number of pending bridge transfer requests. This is distinct from the prior case due to the fact that sequencers could in fact have the data available for processing all requests, but the limiting factor being the circuit size on the L1 preventing the batch from actually being finalized (with no alternate state transition available, this would keep the L2 from making progressing forward).

Due to these restrictions, the L1→L2 bridge must be designed as a partially synchronous protocol which does not guarantee instant inclusion of transfer requests, but instead ensures that all transfer requests will eventually be fulfilled or refunded.

As briefly touched upon in the Protocol Summary section, this is accomplished by:

- Maintaining a bridge transfer request pool in the DA layer
- Allowing each sequencer to build their own local pending transfer queue based off of the pool + L1 state
- Recording the receipts of the L1→L2 transfer requests together with the batch data on the DA layer before finalizing the batch on the L1

6.1 Bridge Transfer Requests

In regards to the L1→L2 bridge transfer requests submitted to the pool, there are two approaches we can take, each with differing trade-offs:

1. Assume that any transfer request submitted will eventually be included by a sequencer in an upcoming batch.
2. Implement a timeout sub-protocol which returns funds back to users if their transfer request was not included within X hours (measured in blocks).

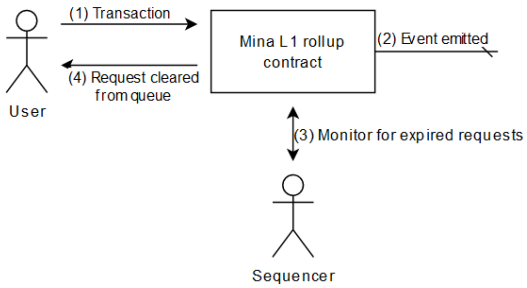

Figure 2: Failed L1 → L2 transaction

The first option simplifies the initial implementation by requiring no state-change to take place in the bridge smart contract after funds have been deposited by a user. Furthermore valid requests which have been relayed into the pool in the DA layer are also guaranteed to be available to be included in upcoming batches if they haven't yet been fulfilled.

The major negative of this approach is that if for some reason sequencers are offline, the transfer request never gets relayed to the DA layer (due to software error or malicious action), or the only online sequencers are malicious, then the user may end up with their funds stuck with no recourse (users will also be able to re-submit their transfer request inputs to the sequencers via the bridge zkApp frontend, however this is merely a good stop-gap, not a perfect solution).

As such, especially when entering into a future with a multi-sequencer setup with open membership, it will be important to implement a timeout protocol which specifies that any L1→L2 bridge transfer request is only able to be fulfilled if it is included in a new batch within X hours (X being measured as a predefined number of blocks that is universal for all requests).

6.2 Local Pending Transfer Queue

Each sequencer builds their own local view/queue of which L1→L2 transfer requests in the pool are still pending. This is required due to two primary vectors:

1. Mina L1 can rollback, meaning previously "confirmed" bridge transfer requests can be reverted back into a pending state.
2. Individual sequencers can potentially go rogue and post fake (never included on Mina L1) or previously valid (requests submitted before a rollback) transfer requests which are not valid currently.

Because Mina L1 does not have deterministic finality, we cannot guarantee a 1-to-1 correspondence of submitted bridge transfer requests and bridge transfer requests posted in the DA layer pool. There is the possibility of rollbacks taking place which suddenly makes previously confirmed bridge transfer requests be put back into pending or potentially even invalid state (because the L1 was rolled back to before the bridge transfer request was posted at all).

As such we cannot treat the pool as a pure source of truth, meaning that sequencers must always reference Mina L1 to verify that said transfer requests were indeed posted/are currently pending. Of note when a sequencer posts a batch to the Mina L1 rollup contract, it will check the receipts to ensure that the bridge transfers are indeed valid, thereby preventing incoherent states from being reached (tokens being sent in the L2 without depositing them on the L1). Even though this check is in place to keep things secure, the sequencer still needs to know which requests are indeed valid when building their batch (so that they don't do wasteful work in aggregating proofs and submitting their batch to the DA layer just for it to get rejected because of invalid L1→L2 transfer requests).

This is why the local pending transfer queue is required.

6.3 L1→L2 Receipts

To reiterate, receipts are a piece of data which get bundled together with batches which specify which transfer requests were included inside of the batch. From the L1→L2 bridge perspective, receipts allow us to:

1. Ensure that each batch posted to the Mina L1 rollup contract only include valid (currently pending) L1→L2 bridge transfer requests.
2. Update the state of a bridge transfer request on the L1 to be explicitly fulfilled (required when request timeouts are implemented in the bridge).

As such, receipts guarantee that the L1→L2 bridge maintains coherency, while also ensuring that every bridge transfer request will terminate, either as fulfilled or redeemable. Redeemable requests will be processed by sequencers and not by users as allowing users to redeem directly would lead to problems arising from concurrent contract modification which would slowdown the rollup.

7 L2→L1 Bridge (Withdrawal)

Unlike the L1→L2 bridge, the L2→L1 direction is more straightforward mechanically. Interestingly enough, this is actually opposite of classical rollups in other ecosystems, where L1→L2 is trivial but the inverse is the actual hard part.

In our case much of the overhead is due to the requirement of having a partially synchronous L1→L2 bridge which relies on the DA layer to store relayed L1→L2 bridge transfer requests. Furthermore, because the set of Mina L1 validators is not equivalent to the set of L2 sequencers, there is the ability for the L1 to move forward without the L2 progressing. While benign at first glance, this introduces the major issue of users potentially being able to submit such a larger number of bridge transfer requests that the rollup smart contract circuit on Mina L1 cannot support processing all in one go.

In the L2→L1 context however, even if the sequencers go offline, the L2 does not continue forward and thus no new L2→L1 bridge transfer requests are possible to be committed. Because a sequencer has full freedom to include however many L2→L1 bridge transfer requests as they desire into the L2 batch, this means that the sequencer can guarantee a fixed number are included. In other words, we can enforce that every L2→L1 bridge transfer request submitted in a batch is instantly processed because the sequencers themselves will only produce batches that are within the required size constraints and thus be processable by the L1 rollup smart contract.

Thanks to this simplification, the L2→L1 bridge skips the whole process of relaying requests to the DA chain pool and building a local pending transfer queue.

7.1 Bridge Transfer Requests & Receipts

Users who wish to move from L2→L1 must simply submit an L2 transaction (to the sequencer) which moves funds into the bridge smart contract on the L2. This contract acts similarly to the bridge smart contract on Mina L1, wherein it emits an event with the clear text of the inputs the user provided.

Capitalizing on the fact the sequencer cannot miss the event emitted by the bridge smart contract (as they are the one building the batch and aggregating the proofs), the sequencer generates the L2→L1 receipts without interacting with the DA layer. All L2→L1 bridge transfer requests are forced to be executed upon inclusion, meaning that these requests differ compared to their L1→L2 counterparts in the fact that they have no pending or redeemable states in the bridge smart contract on the L2.

Once the batch is created, it is joined together with all receipts (L1→L2 & L2→L1) and posted to the DA layer. From here the previously outlined path of posting the batch + receipts + DA consensus to Mina L1 takes place.

All L2→L1 receipts are verified to match all included L2 transactions sent to the bridge smart contract on the L2, and if matching then tokens are sent from the bridge on the L1 to the corresponding addresses.

8 L2 (ZK Rollup) Node

The ZK Rollup L2 node will function similarly to an L1 Mina node, however with a few key differences.

Firstly, as this is an L2, all networking and consensus implemented in the L1 node are irrelevant for our use case. Rollups inherit the sequencing of the underlying L1 they are deployed on, and as such all batch data can be directly committed to the L1 at the transaction/smart contract level to progress the L2 state forward. Sequencers are not required to communicate off-chain to perform consensus prior to submitting a new batch to extend the L2 rollup history. (One note, the DA layer is technically used in our system as an off-chain communication channel (from the perspective of the L1) to share L1→L2 bridge transfer requests and receipts transparently with other sequencers.)

As consensus is outsourced to the L1, the L2 node requires using an indexer which watches both the L1 and the DA layer for when a new state transition (batch) is finalized on the L1. The rollup smart contract on the L1 will always ensure that the batch is validated before being finalized in the contract, and so from the L2 node perspective it can simply trust that when a new batch has been committed, it is indeed valid. In other words, the L2 node can apply any batch to it's local state if the L1 has accepted it in the rollup contract.

This reliance on L1 consensus also means that the L2 node must track the block heights of the L1 and watch for rollbacks. As is typical of proof-based rollups, if the L1 rolls back, so does the L2 and all bridge transfers both ways, which means that a bridge desync is never introduced. As such the L2 node/L1 indexer must be engineered to watch for forks/rollbacks in the L1 and be able to cleanly roll back to a previous state.

8.1 Supporting L1→L2 Bridge Transfers (Deposit)

Besides accounting for sequencing in the rollup model, the L2 node must also add a primitive to support L1→L2 transfers. This primitive can either be:

1. An extension of existing asset/token transfers but with a customized *from* field which denotes that this is a L1→L2 transfer and thus the assets are not required to exist previously in the L2 state in order for the transaction to be considered valid within the L2.
2. A new type of transaction which doesn't override the default asset transfer standard, but instead explicitly mints tokens out of thin air and sends them to a specified address.

In either case, only the sequencer of the batch is allowed to use this new primitive, and the L1 rollup smart contract will enforce that this is the case (and that valid receipts which match bridge transfer requests are included for each). Because the L1 and the L2 have an identical execution model, this also translates into identical token standards, thus allowing us to preserve the token ids.

Alternatively if adding such a primitive adds excessive implementation overhead, it is also possible to implement similar functionality with no new additions to the L2 node by having an explicit whitelist of assets allowed to be bridged to the L2 and minting a new token on the L2 which represents a 1-to-1 copy of the L1 token. These tokens would be locked in a smart contract that only allows the sequencer to withdraw funds from (and these would likewise be validated to match the receipts by the L1 rollup contract), thus achieving the same end goal however with a bit less flexibility and more steps overall.

9 Protocol Fees

In order for the ZK Rollup to be sustainable, it must have its own token as an integral part of the fee structure, both for the L2 itself and for the DA layer. These fees map onto work (computation), DA (networking + storage), and other real costs which underlie the functioning of the L2 itself.

At a high level, the fees in the system are based off of:

- DA storage fee (L1 token, bridged over 1-way from Mina L1)
- Batch creation & proof aggregation fee (L1 token)
- L1 batch posting fee (L1 token)

9.1 User-Facing Fees

The above listed fees represent where fees are required in the process of building and posting an L2 batch, however these do not precisely map onto the user experience. For example, most users will likely not interact with the DA layer directly, but will instead go through a sequencer who will do so in their place.

As such we will now look at the user-facing fees, which vary based on the complexity of the underlying mechanics taking place.

9.1.1 L2 Transaction Fee

Users submit proofs (transactions) to be included into an L2 batch, however the underlying batch creation and proof aggregation requires real work to be performed. Furthermore, not all proofs are equivalent depending on the complexity of the circuit, which translates to differing amounts of work required. This fee is thus proportional and charged in L1 tokens.

The table below shows some time examples for processing proofs for specific circuits. It is also worth noting that in Mina there is an upper cap to the complexity of circuits, thereby establishing a limit to the number of proofs that can be aggregated (not because the rollup block gets too big, but because the number of account & state updates that can be performed on the L1 is limited)

Small circuit (2^5)	10 [s]
Medium circuit (2^{15})	27 [s]
Large circuit ($2^{16} - 2^7$)	35 [s]

Values from a benchmark by Florian Kluge from $O(1)$ labs

9.1.2 L2→L1 Bridge Fee (Withdrawal Fee)

Due to the fact the L2→L1 bridge does not require using the DA layer or any coordination from other sequencers, this direction of bridging will be the cheaper of the two. These fees will be charged in L1 token.

9.1.3 L1→L2 Bridge Fee (Deposit Fee)

The highest user-facing fee will come from L1→L2 bridge transfers. The reason for this is that these bridge transfer requests will be relayed onto the DA layer to the request pool (with an L1 inclusion proof) before even being added into an L2 batch. Furthermore, sequencers must also maintain a local pending transfers queue (which adapts to both L1 rollbacks, and high loads), which all together have a real added cost.

These feels will be charged in L1 tokens

9.1.4 Sequencer Election Fees

Becoming a sequencer for the L2 requires purchasing and burning L2 tokens. The goal is that the value burned at the auction find equilibrium at approximately equal the revenue (both direct from L2 tx fees and indirect through MEV).

This auction system can also be reused to not just decide the sequencer for a single L2, but many L2s created from the same SDK (similar to *OP Stack*). These will be called *Kimchi Sequencers*

9.1.5 User Fee Analysis

Current Mina Mainnet transaction fees are in the range of 0.007 USD. As seen in the table above, the complexity of the circuit does not impact time by orders of magnitude in the processing of the proofs. As such, the expected price of an L2 transaction for users should be in the 3-6x range, around 0.03 - 0.04 USD (1-3x for computation, 1-2x storage, and 1x proportional fee to post on L1). In general terms, this translates to single-digit cent costs for a complex transaction.

Of note, this higher fee does not mean that the rollup is less scalable compared to the L1 (if one were to assume transaction fee were to map onto transaction throughput 1:1). It simply means that there are more base mechanics at play when producing a batch (such as providing DA for all dApps in the L2 baked-in, which is a major win), while also helping to scale Mina L1 further at the same time. In the event that demand for Mina blockspace increases leading to average fee increases, the L2 fees will start being cheaper as many L2 transactions are batched before writing back to the L1.

Compared to other L2 solutions in Ethereum such as Optimism, the pricing for using the Mina L2 Rollup is many times cheaper. Currently on Optimism, a DEX swap requires approximately 130,000 units of gas, which costs around 0.30 USD. We estimate that a similar DEX swap on our ZK Rollup should be within the single digit cent range (the assumption for this calculation, granted future crypto pricing cannot be predicted, is that the price of the L2 token will increase proportionally to Optimism). Thus, the fees are in line with being competitive in the wider rollup space.

10 Ecosystem Tooling Required

One of the implicit goals of the ZK Rollup is to boost activity and adoption of Mina through bolstering the development of zkApps and overall ecosystem growth. To make the road ahead easier for all parties involved, the plan is to leverage all current Mina tooling with the minimum number of changes possible and maintain near perfect compatibility with the L1 itself.

10.1 For Developers

10.1.1 L2 Indexer

Every ecosystem requires an indexer to extract information from on-chain state, which will then be used by other off-chain tooling. Our rollup will have the same interface as the L1 from the indexer's

perspective when interacting with the L2 node (despite relying on a DA layer and having differing architecture underneath), making it possible to maintain compatibility with the majority of the existing indexer tooling with the main caveat of potentially requiring an adapter to also support reading from the DA layer for accessing all batch data.

10.1.2 GraphQL

The resulting indexer for the L2 will behave the same way as for the L1, so from the perspective of GraphQL, there are no differences.

10.1.3 Rosetta

Rosetta, a tool created by Coinbase and used by multiple exchanges, will need to be expanded to be compatible with the L2 to eventually gain exchange support.

10.2 For Users

10.2.1 Explorer

Implementing the Explorer should not be difficult, although some specific features need to be added to provide better visibility and transparency on the progress of bridge transfers and L2 transactions. Specifically information related to relaying/posting data to the DA layer and the status of the batch are inherently different compared to the L1. For simplicity, some information may need to be hidden at the user/high level, but low-level developers should have full access to the exact status for full expressiveness in development.

10.2.2 Wallet integration

Users interact with DApps on a blockchain using their wallets, so it's important to make the integration with the L2 as easy as possible for the wallet developers. As previously specified, changes to indexers and other endpoints are minimal, so most of the work for wallets should be related to re-targeting their infra to use the "new chain" (using L2 indexer/node to fetch information about the L2 state) and potentially include the ability to bridge onto/off of the L2 directly in the wallet for a better user experience.

10.2.3 Mina L2 Bridge Explorer

The plan is to have a 'Bridge Explorer' where users can check the status of their bridge transfers. The main idea will be similar to what can be seen on https://bridge-explorer.milkomeda.com/algorand-mainnet, however branded to fit with the look and feel of the current Mina Explorer.

Functionalities

- View list of pending bridge transfer requests (for Mina, rollup token, and other tokens)
- Search for specific transaction IDs to check the status of the bridging
- General information about the bridge
- See details of specific bridge transactions
- (Optional) Show information about the batch created by the sequencer and its status.

11 Impact on Developers and Users

Emphasizing the significance of developer and user experience, both aspects emerge as vital elements for the effective functioning of any software or application, especially in blockchains. These two aspects complement each other, working in tandem to ensure the contentment and achievement of both developers and end users.

Expanding upon the objectives delineated in section 3, as well as the underlying drive of this project detailed in section 2, it becomes evident that enhancing developer and user experience is a pivotal requirement for success. However, it is essential to note that this improvement should not come at the expense of either experience. Fortunately, the proposed rollup architecture, in conjunction with the

properties and structure of Mina, presents a unique opportunity to meet this requirement. By enabling the utilization of the same execution environment employed on the L1, developers can leverage existing L1 ecosystem tools, smart contracts, and platforms. This provision empowers developers with familiarity and compatibility, ultimately augmenting their experience. To underscore this point, let's delve into some key objectives and how we plan to address them:

- **Ensuring User Satisfaction**: User experience holds paramount importance in ensuring user satisfaction. A well-crafted and user-friendly interface, seamless interactions, and a smooth user journey contribute to a positive user experience. When users find an application easy to navigate, visually appealing, and responsive, their satisfaction with the product increases. While user satisfaction is the ultimate goal for application developers, the underlying infrastructure, specifically the blockchain or rollup, plays a crucial role in achieving this. By facilitating higher throughput, shorter block times, and other enhancements, the rollup provides developers with a robust foundation to build cutting-edge applications upon.
- **Efficiency and Productivity**: A positive developer experience promotes efficiency and productivity. When developers have intuitive tools, well-documented resources, and a supportive development environment, they can focus more on building high-quality software. This leads to faster development cycles, reduced errors, and increased overall productivity.

By enabling a more efficient and performant environment, developers can focus on creating intuitive and visually pleasing interfaces, enhancing the overall user experience. This synergy between the infrastructure and application development empowers developers to meet user expectations and drive higher levels of satisfaction. Our aim is to create a resilient, robust, and efficient execution environment that equips developers with all the necessary tools to build sustainable applications and businesses that effectively meet the needs of users. Our primary focus is to provide these resources and capabilities without imposing any additional costs - all while settling to Mina and leveraing all its properties and benefits.

12 Future Areas Of Development

The scope of the ZK Rollup project intends to first implement the needed essentials to get the rollup off the ground, and then pivot to a long-term plan focused on cleaning up all loose ends and building something truly long-lasting. The following are future areas of development part of the road-map:

12.1 Decentralizing the DA layer

For the first version of the DA layer, anybody will be able to run and validate the DA layer on their own machine, but will not be able to become an active contributor.

Decentralizing the DA layer means that we need to be able to submit cryptographic proofs that can be checked by a Mina smart contract to verify that not only has a DA validator stored the current rollup block, but also that they still have access to historic data.

Below is a table of the performance of polynomial commitment schemes (not of data availability sampling algorithms using these commitment schemes). Effectively, this table represents the fundamental limit of how fast any optimal data availability solution can be based on these primitives.

	FRI	IPA	KZG
Proof size	$O(\log^2 d)$	$O(\log d)$	$O(1)$
Verify time	$O(\log^2 d)$	$O(\log d)$	$O(1)$
Prove time	$O(d \cdot \log^2 d)$	$O(d)$	$O(d)$

Table 1: Performance of polynomial commitment schemes

This means there are three main research directions possible:

1. Exposing BLS curves to Mina contracts (may have adverse performance restrictions and impact proof system complexity)
2. Wrapping KZG proofs with some proof of equivalence under Mina's cryptography (may have too high overhead to be realistic)

3. Novel research work on usage of commitment schemes supported by Mina for DA sampling (may be time consuming)

12.2 Governance - DAO

Rollup project token holders have the ability to vote for various proposals that dictate protocol parameters and upgradability of smart contracts. The DAO itself will exist on Mina L1, but users will typically vote on the L2 which will be transferred over to the L1 for them. This allows users to take advantage of faster block times/better UX, but enable the DAO to be more robust.

12.3 L2 Token Staking

In the Blockchain space today, users are accustomed to earning staking rewards from the tokens that they hold. This enables ecosystems to grow faster as those who believe in the project end up holding a larger stake in the system, and thus have the ability to have their voice be heard when taking part in governance.

A portion of bridge fees, along with potentially other fees throughout the L2/DA layer can be repurposed to go to stakers. Further research needs to be done on the best mechanism, and at the same time taking advantage of the fact people are staking to improve security when we move towards a multi-sequencer setup.

Of note, native staking/delegation in Mina is not sufficient in and of itself off-the-shelf for the L2, due to the fact Ouroboros does not guarantee that a leader is elected for every slot/block, nor does it prevent multiple leaders from being elected at the same time.

12.4 Safe Exit From L2→L1 Without Sequencers

In the case that sequencers go offline or go rogue and stop posting batches, having a safe exit path for users on the L2 to withdraw their funds is an important step for the ZK Rollup. One difference in comparison to safe exit protocols with Ethereum-based rollups is that we require DA to be ensured for the safe exits as well. This means that there will be further engineering work required to ensure this will be a seamless experience for the end user who has to post directly to the DA layer prior to performing the safe exit on Mina L1.

12.5 Interoperability With Other L2s

Other rollup/app-chain L2s will likely be deployed on top of Mina which may desire to read data from our ZK Rollup. Providing rails for these alternate L2s to easily read proofs of L2 state will improve composability of the entire Mina ecosystem.

12.6 Upgradability

Smart contracts deployed both on the L1 and L2 level will need to be upgraded as the ZK Rollup project matures and new features/capabilities are added (ex. DA layer upgraded). Research into how smooth upgradability can be performed is vital, with the eventual goal of handing off the responsibility to the DAO (requiring a significant majority of the stake to agree upon such updates).

12.7 Connect ZK Rollup To Mina Bridges

As more zkApps are expected to live at the L2 level rather than the L1, this means the majority of useful state that users will want to bridge over to Ethereum will be on the L2. As such it will be important to ensure that the bridges built for the L1 can also be used directly with our L2 to make the developer experience one-to-one, no matter on what layer the zkApp is deployed on.

13 Security Model & Decentralization

13.1 Assumptions

System model. There are multiple kinds of actors in our model:

- **Client**: can lock their funds in the rollup to access the layer 2 ecosystem
- **Proof Generators**
  - **Data availability validator** (*DA validator*): responsible storing historical data and ensuring data is published timely to the data availability layer and generates data availability proofs from this data
  - **Snark worker**: generates the proofs for the L2 transactions and blocks. Can be implemented as a proof market
- **Sequencers**
  - **Sub-sequencers** creates a rollup block based off proofs by the snark worker and a DA proof generated by the DA validator
  - **Aggregators**: aggregates the proofs by the sequencers into a single proof and aggregates all the DA proofs into a single multipoint proof for submission as a rollup block
    Note: if L2 block time = L1 block time, then the sequencer and aggregator are the same actor

The protocol proceeds in (asynchronous, event-based) rounds that are determined by the underlying blockchain – one block on L1 defines one round. s consecutive rounds determine an epoch.

Network model. We assume a partially synchronous communication model as the rollup is intended to operate on blockchains that also remain secure under this network model such as Ouroboros. This means that there is a fixed upper bound on the communication delay but this is unknown to the protocols. Equivalently, we can assume arbitrarily long periods of asynchrony – during which the protocols must remain safe – followed by long enough periods of synchrony (after GST which is unknown) – during which the protocols shall make progress.

Cryptographic assumptions. We make the usual cryptographic assumptions, that secure communication channels, digital signatures, and PKI exist. We only have access to cryptographic functions support by Mina - notably to the Poseidon hash function and Pasta curves.

Threat model. We assume the adversary respects the cryptographic and network assumptions, meaning they cannot break cryptography and they can reorder messages but cannot drop them. Furthermore, the adversary may control up to 1/3 of any actor type (including clients). Lastly, we assume the adversary is slowly adaptive, i.e., within an epoch the adversary is static, that is, they can choose which actors to corrupt at the beginning of an epoch but cannot change the corruption set on the fly within the epoch.

Incentives. Our goal is to eventually design protocols that are secure under rational participants, that is, all actors act in order to maximize a utility function, i.e., their profit. In this model actors are allowed to collude with each other.

13.2 Desired security properties

In this section, we determine what security means for the rollup under the aforementioned model.

- **Liveness:** Any transaction submitted on the rollup becomes stable, i.e., is executed on the rollup by an honest client, within u blocks, where u is the liveness security parameter.
- **Bridging:** Any participant of the L1 can enter or exit the rollup upon request within w blocks, where w is a security parameter, according to the state that an honest rollup client reports. From an honest client perspective, the state of bridging requests on the L1 always matches the state on the rollup client

Bridging encompasses the safety of the rollup. It expresses that no party loses money, i.e., any transaction that is executed on the rollup will remain confirmed at any future time. Bridging also implies that any party that wishes to exit the rollup will leave it in the state that corresponds to the sequence of relevant transactions, meaning the rollup parties will not gain or lose money when bridging to the blockchain. Synchronizing the state of the bridging requests is also defined to be immediate (no

challenge period), which means that rollup blocks need to be able to cryptographically prove which requests they have handled (a system for transaction receipts).

Bridging, in addition, encompasses a liveness notion, as for the rollup to be functional and make meaningful progress, parties must be able to join and leave on demand. Liveness, on the other hand, expresses that the rollup ledger will make progress, thus transactions submitted in the rollup will be eventually executed.

Note that both liveness and bridging security parameters can be expressed in time units when the network is synchronous. However, the L1 may be operating in partial synchrony, in which case we have no liveness guarantees. For this reason, we express the liveness and bridging security parameters for the rollup in blocks.

14 Sequencer Decentralization Auction Model

Instead of leveraging Mina's built in delegation mechanism, we change the leader selection algorithm to minimize the number of missed slots or duplicate work (both of which occur in Ouroboros in the L1). Notably, we introduce an enshrined staking contract that consists of an auction

Although the paper is written in the context of creating a single layer 2, the same protocol can be used to create an unlimited number of L2s unlocking fractal scaling. Any layer 2 created using the Zeko SDK can be validated by a shared sequencer set selected through this auction model, allowing anyone to easily and quickly bootstrap new L2s as needed - effectively turning those selected by the auction model into a set of Kimchi Sequencers

This can be done similarly to the Milkomeda rollup decentralization plan that can be found here, but with a token burning instead of token locking

15 Conclusion

By capitalizing on the unique qualities of Mina and further enhancing its capabilities through the development of the aforementioned project, we strive to advance the experiences of developers and users. Our strategy involves offering a distinct set of advantages that surpass those of existing competitors. We are committed to tackling crucial aspects, including the creation and design of a data availability layer, while proactively considering future plans. With our extensive expertise in the rollup domain, we not only address technical challenges but also prioritize vital attributes of a well-functioning system, such as developer and user experience, as well as the economic and composability aspects between Layer 2 solutions and other blockchain networks. We envision a future where Mina's Layer 2 solutions are built upon interoperability and shared infrastructure, utilizing the knowledge and development resulting from this project.