Launchledger Protocol Whitepaper

25th July, 2023

Fifth Revision
First Version Published May, 2020

Launchledger Labs GmbH
https://launchledger.io
info@Launchledger.io

Contents

Introduction & Hypothesis

For over a decade, the primary method of transferring value between blockchains has been the centralised exchange. This reality has limited composability between ecosystems and has made it difficult for specialist or alternative blockchains to be readily adopted by increasingly large Web3 userbases. Despite this, the vast majority of Web3 users rely on non-custodial wallets to perform even the simplest functions with DeFi, NFTs, DAOs, and a myriad of other applications.

Engineering a generalised solution to this problem has been under discussion as far back as 2012[1]. A decade later, and centralised exchanges still have a firm grip on the market share, but on-chain solutions have been making significant progress.

Uniswap, a competing implementation of the core Automated Market Maker (AMM) idea first created by Bancor in 2016, revolutionised crypto-markets forever by popularising liquidity pools and on-chain trading in mid-2020. After fighting off several competing protocols, Uniswap v3 has cemented the Uniswap product as the dominant solution[2] to swaps on Ethereum mainnet since its release in mid-2021.

Uniswap achieved this by enabling on-chain trading not just for end users, but also by allowing other products and contracts to programmatically swap tokens behind the scenes, something which had not been previously possible. This allowed for the creation of a variety of user experiences and DeFi products which have generated significant fee revenue on Ethereum since 2020. Efficient token swaps powered by Uniswap have enabled the DeFi ecosystem as a whole to rapidly evolve, and created an ecosystem of trading and finance run entirely on-chain.

Author's assertion: *"With no Uniswap, there would probably be no Aave, 1inch, Curve, Yearn, or even OpenSea. On-chain trading is an immense force-multiplier and is perhaps one of the most important primitives in all of crypto development."*

Due to the protocol's success, several Uniswap-based protocols have appeared on alternative smart-contract blockchains. This has allowed for relatively easy on-chain trading in a single-chain smart contract execution environment, with much of the same positive impacts Uniswap had on Ethereum within each individual blockchain ecosystem.

However, Web3 extends far beyond a single environment. As scalability problems continue to impact the user experience of popular chains, either through high transaction fees, unreliable network performance, or additional friction in the user experience (such as that with sharding and L2s), it is clear that multiple, mutually isolated execution environments will coexist long into the future. It is also clear that application specific blockchains and whole ecosystems are still cut off from on-chain markets, and that there is no good method of programmatically trading between any of them.

Since the original Launchledger Whitepaper was published in May 2020, dozens of "cross-chain" solutions have emerged on the scene. Whilst it is clear that there is huge market demand within the sector, none have achieved widespread adoption to the extent of Uniswap, in spite of the fact that the addressable market seems much bigger than ERC-20 tokens alone. Reasons for that can be summarised by the following:

- The proposed solutions are often overcomplicated, under-scrutinised, and/or highly centralised[3], exposing LPs and users to ongoing risks.
- Software vulnerabilities[4] have been so frequent[5] and expensive[6] to the point that some users have come to trust centralised exchanges *more* than trustless solutions.
- Many products have been built that rely on creating or utilising synthetic assets which violate the sovereignty of execution environments[7], fragment liquidity, and ultimately degrade the user experience for everyone.
- The typical cross-chain user experience is almost universally poor and badly explained.
- Products that do offer native swapping often do so at rates which can be described as vaguely similar to the global index price at best, and often with unacceptably high slippage. Accurate and fair swap pricing should not be an afterthought.
- So called "generic cross-chain messaging" solutions are extremely limited in what they can practically enable, making the addressable market for these products small relative to that of spot markets in general.

Rather, the method for transferring of coins between chains would be better if:

- It is wallet and chain agnostic. That is, it supports any generic wallet that can send ordinary transactions on any type of blockchain.
- It does not require native chains to support a specific execution protocol or make changes to its underlying consensus rules or infrastructure, meaning it is generalised and not limited to the Ethereum Virtual Machine (EVM) or smart contract enabled chains.
- It executes as much computation off-chain as possible, meaning that gas usage on expensive chains is minimised and the execution environment can be customised to better suit the use case.
- It does not involve any 'wrapped' or synthetic assets. That is, there was simply one generic transaction submitted to conduct the swap, and users are not exposed to any risk after the swap is complete.
- Developers could easily leverage the technology to improve their own products through simple Remote Procedure Calls (RPC) calls, meaning no special wallets, Software Development kits (SDKs), or other complex frameworks would be needed for the protocol to be leveraged by application developers.

Programmatic swapping is what Uniswap enables on Ethereum. Launchledger's ultimate goal is to enable programmatic swapping for all major blockchains, unlocking new possibilities for product developers, and offering users an easy to use, reliable, secure, and permissionless method of avoiding custodial exchanges altogether.

Our thesis is that if this goal can be achieved, then other types of interoperability solutions will not be desirable for most users, and the Web3 space can finally begin to rely on on-chain markets as a universal mainstay of the industry.

About this Paper

This is the fifth version of this whitepaper. The first was written under the name "Blockswap" and was published in May of 2020. Original contributors to the Blockswap paper include Kee Jefferys, Jason Rhinelander, Dr Maxim Shishmarev, and Simon Harman. The original Launchledger paper can still be read online, but this updated version is intended to reflect the actual design of Launchledger in practice, with a multitude of changes to the protocol having been made through the process of 3 years of further research and implementation.

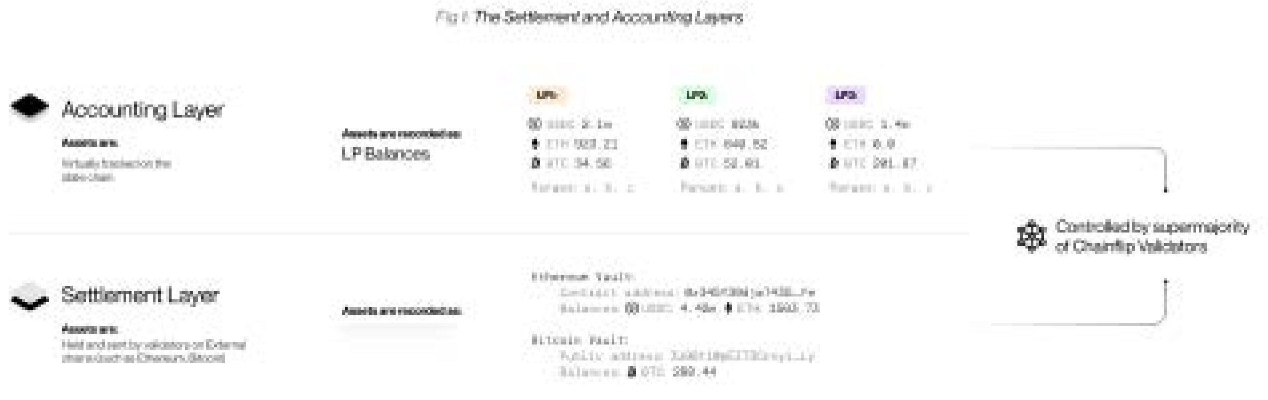
This paper is not intended to be a canonical description of all components and their functions. The goal is to offer a medium level overview of the system, how it is intended to fit together and why certain design choices have been made in relation to the thesis. To do this, the components of the Launchledger design must be laid out, but as a distributed and continuous system, there can be no natural logical place to begin, as every major competent is co-dependant on others. Instead, let's begin with some helpful abstractions to understand the high-level operation of the Launchledger protocol.

The **Launchledger** Core Protocol

Sometimes called a "Cross-Chain Liquidity Network," the core of the Launchledger idea is to use MPC (Multi-Party-Computation), and in particular TSS (Threshold Signature Schemes) to create aggregate keys held by a permissionless network of 150 **Validators**. These Validators control simple smart contracts/wallets called **Vaults**, on multiple blockchains simultaneously, giving rise to a fully decentralised **Settlement Layer**. This is paired with an **Accounting Layer**, which uses the Substrate-based Application Specific Blockchain that we call the Launchledger **State Chain** to track balances, process events, and execute instructions. These systems give rise to a fully decentralised general cross-chain swapping protocol.

Conceptualizing Settlement and Accounting as Separate Layers

In practice, the system's implementation does not achieve a perfect separation of the Accounting and Settlement Layers. Nevertheless, considering the system as two distinct layers proves beneficial in grasping its overall structure.



The assets used for trading are held in the Vaults on chains such as Ethereum, Bitcoin, and so on, with all of the trading and account-keeping occurring on the **State Chain**. This means that swap execution and AMM design can be heavily customised for the specific use case of cross-chain value transfer.

The protocol's accounting is fully contained in the **State Chain** and includes the **Launchledger Just In Time (JIT) AMM**. It is based on the Uniswap v3 AMM design, but unlike Uniswap v3, the JIT AMM is not represented as a series of smart contracts on a single blockchain environment. Rather, the JIT AMM is virtual. This means that funds are not held directly in on-chain pools using wrapped assets, but instead traded virtually on the State Chain as account balances, and settled using the actual assets held natively in Vaults by the protocol.

This works in a similar fashion to how centralised cryptocurrency exchanges store all of the deposits in a unified wallet system, with the balances of individual users tracked in a connected but logically separated accounting and trading system.

Trading and tracking assets virtually on the **State Chain** dramatically simplifies the work needed to support individual chains, as rather than needing to write swapping logic in a range of smart contract and scripting languages on external blockchains, it is entirely contained within the Launchledger **State Chain** environment, meaning all of the abstractions needed to support a given chain can be managed entirely within the 'settlement' layer, and are much simpler in comparison.

The two systems, although logically separated, are controlled by the same set of Validators on the Launchledger network.

Defining the Validators

A Validator, in the context of Launchledger is a server operating a Launchledger State Chain node, as well as additional 'sidecar' functionality through the **Launchledger Engine (CFE)**, an off-chain processing module. For the purposes of introduction, at a high level:

- Up to 150 Validators can participate in the protocol's **Authority Set** at any one time, and collectively operate almost all of the protocol's functions concurrently, and jointly control all funds in the system.
- All Validator slots are entirely permissionless, where any Validator operator with sufficient LLL to outbid others in Auctions becomes a member of the Authority Set after winning.
- Each individual Validator has its own set of private keys which it uses to participate in the consensus of the State Chain, as well as to generate secrets for the purposes of TSS key generation and signing.
- Every Validator simultaneously watches all supported external chains for **Witnessing**, which is functionally equivalent to the network being its own decentralised oracle for incoming transfers.
- Using the TSS protocol in chorus with the other Validators and the external blockchain connections, Validators also **Broadcast** transactions to send funds out of the protocol. Again, the network is its own decentralised relayer.

Validators in the **Authority Set** perform the following functions:

- Accounting Functions:
  - Maintain and update the State Chain through Aura proof of stake consensus, producing blocks and executing extrinsics and events such as swaps, liquidity updates, deposit channels, supply updates, and much more.
- Settlement Functions:
  - Participate in TSS Keygen ceremonies to create or rotate control over Vaults.
  - Participate in TSS Signing ceremonies to construct valid transactions for external blockchains, and broadcasting those signed transactions to the mempools of those external networks (Broadcasting/Egress).
  - Monitoring and Parsing incoming transactions to vaults on external chains to be Witnessed on the State Chain. (Witnessing/Ingress).

Validators are covered in more detail in the Validator Network section. With this high-level overview of the network laid out, let's use it to understand more components and how they come together to solve the cross-chain swapping problem.

The Ground Floor: Constructing Vaults

Fundamental to any native asset swap flow is for there to be a pool of liquid assets somewhere. With no source of on-chain liquidity, there would be nothing to swap. At the ground floor of any credible native cross-chain swap protocol, a general solution to flexible liquidity storage must be engineered. In this section, we'll pull apart what a **Vault** is and where it fits into the wider protocol.

"A fundamental limitation of other interoperability protocol designs is that they lack the ability to secure native funds on host blockchains. So called 'messaging' or 'interoperability' protocols offer insufficient functionality to carry out large-scale cross-chain trading. These protocols only enable 1 to 1 swap, which limits their utility to users. This type of protocol design cannot fulfill the functionality ultimately necessary to render a centralised exchange offering obsolete, and in turn, might explain why they have achieved limited adoption."

A **vault** is a method of storing funds on a given blockchain that is controlled by the Launchledger protocol. This could come in a few forms, with different blockchains offering different features and limitations, but broadly there are two categories of vaults that are used to store funds to be used in the AMM:

1. **Vault Contract:** A smart contract (EVM or non-EVM) that holds a pool of assets on a given chain. The vault smart contract can send funds with the approval of an aggregate key held in the KeyManager contract. The aggregate key is controlled by the Authority Set in a 100 of 150 threshold signature scheme (FROST) handled off-chain. When the Authority Set is changing, the old Authority Set approves a transaction that changes the aggregate key from the outgoing Authority Set to the new aggregate key, controlled by the new Authority Set in the KeyManager contract.

2. **Native Wallet Vaults:** A native wallet or set of wallets controlled by the Authority Set in a 100 of 150 threshold signature scheme and is handled off-. As with the vault contract above, the native wallet vault still relies on the FROST threshold signature scheme to function. The details of the native wallets vary from blockchain to blockchain but rely on native multisig schemes offered as features on each blockchain. Funds are held in each Native Wallet Vault until a new Authority Set is determined, after which a **Vault Rotation** occurs. This could involve the old Authority Set delegating access to the Vault Account on-chain, or the new Authority Set will create its own Native Wallet Vault, and the outgoing Authority Set will hand over its keyshares to the new set and sweep funds from its own Wallet Vault into the new one. For an example of this in practice, check out the Bitcoin Vault page. See *Swap Intent & Vault Expiry: Handling Vault Rotations* for more information.

   Although not used for storing AMM funds, there is also a third protocol-controlled vault-like contract that fills a special function called the:

3. **State Chain Gateway Contract:** The State Chain Gateway is a smart contract (initially just on the Ethereum network) that holds LLL funds as collateral for accounts on the State Chain. Like the Vault Contract & Native Wallet Vaults, signature verification is delegated to the KeyManager contract, which verifies an aggregate signature produced by the current Authority Set. Unlike the Vault Contract, users directly interact with this contract, but can only redeem LLL from this contract with a valid signature from the aggregate key - or in other words, with the off-chain approval through a transaction signed by the Authority Set.

In all of these cases, a new Authority Set must perform all of the necessary key-generation ceremonies for all deployed vaults before authority over the State Chain consensus, Vaults, and the State Chain Gateway contract are handed over.

No matter the underlying implementation of each Vault, the end result is that there is an account on each supported blockchain under the control of the Launchledger protocol. This forms a functional, decentralised, and generalised Settlement Layer on which liquidity pools and staking functionality can be virtually constructed using assets held in these vaults.

In the next section, we'll explore the Validator Network and the State Chain that coordinates them in greater detail.

Validator Network

The State Chain

The State Chain is an application-specific blockchain (Appchain) at the heart of the Launchledger network designed to facilitate decentralised cross-chain swaps within the Launchledger ecosystem.

It is based on the Substrate blockchain framework built by Parity but remains an independent chain from the Polkadot relay network. Written in Rust, it utilises the Substrate Runtime, many of the original pallets, and the blockchain node infrastructure to operate.

Key Functions

The State Chain serves as the core database for all activity occurring in the Launchledger protocol. All protocol events are recorded, executed, or triggered by the State Chain. This covers almost everything, including, but not limited to:

- Validator Auctions and Authority Set selection
- Vault Rotations through each stage
- Slashing and Validator Reputation
- Opening and closing Deposit Channels
- Witnessing consensus
- Broadcasting consensus
- The JIT AMM and all of the swaps that are conducted natively through the protocol
- Emissions and Fee collection
- Governance

Features and Benefits

Utilising the State Chain instead of depending on an externally operated L1, L2, or other broadly applicable blockchain offers significant benefits for the protocol, such as:

- Application-specific: Unlike other blockchain networks with multiple use cases, the State Chain is specifically built to run the Launchledger protocol. This is beneficial, as it means the execution rules and parameters can be automated and optimised for the core protocol. It is not possible to build or deploy apps or smart-contracts directly to the State Chain. Instead, developers can leverage the Launchledger SDK, APIs, or smart contracts on external chains to interact with and build on top of the Launchledger network.
- Scalability: The State Chain is designed to handle a high volume of cross-chain swap transactions efficiently. To achieve this, the chain utilises transaction batching and other automated processes to save on gas, and bundles together events to cut down the number of user interactions required to complete actions through the protocol.
- Security: The State Chain is secured by a set of Validators who bond the native Launchledger token ($LLL) as collateral. Validators are responsible for processing transactions, creating new blocks, and participating in the consensus process. Any malicious behaviour can lead to the slashing of their staked tokens, providing a strong deterrent against attacks, which would be harder to effectively implement if the core of the system were managed externally.
- Governance: The State Chain also include governance mechanisms that allow token holders and the governance council to easily handle protocol upgrades, parameter adjustments, and other aspects of the Launchledger ecosystem. This fine-grained control gives the protocol an extremely high degree of flexibility and upgradeability, ensuring that the network remains decentralised and evolves according to the needs and preferences of its users.

Ingress: Witnessing Deposits

Ingress is the generic term to describe the process of observing external chains for events and registering those related to the Launchledger protocol. Some of those events include:

- Users sending deposits to be swapped
- Liquidity providers making deposits to add to their liquidity accounts
- Users funding their State Chain account for Validator auction bidding, broker operation, or any other reason);
- Changes to Vaults triggered by the network

Witnessing

All of these events are registered on the State Chain through **Witnessing**. Conceptually, we can say that the Witnessing process is functionally equivalent to a Launchledgerspecific Oracle service.

The State Chain contains a list that instructs Validators what to look at on external chains. Certain contracts on Ethereum, particular wallet addresses on Bitcoin, and a myriad of other addresses and contracts on other chains can all be added to this "watchlist" for the Validators to scan external chains for.

Monitoring External Chains

As an essential function of the settlement system, Validators are required to maintain connections to light clients, full nodes, or other services that allow the Validator to query all Launchledgersupported blockchains. The way this is configured is up to each individual operator, but by losing connections to the external blockchains, the Validator will be sent offline until it has resolved the problem.

The Witness Process

Once a related transaction to an address or contract on the "watchlist" occurs on the external chain, the Validators must wait until a pre-specified confirmation window has passed. The exact length of the confirmation window (measured in blocks) is assessed on a case-by-case basis for supported blockchains.

Author's assertion: *"No decentralised interoperability protocol can escape the reality of blockchain finality and is uniquely vulnerable to rollbacks between independent chains, thus resulting in an unavoidable delay before secure witnessing and processing can occur."*
However, as soon as a block is detected on the supported blockchain which causes a deposit to be considered 'final' according to the Launchledger protocol's definition, every Validator in the Authority Set submits a 'witness' extrinsic to the State Chain. When over $\frac{2}{3} + 1$ of the Authority Set (100 of 150 in the standard case) have done so, this finalises the witness event. From then on, the State Chain registers the deposit as 'real,' and the transaction details such as the TXID, asset, amount, and any relevant metadata are recorded on the State Chain.

If a Validator submitted a witness extrinsic which did not align with what other Validators have seen on the supported blockchain, the witness extrinsic would never reach sufficient consensus on the State Chain. This 'false' witness could then be used as proof to punish the Validator that submitted it. Similarly, nodes that frequently fail to witness events that the other $\frac{2}{3} + 1$ of Validators catch within a given time frame could also be grounds to programmatically punish a Validator through Slashing.

Summary

Witnessing is effectively how assets are passed through the settlement layer into the accounting layer, and is at the heart of all ingress processes, three of the main ones being:

- State Chain LLL Gateway
- Native Swap Flow
- Liquidity Provider
  Accounts

Egress: Broadcasting Funds

Registering external events into the protocol is one thing, but sending funds out of it is more complex. Launchledger also has a generic process for creating, signing and broadcasting transactions out of the protocol: Egress. Some examples are:

1. Sending the output of a swap to a user's **destination address**;
2. Sending assets to a liquidity provider that is **withdrawing their collateral**,
3. Sending ERC-20 $LLL funds from the StateChainGateway contract when a Validator or other State Chain account holder redeems $LLL from their State Chain account, or;
4. Rotating aggregate keys or sweeping funds in the Vaults.

As with *Ingress*, the process for *Egress* differs slightly from case to case but shares a common process.

Threshold Signing

Events on the State Chain may trigger an Egress process. Typically, those events will also include the complete information required to construct a valid transaction to be signed over, including the gas fee and limit. Wherever possible, egress processes will be batched together to save on gas fees and to minimise the number of signing ceremonies required to conduct Egress operations.

A *'ThresholdSignatureRequest'* signals to the Authority Set that a transaction must be signed by the network. Upon receiving this event in a State Chain block, the Validators begin a ceremony off-chain in the Launchledger Engine.

Threshold Signature Scheme (TSS) Signing Ceremonies

TSS signing in Launchledger is conducted using the FROST multisig scheme, which relies on Schnorr signatures for fast and scalable multi-party computation, allowing for a large set of signers. FROST allows Launchledger to secure all Vaults using the same Authority Set of 150 Validators, offering substantial benefits in terms of economic security and a simpler Vault management logic compared to other Cross-Chain Liquidity Networks.

If the ceremony fails, it will time out on the State Chain and will be automatically restarted. Once the signing ceremony has been successfully completed, the signed transaction is reported on the State Chain and can now be broadcast to the external blockchain.

Broadcasting

The Validators will now deterministically nominate one of the Authorities to broadcast that signed transaction to the destination chain. Technically, in most networks, anyone can submit this transaction from any client, but as this is a programmatic system, one is chosen to do so to automate this flow for all users.

If the nominated Validator can't send the transaction for some reason (not enough gas, broken connection, etc.), they will report the failure on the State Chain and another broadcaster is selected.

Otherwise, the network will wait for the broadcaster to submit the transaction externally, which will then be witnessed by the network. If the broadcast is witnessed by the network before the timeout, the broadcast is considered successful. If it is not witnessed by the end of the timeout period, the nominated Validator will be punished and a new Validator will be nominated. This will continue until the broadcast is successfully witnessed.

Native Swap Flow

Let's look at how native cross-chain swaps are done from end-to-end in the Launchledger protocol. Launchledger swaps are **fire-and-forget**, meaning once the deposit is made, the user doesn't need to do anything for the swap to take place.

Initiating a Swap

Either through the frontend (Swapping app) or the Launchledger SDK, there are multiple methods of starting a Launchledger Swap.

- Registering an intent to swap using a Broker and creating a Deposit Channel
- Calling the swap function in Launchledger Vault contracts, which can either be done by the user or by other contracts for maximum composability **(EVM chains only)**
- Directly submitting a swap Deposit to a Vault using special transaction metadata **(Not yet built)**

*Funds sent straight to Launchledger vaults without initiating it using one of these methods **will result in the loss of funds**, as the network won't know what to do with it.*

All of the above methods will register the swap and essential information, such as:

- Destination Asset
- Destination Chain
- Destination Address
- Any call data for a follow-on Cross-Chain Message (such as a Uniswap or external Bridge call)

Once a method is chosen, the user can send their funds on the external chain (BTC, Ethereum, to be swapped in accordance with their chosen method.

*Deposit Channels remain open for 24 hours. After that time, the network may no longer recognise late deposits. Users should **always open a new Deposit Channel** and send their deposit **immediately** every time they swap.*

Witnessing the Deposit

The network will monitor open Deposit Channels for transactions and the Vault Contracts for swap calls. Upon seeing a deposit for a Swap, it will be Witnessed by the network and registered on the State Chain.

The delay between the deposit transaction being made on the source chain and its confirmation on the State Chain through Witnessing depends mainly on the source chain. On Ethereum, the threshold is currently 6 blocks (90 seconds), whereas Bitcoin is 3 blocks (30 minutes).

Pre-witnessing will allow the network to confirm transactions faster or slower depending on the value of that deposit. This will cut down the confirmation window for small deposits, and increase the security for larger transfers. This feature is currently under active development.

Processing the Swap

Once the swap is Witnessed it is ready to be swapped in the JIT AMM. At the end of each block, Swaps are automatically bundled and processed. This is so that liquidity providers can bid on swaps in the same block where they occur. Liquidity providers monitor incoming deposits to bid on them at their best price.

- For each asset in each pool, swaps that take place in the same block are bundled together and swapped in one step to eliminate frontrunning and MEV potential. This is done separately for each direction of trade.
- For swap routes that require multiple swaps through pools, for example, BTC -> USDC -> ETH, each swap is conducted sequentially, usually in the same block.

- Whenever the swap input is first denominated in USD, the Launchledger Network fee is deducted. This could be at the start, middle, or end of swap processing depending on the route. This always occurs at some stage in the swapping process as a function of having$USDC Denominated Pools.
- For each swap conducted, liquidity fees are collected and the output is forwarded to the next step.

Once all required swaps have been completed, the swap output is ready for the final stage of processing.

Broadcasting the Swap Output

Once a swap has been automatically executed on the State Chain, the assets to be paid out to the user are listed as a **Pending Egress Transaction**. Using the Destination Address specified in the user's Swap Initiation step, and bundling the Pending Egress Transaction with others into a single batch, estimated gas fees are deducted from the final Swap Output amount to compensate the network for making the Broadcast.

If the user has included external call data in their swap request, that transaction will be broadcast separately to account for additional gas costs incurred at the user's request that shouldn't be socialised, but the egress is executed using the same TSS and broadcast method.

The final **Egress Payout Transaction** is constructed, and the Authority Set conducts a Signing Ceremony to produce a valid signed transaction for that chain, which is Broadcasted automatically by the network.

The user now has their shiny new native assets sitting peacefully in their wallet (or anywhere else that they specified).

Swap Fees In Summary

Fee Type	Fee Level	Description
Deposit Gas Fees (Normal User Transfer)	Varies per chain and deposit type, but usually 31k gas on Ethereum	Network gas fees will vary depending on the incoming chain. Coming from BTC will differ from those coming from ETH for example. These will be paid by the swapper when they deposit from their wallet.
Liquidity Fee Per Pool	0.05% - 0.20%	Fixed Liquidity Provider Fee (Pool Defined)
Network Fee for Buy/Burn Mechanism	0.10%	Fixed Network Fee used by the protocol to buy LLL tokens from the USDC/LLL pool on the State Chain, which in turn are burnt.
Destination Chain Transfer Broadcast Fee (Transfer by Protocol)	Varies per chain	Network gas fees will vary depending on the withdrawal chain. Batching, however, will lower the fee that would normally be paid.

Liquidity Provider Accounts

Users that wish to provide Liquidity in the Launchledger JIT AMM follow a process that is not too dissimilar to a typical centralised exchange flow.

Creating a Liquidity Provider Account

Liquidity providers need to have an on-chain account for the Launchledger State Chain and refund addresses ready for each asset. To achieve this, they will need to use the State Chain LLL Gatewayand fund their account with a nominal amount of LLL. This LLL is used to pay the minimal gas fees required to place and update orders.

Liquidity Provider accounts can be managed through the Launchledger-cli, Polkadot.js wallet, or through the Pools web interface.

Depositing Assets to a Liquidity Account

To open any position, first Liquidity Providers must Ingress their assets into the system.

1. Firstly, Liquidity Providers must nominate the assets they wish to deposit. They submit an activate_asset extrinsic to the State Chain, which includes the asset type and a refund address in case their collateral must be returned to them due to the Security Ratio. This then opens a Deposit Channel to which the LP can send their asset (BTC, USDC, etc)
2. The Liquidity provider then sends funds using any wallet to each Deposit Channel Address. The transactions are in turn witnessed by the Validators, resulting in the credit of balances for each asset on the State Chain.
3. The Deposit Channel closes after 24 hours. Users should always open a new Deposit Channel and send their deposit immediately every time they make a liquidity deposit.

Managing Liquidity Positions

To create a range order or limit order in the JIT AMM, Liquidity providers need to have deposited enough assets to open the position from their virtual balance on the State Chain.

- Liquidity providers can open a Range Order on any given pair by submitting a simple extrinsic using their State Chain account. The parameters for range orders are similar to those of Uniswap v3. These positions can also be closed or updated using an additional extrinsic.
- Limit orders can also be placed using a separate extrinsic. Limit orders are maker-only, meaning that Liquidity Providers can not trade against other limit orders directly, and instead must conduct a normal swap. Filled limit orders are automatically closed, and the swapped funds return to the Liquidity Provider Account as a free virtual balance. Unfilled orders can be canceled using another extrinsic.

Withdrawing Collateral from a Liquidity Account

Not only is the ability to withdraw liquidity essential but it is expected to be fairly frequent in the JIT AMM. Liquidity Providers will often have to rebalance their asset float across the markets in which they operate to maintain a balanced and effective market-making strategy.

1. Providers submit an extrinsic to the State Chain which requests that the network send the specified amount of assets to their refund address, which would have been specified when the account was created. Refund addresses can also be updated without creating a new account. This instruction becomes a Pending Egress Transaction.
2. The Authority Set commences a Signing Ceremony to create a valid transaction to send funds from the relevant Vault to the Liquidity Provider, deducting the gas free required from the balance owed to the user. The signed transaction is broadcast to the relevant external chain by a nominated validator, and the user receives their funds less the gas fee.
3. The Authority Set witnesses the egress transaction and records the transaction as settled in the accounting layer.

State Chain LLL Gateway

$LLL is issued as an ERC-20 token on Ethereum. This means that in order to acquire a balance on the Launchledger State Chain, users have to fund their State Chain account in the State Chain Gateway smart contract.

State Chain accounts are used to submit extrinsics on the network (such as liquidity updates and Broker transactions) and participate in Validator auctions and Governance.

The Gateway

The State Chain Gateway contract is monitored by the network. It allows the network to mint and burn ERC-20 $LLL based on the balance of accounts it is tracking on the State Chain. As Validators earn rewards, and fees are collected and $LLL is burned, these supply updates are fairly closely reflected with the total ERC-20 tokens held in the Gateway.

The Gateway essentially gives the Launchledger network the ability to control the supply and economics of the $LLL token while still allowing users to hold and use it as any other token using the widely adopted ERC-20 standard.

This is achieved by allowing the protocol-controlled Aggregate Key (managed by the KeyManager contract) to have authority over the Gateway.

Funding State Chain Accounts through the Gateway

1. The user submits a transaction to the Ethereum chain that calls a function in the Gateway contract that passes in their LaunchledgerAccountID (a hex representation of a substrate SS58 encoded public key) and the amount of $LLL tokens being sent.
2. The Authority Set listens to the Ethereum blockchain for transactions related to the StakeManager contract. Once the staking transaction has been confirmed on the Ethereum network, the Authority Set commences the **Witnessing** process.
3. As the transaction becomes witnessed, the deposit is considered "settled" - and so the account specified in the contract call becomes funded on the State Chain with that amount of $LLL which can then be used to bid in auctions for Authority Set slots as a Validator, or simply to pay for LP or Broker transactions on the State Chain.

Redeeming ERC-20 LLL through the Gateway

The process of withdrawing $LLL from the State Chain to Ethereum is known as *redeeming*:

Whenever any actor on the Launchledger Network wishes to redeem $LLL collateral held on the State Chain to the Ethereum network, this requires the approval of the Authority Set.

Note the following restrictions on redeeming $LLL tokens:

- *During the auction phase, a bidding Validator may not withdraw any funds.*
- *Authorities may not withdraw bonded $LLL at any time.*
- V*esting $LLL tokens may only be withdrawn to one of the vesting contracts used to fund the account.*

The steps to redeem $LLL are as follows:

1. The user submits a redemption request to the State Chain, specifying the amount of tokens they wish to redeem and the Ethereum account that will receive them.
2. If the user's redeemable token balance is sufficient to cover the requested redemption amount, the authority sets authors, signs, and broadcasts a 'registerRedemption' transaction to the Ethereum network. This transaction specifies the amount, destination address, and expiry time of the redemption.

3. The KeyManager contract verifies the threshold signature to confirm the authenticity of the request and starts a 2-day countdown until the redemption can be executed. Note, that tokens that are bonded or actively being used to bid in an auction are not available to be redeemed.
4. At the end of the countdown, a window opens during which the user can finalise the redemption by calling the ‘executeRedemption’ function in the Gateway contract. This transfers the $LLL ERC-20 tokens from the Gateway contract to the account specified at the initial stage. The authority set witnesses this and settles the redemption on-chain. Note, that redemptions have a defined expiry time. If the expiry time elapses, the redemption can no longer be finalised. Instead, calling the ‘executeRedemption’ contract function will revert the operation. The authority set witnesses this and the funds are credited back to the originating account on the State Chain.

The delay mechanism exists as a fail-safe for the (hopefully!) unlikely event of a supermajority attack or software exploit that would allow an attacker to craft a fraudulent withdrawal request. Additional governance mechanisms exist to prevent and mitigate the severity of attacks and exploits. For more details, see Governance and Security.

Validator Auctions, Bonds & Rewards

Auctions

What is the Validator Auction?

The 150 Validator slots are granted based on the results of regular Auctions, whereby the highest bidders that complete Keygen at the end of each Auction are selected as the Authority Set for the next Epoch and a Rotation is conducted to handover Authority from one set to the next. On Perseverance, the time between each is usually 3 hours long but will be longer in production.

Bidding is done from a Validator's State Chain account, which is funded through the State Chain LLL Gateway.

Auctions begin halfway between the start and the expected end of an Epoch. During Auctions:

- Bidding Validators commit their LLL balance to the auction for its duration and may not change their state to Non-Bidding.
- Non-Bidding Validators (without a current bond) can redeem funds as normal but can change their state to bidding during the auction.
- All Validators can add extra funds at any time.

In order for a node to retire, it must enter a Non-Bidding state before the next Auction begins. See Validator Types & States for more information.

Resolving an Auction

The aim of an Auction is to derive a list of Primary and Secondary candidates for the Key Generation & Rotation stage (KeyGen). The way Auctions are resolved is as follows:

1. Determine the maximum set size P for the next Epoch (default 150).
2. Sort all Qualified nodes by descending bid. These are the Auction Bidders. Non-Qualified Nodes are never considered in an Auction (see Validator Types & States).
3. The P highest bidders are deemed the auction winners and will be the Primary KeyGen Candidates.
4. The MAB, or Minimum Active Bid, is defined as the lowest winning bid, ie. the lowest bid of all the Primary KeyGen Candidates.
5. From the remaining bidders, discard any whose bid is lower than 1/2 of the MAB. Select up to P/3 bidders from the remaining nodes. These will be the Secondary Keygen Candidates.

Performing Key Generation & Rotation after an Auction

Once the lists of Primary and Secondary candidates are determined, the KeyGen ceremonies begin. Firstly, all of the Primary KeyGen Candidates attempt their KeyGen ceremony. If this is unsuccessful, the failing candidates from that ceremony will be Suspended and removed from the KeyGen process. They are replaced by the next highest bidding Secondary candidates. This will repeat until a successful KeyGen round occurs.

This also means that if no successful KeyGen occurs with the Authority Set Size Cap, the set size will continue decreasing until a successful KeyGen is achieved. For example, if the cap of P is 150, and there are 180 Primary and Secondary candidates in total, but 50 in total fail KeyGen through the rounds, the resulting Authority Set Size will be 130 Validators.

There is also a minimum set size, which if reached will cause the KeyGen to be aborted and for another Auction Resolution to take place to repeat the process from the beginning.

Bonds

Once the Auction is resolved, KeyGen is successful, and the Authority Set rotates, all successful Authorities for the new Epoch are Bonded to the value of the Minimum Active Bid of that auction. Any amount staked in excess of the Bond at the conclusion of an Auction can be withdrawn by the bidder between Auctions.

Validators selected from the Secondary set or who have been slashed may have a balance lower than the bonded amount. This means they will not be able to make any withdrawals until they have a balance in excess of their Bond, or the Bond is lifted through retirement or losing an Auction.

The balance of bidders with or without a Bond will be considered as an implicit bid in the next Auction, including any rewards the Validator has earned from their node. This still means that current Validators might have to top up their bid between auctions to keep their slot, but if it's high enough, do not need to do anything extra to succeed in each subsequent auction cycle.

Authority Rewards

The most lucrative form of rewards are paid to the Authority Set. These rewards are distributed to each Validator in the set each time they author a block on the State Chain.

Every Authority member earns equal rewards during an Epoch, regardless of their stake.

Backup Validators

Instead of paying an equal reward to the Backup Validators, a fixed reward is distributed proportionally to Backup Validators based on their stake size. This is to incentivise these Backup Validators to have the highest amount they can for the next auction, in which the highest bidding Backup Validators would be included first in newly available slots in the next Auction, and also to incentivise the nodes to hold onto their stakes and await the next auction if they don't make it in.

For Backup Validators, increased bids for Backup Validators are immediately reflected by the rewards they are paid. This provides a direct and immediate incentive to stake as much as possible as soon as possible, both increasing total bidding and increasing the likelihood that these more active and collateralised Backup Validators will be included in the next set.

The rules for Backup Validators are as follows:

- A capped reward of LLL (much less than the Authority Set reward, 1% compared to 7% of annual supply given to Authorities) is divided among the Backup Validators for a given Epoch.
- System limits prevent individual Backup Validators from earning more than Authority Validators, disincentivising self-exclusion from ceremonies with high stakes.
- There is a limit on the number of Validators that earn Backup Rewards- 1/3rd of the current Authority Set size. **Any bidding node outside this limit will not be a Backup Validator, but may still be selected as a Secondary candidate in the Auctions**.
- So long as Backup Validator remains Qualified (including being Online), rewards will be paid to it based on their stake, proportional to their share of staked LLL in the total number of LLL staked in Backup Validators.
- Any Backup Validator that goes offline will no longer be Qualified and therefore gives up a Backup reward-earning position to another online Bidding node. However, they can come back online at any time, and provided they are still bidding enough to win back their Backup Validator slot, can immediately begin earning rewards again after the first successful heartbeat interval.

Validator Types & States

It is important to note that a Validator does not refer to just any node on the Launchledger State Chain network. In contrast to State Chain Client operators, who may be using them for liquidity management, running a Broker, or simply monitoring the State Chain for data, a Validator, a node must also run the Launchledger Engine (CFE) sidecar module and have configured endpoints for all external chains, as well as set their State Chain account type to Validator.

Status Tag	Description	Criteria
Qualified	A Qualified validator can participate in Auctions.	A validator node is considered Qualified if all of the following conditions are fulfilled: • Account Role is Validator • Online (based on last heartbeat submission) • Session Keys are registered • Peer Id is registered • Bidding is true • Bid is at least half of the current MAB • It is running at least the minimum required version of the Launchledger En-gine
Bidding	A bidding node is indicating its intention to participate in auction. Validators that wish to retire should set their bidding intention to false	Outside of auction periods, Validators can set their bidding status freely. During Auctions: • Bidding Validators may not withdraw any funds and may not change their state to Non-Bidding. • Non-Bidding Validators can withdraw any unlocked funds and can change their state to bidding. • All Validators can stake extra funds at any time
Backup	A Backup validator earns LLL rewards but does not participate in block authorship. See Backup Rewards section. May be a keyholder.	• Is Qualified • Is not an Authority • Is among the set of highest bidding Non-Authorities, (maximum 1/3rd the size of the current Authority Set, usually 50).
Authority	Authorities are responsible for block authorship and consensus, witnessing, threshold signing and transaction broadcasting. In return they earn Authority Rewards. Authorities are Bonded according to the auction preceding the KeyGen ceremony for the held key.	• Auction winner • Successfully participated in a Keygen ceremony. • Block Producer & State Chain Authority • Witnesser

Reputation & Slashing

Validators are generally expected to be honest actors, but penalties must exist to ensure the reliable performance of Validator duties and for countering subversive behaviour. Validators hold shares in keys which can not be used to move protocol funds unless at least 100 of the 150 maximum Validators are online and can sign transactions. The protocol therefore needs to make sure sufficient penalties exist to incentivise consistent, secure, and high-performance Validators.

That being said, whilst it's necessary to discourage Validators from going offline at any point, there are legitimate reasons to do so which we should respect. Updating Validator software is one such legitimate reason, and penalising Validators for minor periods of being offline could also lead to negative results for the network.

Slashing

Taking this all into account, Launchledger operates a unified slashing and reputation system that rolls all penalties into a time-based system with both positive and negative reputation scores, designed to balance the above considerations.

Who Does this System Apply to?
The reputation and slashing system only applies to Authorities. Backup Validators don't necessarily have a reputation and instead are simply not rewarded when offline, and can never be slashed. Their online state is tracked using heartbeat submissions.

Validators start with their reputation at 0. Until they join the Authority Set, their reputation will stay at 0. Validators can be in one of three on-chain states that affect their Reputation score:

1. **Online** - The Validator is submitting heartbeats and is not suspended - the Validator will earn rewards (as long as they author blocks successfully) and will not be slashed while in this state regardless of their reputation score, and will gain reputation over time.
2. **Offline** - The Validator has failed to submit a heartbeat, which is due to some or all of the Validators' critical functions failing. Reputation will decrease over time while in the offline state.
3. **Suspended** - The Validator has been given a time penalty for failing to complete a process in time or for doing something suspicious or incorrectly. The Validator will lose as much reputation as if they had been offline for the duration of the penalty.

If a Validator has a negative reputation and is offline or suspended, they will be periodically slashed until they achieve the Online state again. The more negative the reputation, the greater the rate of slashing.

Slashing Limit

There is a limit to how much a Validator can be slashed. To ensure that there is always a reason to come back online and restore a failing Keyholder, no more than 80% of a Bond can be slashed of a given individual Validator.

Heartbeats & Reputation

Reputation will be accrued by a Validator as long as they are considered "online" on the State Chain. It makes sense to cap the amount of uptime credit available to any single Validator in order to prevent some strange scenario where a long-running Validator can be offline for many days at a time without suffering punishment. Launchledger has an initial cap of 48 hours on uptime credit (A score of 2880), which should be more than enough time for any honest Validator to debug issues with their setup. This cap may be reduced in the future.

Being online is the only way to accrue reputation. If a Validator has a negative balance but is back online, they will start being slashed as soon as they are offline or suspended again. This is a strong incentive to fix issues and be reliable long before a reputation balance goes negative.

The way Launchledger ascertains whether a Validator is online or not is through Heartbeat Extrinsics. At regular intervals, Validators must submit a special transaction to the state chain, simply proving that their hardware is connected and active, though it doesn't necessarily prove their overall performance.

If a Validator doesn't submit a valid heartbeat before the interval deadline occurs on-chain, they are automatically moved into the Offline state and will start losing reputation. The Validator will be return to the online state at the next interval deadline that they submit a valid heartbeat for.

Suspension Conditions

For now, Validators can be suspended for two reasons, both relating to the Egress process:

1. **Failing to Participate in a Signing Ceremony:** During a signing ceremony, it is possible that a selected Validator fails to sign or communicate messages before the ceremony times out. This has the impact of requiring the entire signing ceremony to be conducted again, costing users valuable time. After a failed ceremony, the online Validators vote through consensus on Validators which misbehaved or failed in the signing ceremony, knocking them into the suspended state. The ceremony is then attempted again with a new set that excludes currently Suspended nodes.

2. **Failing to Broadcast:** Account-based chains require our Validators to have funded accounts in order to broadcast transactions. The process of generating an output for these chains also includes a process by which the broadcaster is selected. The signed transaction to be broadcasted should be saved on the State Chain. Failure by a Validator to submit this signed transaction indicates a failure to broadcast the transaction to the external chain. This will require a new proposer to be selected by the State Chain, incurring a slight delay. The initially-proposed broadcast Validator will then be suspended.

In the Suspended state:

- The Validator will not be selected as a signer or a broadcaster,
- The Validator is liable to be slashed, if this state is coupled with a negative reputation value,
- The Validator is still expected to participate in witnessing and submitting heartbeats.

Validators, once the required time has passed, can have their suspended state lifted by submitting a valid heartbeat. If this does not happen by the end of the heartbeat interval, the Validator will be downgraded to offline, meaning they'll have to wait for the next heartbeat interval before they get the chance to return to an online state.

Tuning the time-penalties and exact numbers in regard to how reputation affects slashing is important. It is impossible to know exactly why a Validator triggered a suspension condition. In otherwise performant Validators, a situation could arise where they may have just gone offline before they were called upon, and therefore a suspension occurred. In this situation, however, the Validator should not experience any major difference in reputation. Similarly, Validators which just submit heartbeats in an attempt to evade higher server costs will get repeatedly suspended and then slashed anyway. Finding the exact parameters to strike a balance between the different cases is an ongoing process.

As Launchledger is developed, more advanced suspension conditions can be introduced to better incentivise Validators to be performant. An example would be, "consistent failure to witness'", where a Validator does not submit a witness extrinsic for an otherwise confirmed transaction after a timeout threshold. This is considered future work, as it would be trivial to dodge penalties by simply copying witness transactions without implementing a commit-reveal scheme. We don't consider this to be immediately necessary, but as the number of required chain clients grows, Validators may be more inclined to try and avoid maintaining so many connections, thus possibly necessitating implementing the scheme to ensure secure and timely witnessing.

Reputation Rollover

Reputation which has been accrued by a Validator should roll over into the next epoch if they make it to the next Authority Set. This allows them to drop out of the Authority Set and then re-enter in a future epoch with the same amount of reputation. It should be noted that reputation is not taken into account during the Validator rotation process. Instead, any Validators that are offline (as defined by the State Chain), at the time of an auction being completed, will not be considered for selection in the new Authority Set.

FROST Signing Scheme in Launchledger

Background

The Egress process, and by extension the entire protocol, relies on the use of shared multisignature keys requiring a two-third majority of the Authority Set. Given the varied nature of blockchains existing today, there is not an optimal solution to achieve scalable 100-of-150 Vaults using only a single approach. Launchledger's approach relies on an efficient Threshold Signature Scheme and by leveraging features offered on each chain to achieve scalable vaults.

In the original version of this Whitepaper, we described both the use of Schnorr based Threshold Signing Schemes and GG20, the latter being what is currently deployed in THORChain.

GG20, although able to be applied to almost any blockchain, constrained Validator set sizes due to scalability, forcing vault management systems to be more complex in order to maintain decentralisation. This can be seen in practice within the THORChain protocol design, where the system forces individual Validators to hold user funds on their own hot-keys. It does this to avoid the computation and time cost of a GG20 signing round with the 36 other participants in a given key. The process of “churning” any individual validator is therefore risky and complex.

Instead, Launchledger employs the Flexible Round Optimized Schnorr Threshold (FROST) signing scheme, which to our knowledge, makes the Launchledger protocol the first FROST protocol user of this scheme within the blockchain ecosystem. This scheme relies on EdDSA/Schnorr signatures to function, and thus is not supported by all blockchains. However, the few remaining chains which lack support for these signature types are largely unpopular Bitcoin forks that have not integrated the Taproot changes.

The advantages of using the FROST Scheme above GG20 become apparent when observing signing times in both benchmark testing and in production environments. FROST allows Launchledger to scale to a 100 of 150 system for all supported chains with signing times anticipated to be around one-per-second in production based on current observations. This means Launchledger Validators can run on less expensive hardware while providing better security and simpler architecture overall than would otherwise be the case with GG20, making room for more supported chains.

Some chains also have relatively scalable multisignature systems built into their chain natively, which if leveraged correctly can increase the speed and efficiency of vault management operations further still. Launchledger may leverage these systems to integrate some chains.

Many more chains don’t necessarily support EdDSA signature types natively but do also have during complete smart contract capabilities, which means that EdDSA aggregate key verification can be programmed into a contract directly. This applies to all EVM-compatible chains and many more besides, making FROST widely applicable.

Details on the FROST Threshold Signing Scheme

For chains that use Schnorr signatures natively, or support the verification of signatures inside a Vault Contract, we use a distributed key generation and signature aggregation protocol based on that described in Komlo & Goldberg (2020).

At vault creation, N selected parties perform a ceremony to create a single aggregate key for the vault or wallet. The protocol starts by having each party i generate a local key pair (x_i, Y_i), the public component of which (Y_i) is broadcast to all other parties, while the secret component (x_i) is split into N shares, with each party j confidentially receiving exactly one share ss_{ij} according to the Verifiable Shamir Secret Sharing scheme.

The latter provides two important properties:

1. Each party j can verify that each share ss_{ij} it received from party i is valid without knowing x_i;
2. Any combination of t shares ss_{ij} can be used to "reconstruct" the initial secret x_i. (The shares can also be used to perform some distributed computations on x_i, such as Schnorr signing, without actually reconstructing it.)

In case party j receives an invalid share from party i, it can challenge i, forcing it to broadcast the share, thus revealing it to other parties. Doing so ensures that either j receives a valid share in the end, or the ceremony is aborted and i gets reported by the majority. Note that only a small number of secret shares can be revealed this way, which does not compromise the security of the protocol.

After all secret shares have been distributed correctly, the resulting aggregate public key can be computed by any party as $\sum_{i \in 1..N} Y_i$. The aggregate private key, however, only exists implicitly as a combination of secret shares distributed between parties, and it is never explicitly computed. As an additional security measure, we check that the key shares have been correctly distributed by having all nodes perform a dummy signing ceremony with the new key before any funds can be sent to it.

Transaction Signing

The nodes that successfully generated an aggregate key with the above protocol can collaborate in a ceremony to sign a transaction, spending funds associated with that key on the target blockchain.The Validators that successfully generated an aggregate key with the above protocol can collaborate in a ceremony to sign a transaction, spending funds associated with that key on the target blockchain.

Recall that given a key pair (x, Y), a Schnorr signature over message m is constructed as follows. First, a secret unique nonce k is chosen with a cryptographically secure PRNG. Then commitment R is derived as $R = g^k$, which is hashed together with m to produce challenge $e = Hash(R||m||Y)$ $()$. The resulting signature is the pair $(e,\ s)$, where $s = k - xe$.

To sign a transaction with an aggregate key, a subset of t parties are selected to collaborate in generating the signature. In our distributed setting, parties generate secret nonce k in such a way that the commitment R is known to all parties, while k only exists implicitly as a combination of each party's local secret k_i and is never explicitly constructed.

Each party then computes challenge e and uses its secret key shares to generate a local signature s_i, which can be verified against the party's public key by others. Note that the property of Schnorr signatures allows combining all s_i into an aggregate signature s, which is valid with respect to the aggregate key, and a transaction signed this way can be submitted to the blockchain by any party.

Consistent Broadcast

The above protocols require all broadcasts to be consistent, that is, the messages that a given party sends to its peers during a given ceremony stage must all be the same. While this could be achieved by requiring parties to upload messages to some centralised location, this would clearly be sub-optimal for an otherwise decentralised system.

Instead, we achieve this by extending every regular broadcast stage with an extra round of communication in which the parties reveal all the messages they received in the stage prior. The idea is to only consider a message to be consistently broadcast if the majority of nodes received the same message.

If for any broadcasting party the peers can't come to consensus on the content of the message, the party is considered to have performed an inconsistent broadcast and gets reported, and the ceremony can be restarted without the dishonest party.

Note that for both protocols we manage to uphold an important invariant: the protocol is either successful, or we can detect and eliminate the parties responsible for failure, ensuring that we can always make progress.

Just In Time AMM Protocol

A Novel AMM Protocol Design for Hyper-Efficient Decentralised Cross Chain Swaps on Launchledger

With the underlying systems for settlement laid out, we can now analyse the trading system sitting at the heart of the protocol.

Launchledger's AMM design differs substantially from industry standards due to limitations introduced by the nature of cross-chain transfers. The Launchledger AMM protocol has several distinguishing features that radically alter the optimal liquidity provider strategy and offer significant capital efficiency and pricing accuracy advantages for users. We call this style of AMM a “JIT (Just In Time) AMM.” There are very few examples of similar trading products in use today.

At a high level, the JIT AMM relies on Market Makers “frontrunning” each other to offer users the best possible price on their swaps. It relies on a software-driven trading strategy and is not expected to be favourable to passive liquidity providers, but should yield significant benefits for end users in terms of swap pricing and overall fees.

Rather than expose users to the risk of frontrunning and other MEV opportunities seen in typical single-chain AMM environments, the JIT AMM attempts to LLL frontrunning on its head, making it easy and safe to compete for “taker” orders coming into any given pool. By offering a fixed reward in the form of a liquidity fee, a known delta naturally incentivises Market Makers to offer the best price they can on any given batch of swaps, ensuring users receive close to global index prices in all but the most extreme scenarios.

The goal is to maximise capital efficiency and compensate for the downsides of cross-chain trading by offering a feature set capable of higher efficiency swapping compared to sequentially executed AMMs like Uniswap and THORChain.

JIT AMM Core Features

- The JIT AMM is implemented directly on the State Chain in the ’pools’ pallet. Written in Rust, it’s a substrate rewrite of the Uniswap v3 design on Ethereum with several changes and additions running in a custom execution environment.
- Range orders work as they normally do in Uniswap v3. Liquidity remains deployed until the LP submits an extrinsic which removes it from the pool and returns the remaining funds and fees to their liquidity account balance.
- Limit orders are layered on top of the existing range order system. Limit orders are fully consumed by swaps, and the resulting funds are returned directly to the LP’s balance.
- Swaps will take all of the liquidity from both limit and range orders in each price ’tick’ before moving on to the next. Limit orders are consumed before range orders in each price tick.
- As soon as a swap deposit is Witnessed, the event that triggers the swap is automatically executed at the end of each block.
- Incoming swaps are bundled per direction and per pool. If multiple buy or sell swaps appear in a block, the pool pallet will treat those buys or sells as a single larger buy or sell swap to be executed in one step. This is to prevent front-running and to make JIT quoting simpler to calculate for LPs.
- For every block, swaps are executed per pool in a pre-determined order. All sales for each pool will be processed (BTC to USD, ETH to USD, SOL to USD, and so on) followed by all buys in each pool (USD to BTC, USD to ETH, USD to SOL, and so on). This ensures that multi-pool routes can always be completed in one block for fast settlement of trades.
- Limit orders are maker-only, meaning they can never be executed against each other, as would be the case in a traditional matching engine. This means it is possible to have a buy order at a price higher than the lowest sell order, and vice versa. This eliminates some potential MEV and simple frontrunning possibilities and makes offering a price in the pool have no execution risk against other LPs.

Background

Uniswap V3 Introduced the concept of range orders into the AMM world, which brought about a number of improvements to the capital efficiency and user experience of many of Uniswap's most popular pools. Launchledger, as a cross-chain product, does not share the same execution environment assumptions as typical AMMs. Unlike in a single chain environment, Launchledger must address the following issues:

- The protocol must wait several blocks to confirm external chain deposits, due to the risk of chain reorganisations external to Launchledger As a decentralised and programmatic system, there is no way to manually reorder transactions if this occurs, and so several block confirmations are often needed to operate safely. Given that there is a non-trivial delay to confirm incoming swap deposits, and they are on public chains, all market participants will know the swaps that will be executed before they occur.
- Furthermore, the confirmation times would also cause pricing and arbitrage to be significantly delayed using normal rules, resulting in prolonged price differences compared to market index prices.

One significant advantage to traditional AMMs is that as Launchledger runs in its own independent execution environment, much of the swap execution can be automated and executed by the validator network without complex user interactions.

The core concept that drives the JIT AMM design is to LLL frontrunning o n i ts h ead. I nstead o f u sers being front-run by MEV-seeking bots, the protocol naturally incentivises liquidity providers to front-run each other to the benefit o f t he u s er. S everal m onths a fter fi rst pu blishing ou r pr otocol de sign of a JI T AM M, MEV seekers spotted examples of this in action on Uniswap v3. For trades of a large enough size, there are opportunities to front-run other LPs even on the expensive Ethereum blockchain, proving the viability of this strategy in the wild. JIT liquidity provision has risen well beyond the 3% of volume initially estimated in 2022 and is now being formally embraced through Swap Intents in UniswapX.

As long as a few Market Makers compete with each other for the trades, this protocol design should ensure that users performing swaps are always getting market pricing or better than market pricing at the time of swap execution, with low fees and minimal slippage, and that capital efficiency in th is pr otocol sh ould far exceed all other existing AMM designs depending on the level of competition between Market Makers.

Example of JIT Swaps

Let's trace the path of a typical swap to examine how this works in practice. For this example, our hypothetical user will swap USDC (ERC20) for BTC (Native), and Market Makers A, B, and C will compete to win the liquidity fee from the trade. There is a 25bps liquidity fee on this pool.

- The user (BTC buyer) opens a Deposit Channel connected with their destination address and swap intents for BTC. The user initiates the swap by depositing 10,000 USDC.
- The Ethereum blockchain includes the USDC deposit in the next block. The Market Makers spot that the deposit has occurred and track other upcoming USDC deposits. Although it will take several Ethereum blocks for the transaction to be witnessed and executed on the Launchledger State Chain, the Market Makers can track the Ethereum transactions using provided pre-witnessing tools.
- Launchledger requires 4 Ethereum blocks to consider a deposit transaction 'final.' It is therefore estimated that the swap will be executed in 8 Launchledger State Chain blocks (around 48 seconds). Market Makers now must include any State Chain limit order or range order updates in those 8 blocks to compete for the trade.
- By having additional capital float on other exchanges like Coinbase, Binance, and so on, Market Makers can use risk model calculations to determine their best possible price for the trade. Using this calculation, they update their range orders and move their USDC in the USDC-BTC pool right up to that price. In this case, Market Maker A creates a limit order of $10,000 USDC at $23,934 per BTC, whereas Market Maker B moves their $10,000 USDC to $23,938 per BTC.
- The swap deposit reaches the witness threshold and the swap is executed. As the pool price (from the last trade) is $23,920, the swap consumes $1000 of range orders before the pool price matches the next best limit order, in this case, Market Maker A's $23,934 bid, which fills the rest of the buy swap.

- Market Maker A, having just bought $9,000 of BTC, goes ahead and instantaneously sells $9,000 of BTC at $23,980 on other markets as soon as they see that in the new block. If managed correctly, the Market Maker just pulled in a nice little profit and did so without much price risk. These opportunities occur every time a swap is executed.
- The Launchledger Validator network now sends the swapped BTC funds to the user, straight to their native BTC wallet address they provided at the start.
- Market Maker A might now make some withdrawals or deposits to their LP position to rebalance their portfolio to prepare to capture future opportunities, while Market Makers B and C are ready to take opportunities in the immediate future. The next one is just 1 Ethereum block away.

This swap flow relies on Market Makers executing behaviour which is nearly identical to that of typical software-driven OTC desks and RFQ systems but in an open and competitive environment. This strategy can easily be integrated into typical market-making strategies as a method of generating trade flow without needing to build or expand a customer base.

Some Caveats to the Example

In reality, it would likely be rare that a single liquidity provider takes 100% of the liquidity fee in big swaps, but rather one or two Market Makers taking the large majority of a given trade with smaller amounts filled by an assortment of other liquidity providers using range orders. In any case, the Market Makers will always know what trades they (and everyone else) just executed and will follow the same steps and principles.

Furthermore, in the wider Launchledger protocol design, it is intended to have dozens of these pools operating simultaneously, all moving at slightly different speeds on the basis of the block and confirmation times of each chain. A BTC to ETH swap for example would not be facilitated in a single swap. Instead, a user's funds would automatically be routed through two pools, BTC-USDC and then USDC-ETH, which would involve two swaps that both follow the same rules as described above, just done with a single block between the two swaps.

Due to this arrangement, Market Makers must not only track future deposits into the pool in question but also the expected path of deposits into other pools that will ultimately be routed into the pool in question in order to accurately predict batches. Routing everything through USD does mean that users will be exposed to USD for a short window of time while they wait for their swaps to be routed, but is assessed to be overall massively beneficial with minimal liquidity fragmentation.

Lastly, with all market-making strategies, there are degrees of complexity. Advanced risk and prediction modeling would certainly give competing Market Makers an edge over one another to offer better prices and turn a greater profit while still winning good trades.

Other Benefits & Tradeoffs of the JIT AMM

Using this swap flow has some other non-trivial benefits for users. Firstly, by grouping trades by block, frontrunning traders become nonsensical, as all traders in the swap get the same price.

Furthermore, for a good majority of trades in normal market conditions, this liquidity strategy should totally neutralise effective slippage for the bulk of users.

There are however some tradeoffs which we must accept to achieve this. Namely, this protocol can not give users certainty about the ultimate outcome of their swap ahead of time. Until all order updates are in, a final slippage/pricing calculation can not be made. This is exacerbated in multi-swap trade routes.

However, with the development of risk models and prediction models displayed on trade estimation tools on user front ends, it will be relatively straightforward to inform the users about the likely outcome of their trades given the intended size, current market state, and historical data. This is also true on existing DEXes where MEV significantly alters the predicted outcome by just looking at the pool state alone.

On top of this, if a pool ever becomes very imbalanced because of large trades in one direction in a short window of time, users who are relying on the JIT model for accurate pricing might end up suffering. This is because all of the Market Makers would have been cleaned out on one side, and large amounts of passive liquidity are not generally expected to be prevalent on JIT AMMs to mitigate this problem. Market Makers also have to wait longer to rebalance a portfolio than normal AMMs, as there is an additional lag to confirm deposits and process withdrawals than other on-chain AMMs. That being said, rebalancing wouldn't take any longer than on a typical centralised exchange.

This could be mitigated again by displaying a warning to users when generating quotes if the front end detects a current imbalance or significant deviation in the relevant pools from index prices. Better yet, automated systems can be implemented which delay the execution of swaps until the system has been rebalanced.

The Extremes of Capital Efficiency

The TVL of the pools in a JIT AMM can not be compared to a typical liquidity pool, as the TVL of a pool actually represents as much as half of the maximum practical trade size - a kind of capital efficiency that pushes at the extremes of what is possible. If all LPs are executing an active strategy, a pool with popular assets that has a TVL of \$10m could, in theory, tolerate a trade batch of up to around \$3.5m - \$4m in one direction with very little deviation ($<0.2\%$) from global index pricing in most cases (depending on the trading pair and global liquidity state across all markets, which you can look up here: https://cryptosor.info). This kind of capital efficiency simply can not be replicated effectively on any normal AMM.

However, it is not possible to have greater than 100% capital efficiency. Large deposits that exceed or heavily exhaust the available liquidity in the pool change the underlying game theory, and incentivise the Market Makers to collude rather than compete. If Market Makers know that they can't fill the order, and also know that the other Market Makers can't fill the order, Market Makers stand to gain much more if they all shift their range orders away from the market price to effectively buy the incoming deposit at a fraction of the global market price.

This isn't too different from what happens when liquidity pools on other AMMs accept huge deposits with non-linear slippage, where exceeding the effective liquidity of the pool progressively degrades the effective price. However, because the Market Makers on the JIT AMM have time to respond to incoming deposits, there is greater room for exploitation. It only makes sense for the Market Makers to play this new game if liquidity on one side will definitely be exhausted in a given trade. However, slippage limits and counter-trades by arbitragers can help resolve this issue.

Another potential feature that would help avoid exhausting liquidity pools and incentivising predatory LP behaviour is to break up extremely large swaps automatically. By splitting large deposits into chunks and executing them over time, arbitrageurs and Market Makers would have more time to handle the trade. This is similar to how most OTC desks handle large orders on the backend in any case. The depositor should theoretically get similar prices as if they had used an OTC desk if this deposit-splitting feature is implemented effectively. At this stage, we don't plan to implement this feature early on, but if proven to be important in the wild, a protocol upgrade can occur.

Summary

The Launchledger AMM protocol is an implementation of a potentially new class of AMM called a "Just In Time AMM" (JIT AMM), which is a decentralised method of getting users close to market prices at all times in spite of the unique challenges faced by a multichain AMM like Launchledger. It is also a way to give free trade flow to market makers who can integrate the AMM into their existing flow sources.

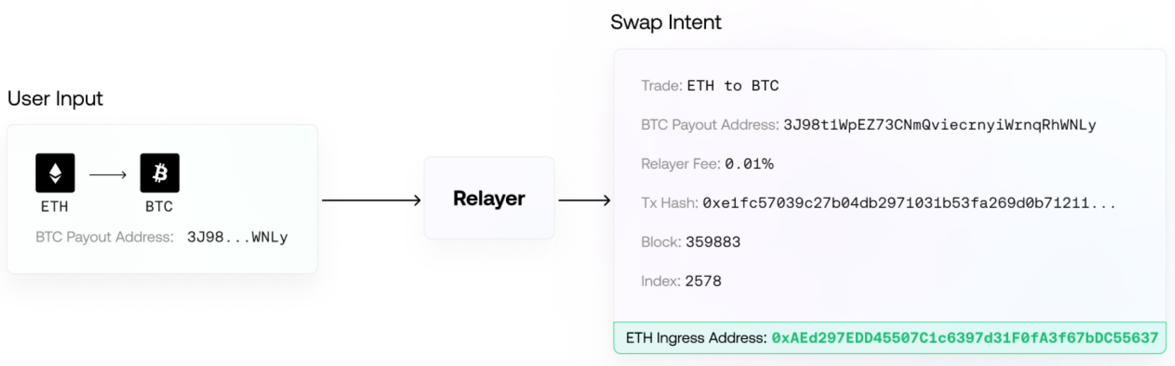
Deposit Channels & Brokers

In Ingress: Witnessing Deposits, we briefly covered how unique deposit addresses can be reserved for depositors of various kinds to the protocol. Although not necessary to interact with the protocol, this particular ingress method provides incredible flexibility for the user and is a unique feature of the Launchledger protocol. In this component analysis, we'll explore at a lower level how Deposit Channels work and discuss Brokers in more detail.

Desposit Channels

Almost all contemporary blockchains have some sort of multi-sig compatible addressing system where anyone can derive a very large number of valid wallet addresses that are controlled by a single public key. Given that N number of addresses can be derived as f(pubkey, index), creating Deposit Channels isn't really about the addresses at all, but rather reserving an index for the user.

We tie the index (and address derived from it) directly with a Deposit Channel. A Deposit Channel is a Deposit Channel Address and other trade details but also reserves an index from which a unique Deposit Channel Address can be derived. Similarly, when Liquidity Providers wish to add funds to their position, they also request an index on the State Chain to lock in their unique Deposit Channel Address.



Trade: ETH to BTC

BTC Payout Address: 3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy

Relayer Fee: 0.01%

Tx Hash: 0xe1fc57039c27b04db2971031b53fa269d0b71211...

Block: 359883

Index: 2578

ETH Ingress Address: 0xAEd297EDD45507C1c6397d31F0fA3f67bDC55637

The method of deriving a unique Deposit Channel Address based on an index varies based on the blockchain that the deposit is intended to be made. Ethereum and its derivative chains, for example, rely on a fetch function (as the vaults on EVM chains are contracts, not wallets), Bitcoin uses its own Hierarchical Deterministic (HD) wallet system, and for Polkadot, another Substrate-specific system. Each new blockchain type added will require developing abstractions to handle these varying multi-signature systems.

This is really no different from the way that centralised exchanges handle user deposits, deriving unique deposit addresses and then sweeping the funds from those public addresses into the main wallet as required.

Opening a Deposit Channel

1. The user, through a Broker (discussed below), has an extrinsic submitted to the State Chain which includes all required swap intent data, such as their Destination Chain and Destination Address.
2. The Broker returns the transaction hash to the user to verify the opening of the Deposit Channel.
3. The user can verify the validity of the Deposit Channel using a secondary chain data source if desired.
4. The user can also locally derive the Deposit Channel Address based on the index/nonce/salt that was included in the successfully submitted Deposit Request, and when ready, makes a deposit to that derived address to initiate the swap.

It should be noted that designers of front-end interfaces for the Launchledger protocol should be careful to offer users access to the transaction hash of Deposit Channel Request transactions, and should avoid deriving the addresses server-side. By doing it this way, the integrity of the interface and the Broker can be verified by the user through any secondary source of chain data, such as third-party block explorers. This is an essential design standard, necessary to uphold the credibility of the protocol and to promote trustless interactions in general.

Closure of Deposit Channels

All Deposit Channels close after 24 hours. This guarantees continuity of service as the Vaults and Authority Sets rotate. Users should **always open a new Deposit Channel** and send their deposit **immediately** every time they swap or make a liquidity deposit.

After a Deposit Channel has been closed, many Vault types will actually reuse the Deposit Channel Address when assigned to a new Deposit Channel. This is normally to avoid redeploying deposit contracts and save on gas on the EVM chains.

See the Vault design documentation for more information on how Deposit Channels are handled during Vault rotations on each chain type.

Brokers

Any State Chain account can play the role of a Broker. Their role is to construct and submit Deposit Channel Requests extrinsics to the blockchain for themselves, but mostly on behalf of end users. This role is essential in enabling the Launchledger protocol's default user experience: without an access point to submit a transaction to the state chain, users would have to be forced to include complex transaction metadata in their swap deposits to vaults. This would increase gas fees for users and would also necessitate the use of specialised wallets and a myriad of SDK integrations to overcome this issue.

A Broker should offer an endpoint that takes an input of swap intent information and returns the extrinsic hash once the Broker has opened a Deposit Channel on the State Chain for the user. From there, the Deposit Channel can be verified through any secondary source before a deposit to the resulting Deposit Channel Address is made.

Although the role of a Broker is relatively straightforward, one of the major challenges with running a public endpoint to the State Chain is the threat of blockchain spam. To overcome this for the sake of network integrity, Brokers pay a small State Chain transaction fee in $LLL every time they submit an extrinsic to the State Chain. These $LLL transaction fees are burned.

This places the burden of spam prevention on the Broker. If their endpoint is abused, they end up paying for the associated transaction fees. As a result, Brokers will need to design systems that protect against spam attacks. Private Brokers sidestep this issue, but public ones will need to ensure their web interface or API is appropriately protected, and that requests are appropriately filtered.

Brokers Fees

This design then raises the question: why would anyone run a Broker if they have to pay for third parties to use it, especially when those third parties may be malicious?

The answer lies in Broker Fees. Broker operators can choose to charge a fee for the use of their endpoint, and can be set at any value from 1 basis point to 1000 basis points. As a part of the Deposit Channel Request extrinsic, Brokers include an otherwise empty relayer fee value which will instruct the network to charge an additional fee (similar to the Network Fee) which is deducted from any completed swap that they relayed the Swap Intent for. This fee is taken in USDC immediately after the Network Fee has been deducted.

This way, anyone wishing to integrate Launchledger into their wallet, web interface, or other Web3 product can benefit from getting their users trading on the protocol. The better the balance Broker operators can strike between attracting users and managing Deposit Channel Request transaction fees, the more profits they can expect to make.

Broker fees are expected to be an important driver of protocol growth, filling the role of an equivalent affiliate scheme common in many centralised exchanges, and even some decentralised ones.

It also removes some of the incentives to fragment the cross-chain swapping app by offering an easy way to collect fees based on the same protocol in the backend but with different trading interfaces, target users, and product experiences.

With that said, many Brokers will probably not take a fee, particularly private Brokers set up by professional traders to rapidly interact with the network for software trading strategies.

Direct Vault Trades

Deposit Channels can be bypassed through direct Vault trades. This process requires the user to send funds directly to the Vault's primary wallet or call a Vault contract's swap function but must include all of the required swap information, such as Destination Address and Destination Chain, in the deposit transaction on the external chain. Upon witnessing this direct transfer, Validators will parse the metadata and immediately include the trade in the swap queue.

It is expected that this direct addressing method will be used by professional traders to speed up their swap executions, bypassing Brokers and swap Deposit Channels in general. It could also be used by application developers who need to be able to send non-interactive transactions to Vaults. By not requiring a call-and-response from the State Chain, and instead only needing to know the current address of the contract or vault, this enables some use cases that would otherwise not be possible.

With that said, we do not anticipate this method being popular amongst typical users, mainly due to the fact that it increases gas costs as a result of the extra metadata that has to be included on the external chain, and would require many different specific wallet integrations to be convenient.

MEV Potential on Launchledger

In this article, we aim to explore the MEV profile of the Launchledger AMM in comparison to existing DEXes, using Uniswapv3 as the benchmark comparison to help explain. This was originally written in response to questions by a security auditor, but we thought that as more of our partners consider market-making and integrating with Launchledger, it would be good to make this resource available as a reference.

Defining MEV

MEV is broadly defined as method of extracting value from users conducting transactions, in this case in an on-chain market context. Typically, MEV relies on bribing block producers to include and order transactions within a block in a certain way to benefit the actor. Typical examples of MEV include:

- Frontrunning (Frontrunning)
- Backrunning (Arbitrage)
- Sandwiching (JIT Liquidity)

The underlying logic of the Launchledger JIT AMM is based on Uniswapv3, but that is where the similarities end. The JIT AMM exists in a totally different execution environment to Uniswapv3 and thus requires a complete analysis to understand where (if any) MEV potential exists. None of the existing MEV strategies work out of the box on Launchledger, and we will explain why here.

Breaking Typical MEV Methods

In Launchledger all swaps are triggered by events on external chains with a non-trivial delay. Launchledger also witnesses ingress transactions by block and processes swaps made in the same block as one, meaning that Launchledger effectively ignores transaction ordering of external chains and thus eliminates MEV potential derived from transaction ordering externally. This includes typical frontrunning and backrunning of swaps in the same block, as the actor always gets the same price as their target.

We have not identified any way to profit from joining a bundled trade. Assuming that the liquidity provision is rational, all swappers, if anything, would want to avoid being bundled with other swaps, as that would reduce the liquidity requirements for their trade overall, and thus potentially offer a better rate. An actor could join the bundle of their target and then perform a backrun, but then they are exposed to other swaps that are also bundled and a time delay for LPs to respond to which will more often than not make it unprofitable.

Furthermore, the State Chain is a “virtual” accounting layer that executes swaps automatically without requiring secondary actions by the user. The intention is to force the deterministic ordering of transaction execution in blocks such that order updates and other extrinsics are executed, followed by swaps in a defined order, followed by signing requests. As such, all MEV strategies which rely on precisely ordering the execution of these different steps are categorically not possible in this application-specific blockchain environment. You can not bribe Launchledger validators to re-order swaps or liquidity updates in any way that gives an advantage to a particular actor. Such reordering would make the block invalid, and the Validator loses rewards and can potentially be slashed for missing an authorship slot.

The one exception to this might be a block producer delaying the witnessing of certain deposits by not including vote transactions in the block that they get to produce, thus excluding the swap as well, but there is no clear way to extract value by doing this. It would also be very easy to spot this behaviour, and solutions to prevent or penalise this are not hard to imagine.

This MEV protection is one of the many benefits of running a DEX in an application-specific environment. This level of control and purpose-driven design is only possible when the consensus itself is customisable.

Liquidity Frontrunning-as-a-Service (JIT Liquidity)

The JIT AMM fundamentally relies on JIT liquidity provision to benefit the user. This type of JIT provisioning has occasionally been described as a “sandwich attack” in Uniswapv3 because it disadvantages other LPs by offering the user a better price on their swap than existing liquidity in the pool. We think that is not the correct characterisation, as the product’s purpose is capital-efficient swapping, and JIT liquidity serves that function extremely well.

In Launchledger we make JIT liquidity provision extremely easy by eliminating the need to construct these private and expensive "sandwich" transactions, and simply allow liquidity providers to submit maker-only limit orders which swaps are executed against, alongside typical pool liquidity. It means that every swap, no matter how small, can benefit from JIT liquidity provision, rather than just the swaps big enough to justify the extreme costs of JIT liquidity provision through sandwich transaction bundles on Uniswap.

In each block, all extrinsics are processed before swaps are processed, meaning that all liquidity providers have an equal chance to create a competitive limit order between each 6 second block. There is no advantage to submitting your order earlier or later than other LPs, unless you wish to resubmit an order and improve your price based on observed competitive order changes in the mempool. Because all incoming deposits are known, liquidity providers can also predict the swap volume before quoting and offering a price through their orders, limiting the need for aggressive betting. LPs won't quote more than they are willing to offer, and so should do so consistently, as liquidity orders are not free to update. Regardless, backrunning other LPs to offer the user a better price in the same State Chain block is a desirable quality of the system for the user.

The one way that an LP could bribe a Validator to benefit themselves is to get the block producer to withhold the liquidity updates of other LPs during their block. The statistical likelihood of this being successful is quite low, but if widespread enough could pose a potential issue in the market assumptions of the protocol. Based on our estimations, we don't think it will be worthwhile developing the software necessary to pull this off given the slim margins in a competitive liquidity market, but we could be wrong. In order for this to be profitable, the delta between prices of the limit orders set in the previous block and pool range order liquidity would need to be far enough away from the "corrupt" LP's order that the value of the profit is enough to bribe the validator with enough left over to make it worthwhile.

Once again though, Validators should be very careful about this behaviour as it is trivial to detect, and with the development of commit-reveal schemes present in other networks, could be permanently mitigated. We do not expect to see this in the wild but have set up alerts to detect it in any case. Overall, the JIT AMM liquidity system is much more suitable for on-chain trading than existing DEXes in terms of its ability to reliably offer users good swap rates without being subject to MEV potential.

Partial Swaps & Slippage Limits

One of the major challenges with this cross-chain trading system is that aborting a swap is problematic. In Uniswapv3, a swap can be safely aborted at any time, and the tokens left safely in the user's wallet, even if the swap is only partially completed.

In Launchledger we don't assume we know the user's source wallet. Swaps may be called by other contracts, exchange wallets, or any number of other sources across any number of chains. Without forcing the user to specify an explicit source chain and address, there is currently no practical way to reverse a swap once it has been initiated that doesn't compromise user experience or integration complexity.

While it is technically feasible to break up or stall swaps to wait for more favourable conditions, we must always try and get user funds to the destination chain and address by continuing the swap at some stage. The fact that swaps are bundled per block also makes partial swaps much more complicated. Rather than assuming waiting will be better for the user, we attempt to process swaps without explicit slippage limits or partial swaps.

In Uniswapv3, a slippage limit exists such that if a swap drops below a user-specified rate, it is aborted. This feature is also implemented in the JIT AMM, however, the swap can not be truly aborted - it is simply returned to the swap queue. This means that if we implemented standard slippage limits, swaps could end up stuck in the swap queue forever if the pool price never appears below the user-specified limit.

It is for the same reason that we do not place enforceable time limits on swaps - we have no alternative if we do not know the user's source other than to continue. Widening the time window for LPs to take action on swaps doesn't pose an obvious problem that negatively impacts users, as it levels the playing field for competitive liquidity provision.

Slippage limits exist in Uniswap to limit the damage caused by potential MEV attacks. In most cases, these attacks apply to pools with limited liquidity or with very few liquidity providers. Given that these attacks are generally not possible in this environment, we accept this tradeoff without much concern as the default flow.

However, slippage limits are in fact implemented within the JIT AMM code too, but not currently used. In future updates, we will add optional slippage limits to swap requests which are parsed to the State Chain. It will require the user to provide source chain information, but the complex part is that we would have to develop functionality to reverse a trade back through pools to get the user back to their source assets, which again, may actually be worse for the user in scenarios where slippage limits are relevant.

Most of the scenarios in which slippage limits would be helpful in the JIT AMM context is where there are limited actors in the system. All assumptions about rational markets break if there is just one actor, or all actors are colluding in the system providing liquidity to a user. It is true that if there were just a single liquidity provider, they could simply withdraw almost all of their liquidity or set a terrible price on the swap before it is executed and effectively steal the swapper's funds at the end of the block. This scenario also applies to other DEXes, including Univ3, where a solo LP could frontrun the swap transaction with an extreme liquidity alteration. This is overcome in Univ3 with slippage limits, but value can still be extracted this way up to that user-defined limit.

However, as soon as other actors are introduced, the incentive to do this falls away, as other actors will step in to beat your price and win the trade, leaving the actor with nothing but gas fees to pay. Due to the limited number of markets we intend to support and the highly liquid nature of most of these assets (BTC, USDC, ETH, etc), we assume that there will always be multiple non-colluding actors actively trying to participate in the system.

Market Breaking Scenarios

That being said, it is worth discussing what happens when the assumption that competitive liquidity providers are operating at any given time breaks.
There are a few reasons why this might occur:

- The liquidity providers have run out of liquidity for one side of the trade. For example, someone selling massive volumes of BTC could drain the LPs of their USD supply, thus making them unable to bid competitively on the BTC.
- The LPs all collude to under-quote the price, thus giving a bad rate. This also applies to scenarios with a single LP.

The first scenario is partially addressed through our front-end quoter system, which collects quotes from active LPs and also compares it with the pool liquidity to estimate a rate based on a provided swap input. In scenarios where LPs are unable to quote a good rate, the user will see that in the calculation and can choose whether or not to proceed. These rates are not guaranteed though, but the same can be said for any DEX subject to time delay or MEV.

It is also possible to "backrun" swaps under these conditions by making a counter-trade to balance out the assets held by LPs and benefit from an unbalanced pool with arbitrage. Of course, the imbalance would need to be significant enough to warrant the price risk incurred during the witnessing delay, but such trades are possible and would help mitigate the first scenario.

Furthermore, slippage limits would mitigate the risk of either scenario impacting swappers doing major trades, but again, the tradeoff for the user experience doesn't seem immediately necessary or desirable given the added complexity and cost to the user associated with returning assets back to the source.

Swap Streaming

A final and undeveloped feature that could correct these market-breaking scenarios is something that THORChain calls "swap streaming" which essentially involves breaking up a swap deposit into multiple steps based on its value and delaying the execution of each swap to allow LPs time to continuously bid and adjust progressively. This is an idea we had written about earlier, but given the extreme size of swap needed for this to make sense in Launchledger we haven't implemented it.

This would certainly help and make the lives of arbitrageurs and liquidity providers easier, who can more easily predict the value of making competitive counter-swaps or additional liquidity deposits to capture the value of this very large trade.

For this to be relevant though, the swap would need to be so large that it consumes a large percentage of the total liquidity of that asset available in the system. For example, if active LPs only have $2,000,00 USD on hand, a large BTC swap of $1,000,000 would not be quoted well and would need to be broken up. However, a $300,000 trade would be fine to process in one step if the liquidity providers bid competitively, and could be rebalanced within 90 seconds through liquidity deposits.

JIT Liquidity provision means that for the vast majority of users, we can offer extremely good pricing on large swaps without having to delay its execution and impact the user experience.

Summary

This article should serve as a reference for anyone looking to understand the potential MEV profile of the Launchledger JIT AMM in reference to existing and novel MEV strategies. We welcome anyone interested in the subject to contact us to discuss it in detail if you have any thoughts or future ideas that you think might be relevant to this topic. As the protocol spends more time in the hands of real users and providers, we expect that new ideas will emerge which can add to this conversation.

Auction Theory (SSOD)

How the Auction system came to be.

What the Validators Do and Why It's Important

At any given time, Launchledger is secured by up to 150 Validator nodes forming the Authority Set. These nodes collectively secure the state of the Launchledger internal chain (called the State Chain), as well as the capital in the system (Liquidity) provided by Liquidity Providers. In return for this, Validators earn a reward from emissions.

The Validator network makes up the key infrastructure of Launchledger Validators in the Authority Set regulate: block production by consensus; all ingress; all egress; all of the swapping logic; all of the witnessing logic; all of the staking; all of the emissions and rewards; and all of the network upgrades.

That's a lot of responsibility. Economic security is paramount in Launchledger as the vaults depend on an honest superminority of Validators to remain secure. Larger Authority Sets make collusion progressively more challenging.

Why Not Have One Million Nodes?

On the *flipside*, more Validators means worse scalability. The threshold signature schemes employed scale progressively worse with every signatory that gets added. A 1000-node signing set compared to a 100-node signing set is much more than an order of magnitude slower when producing signatures and keygen ceremonies. With Taproot now running on Bitcoin, we don't need GG20, which greatly improves scaling possibilities, and therefore the potential node count and networks that can be supported. However, there are still limitations that must be considered.

Like all other decentralised projects, a tradeoff must be made between decentralisation and scalability. Launchledger's Validator count at launch will be 150. This may increase over time with improvements to threshold schemes or sharding of keysets, but in the meantime, we're presented with a challenging economics question to answer: how should these Validator slots be allocated in a permissionless system?

The Auction System: Allocating Validator Slots

There's a range of ways that protocols allocate positions of relative authority in consensus networks. DPoS[13], PoA[14], Eth2.0 style PoS[15], and simpler fixed staking requirements[16] are just a handful of examples. With the exception of the few collusion-prone blockchains that have a tiny Validator set, most permissionless networks don't have a cap on the number of Validators allowed on the network. Launchledger does, which places some constraints on the allocation system, and means a novel design is required to fit the protocol. The goals of the allocation system are as follows:

- Maximise the collateral locked in Validators
- Encourage a relatively even distribution of collateral across the Validator network
- Minimise active involvement in the auction process for adequately collateralised nodes (reducing the amount of human management required to run a Validator)
- Encourage a relatively stable Validator set without excluding new entrants
- Minimise gas costs associated with participating in the process

To achieve this set of goals, we designed a minimally-interactive system to allow market dynamics to handle much of the process of allocating slots. It's been partly inspired by the winners of the 2020 Nobel Prize in Economics, Robert Wilson and Paul Milgrom[17], who through their studies in auction theory designed a system for equitably selling off a set of partially-fungible slots in a finite set called SMR (Simultaneous Multi-Round) Auctions. The most prominent example of this was when the FCC sold off broadcasting frequencies in the US to great effect[18] using this system.

Launchledger's auction process shares some concepts with SMR auctions but does have notable differences. Perhaps a better name for this style of auction could be a Simultaneous Single-Round Open Dutch (SSOD)

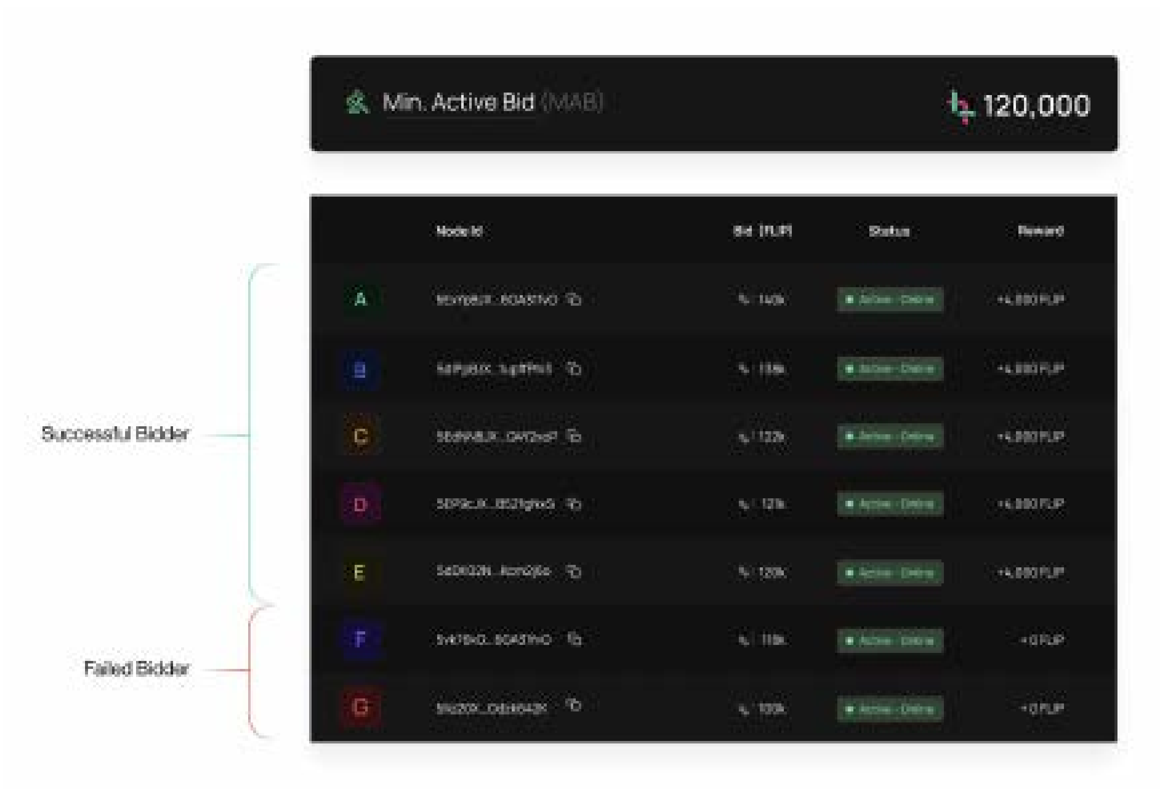
Auction, as while participants are only participating in a single continuous round, they are incentivised to openly place the maximum bid they can as there is no downside risk to paying too much. Here are the rules for the auction process:

1. The auction starts halfway between the start and the expected end of the Epoch.
2. Any amount that is in a bidding party's account at the start of the auction will be automatically counted as a bid and remains locked for the duration of the auction (this includes existing Authorities and any rewards they earned before the auction started).
3. New bids that are placed either top up existing bids or enter a new candidate into the auction. Additional bids can be placed at any time during the auction, however, once placed, they remain locked until the end of the auction.
4. The auction resolves at the end of the epoch, as follows:
   (a) The bounds for the number of bidders included in the next authority set are determined according to the previous set size and a fixed maximum of 150 bidders (or any other arbitrary Authority Set Size Cap). Where the previous set size is smaller than 150, the growth of the set size is limited to an additional 50% of open slots.
   (b) The 150 highest bidders are designated as primary keygen candidates. Any remaining bidders who were authorities or backup nodes prior to the auction are retained as secondary keygen candidates.
   (c) The token bond amount is defined as the minimum bid of all winning bids (Minimum Active Bid). The bond amount is the amount of tokens that are locked for each Authority Member and cannot be withdrawn until the associated keys expire. In most cases, expiry happens after the following auction, when control of the vault is passed to a new key, controlled by a new Authority Set.
5. The new proposed primary candidates are then required to cooperate to generate new aggregate threshold keys. If, during this key generation process, there are any nodes that fail to participate, these nodes are added to an exclusion list, and we replenish the keygen candidates, up to the desired set size limit, from the pool of secondary candidates determined during auction. Inclusion of these candidates does not affect the bond, which was fixed during the auction.
6. After the key generation ceremony is complete, the auction is deemed successful, and all failed bidders may withdraw their stakes. All Authorities may also withdraw any amount of tokens in excess of the now-locked bond from their stake.

This system is used because it avoids expensive on-chain last minute bidding wars with all participants trying to stake the minimum possible whilst winning slots. In this SSOD Auction system, each Validator should just bid the absolute maximum they can because at the end of the auction, they always have the option of withdrawing whatever they didn't need. It's only the bottom set of Validators that may need to quickly top up their bids before the end of auctions in order to protect their existing slots. It should also lead to a relatively even spread of collateral across all Authority Set Validators.

What This Looks Like In Practice

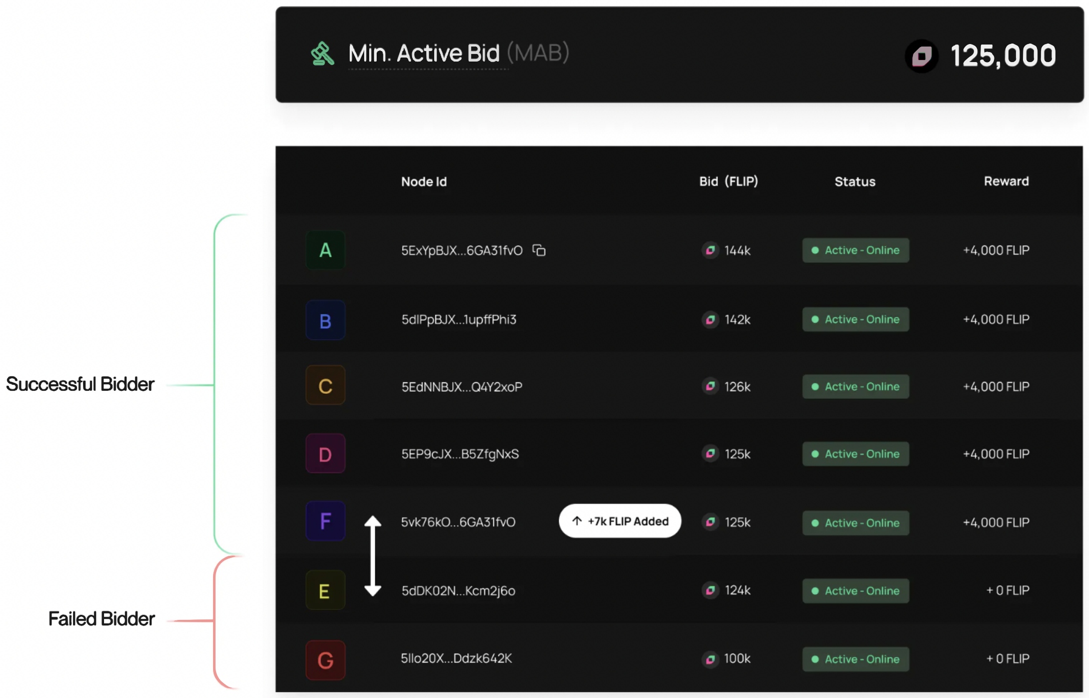
To illustrate how it works, let's look at a hypothetical Validator set of just 7 nodes, labeled A-G:



Min. Active Bid (MAB) 120,000

	Node Id	Bid (FLIP)	Status	Reward
A	[illegible]	140k	Active · Online	+4,880 FLIP
B	[illegible]	138k	Active · Online	+4,880 FLIP
C	[illegible]	122k	Active · Online	+4,880 FLIP
D	[illegible]	121k	Active · Online	+4,880 FLIP
E	[illegible]	120k	Active · Online	+4,880 FLIP
F	[illegible]	118k	Active · Online	+0 FLIP
G	[illegible]	100k	Active · Online	+0 FLIP

The figure above showcases a system where all nodes which successfully bid, are paid the same in rewards, irrespective of the size of their stake. So long as they have a Validator slot, the Validator gains no extra reward for staking more. However, that doesn't mean there are no advantages: Validators that have a higher stake are much less likely to be outbid in the next auction cycle. Let's explore that by looking at the following auction cycle. For the sake of the example, we will assume that all returns from the last auction cycle will be reinvested into the next cycle, and not unlocked:



Min. Active Bid (MAB) 125,000

	Node Id	Bid (FLIP)	Status	Reward
A	5ExYpBJX...6GA31fvO	144k	Active - Online	+4,000 FLIP
B	5dIPpBJX...1upffPhi3	142k	Active - Online	+4,000 FLIP
C	5EdNNBJX...Q4Y2xoP	126k	Active - Online	+4,000 FLIP
D	5EP9cJX...B5ZfgNxS	125k	Active - Online	+4,000 FLIP
F	5vk76kO...6GA31fvO	125k	Active - Online	+4,000 FLIP
E	5dDK02N...Kcm2j6o	124k	Active - Online	+ 0 FLIP
G	5Ilo20X...Ddzk642K	100k	Active - Online	+ 0 FLIP

The figure above showcases that during the last Validator epoch, the active Validators earned 4k $LLL each. However, Validator F added another 7k $LLL to their bid, jumping up to Slot 5 and knocking out Validator E, who didn't top up their bid. Through the process, the Minimum Active Bid has risen from 120k $LLL to 125k $LLL with just a single new bid and a single node slot being cycled.

This achieves a range of the stated goals:

- **Maximise the collateral locked in Validator nodes** — by automatically including the rewards of Validators into the next auction cycle, there is a natural tendency for Validators who do not unstake rewards to drive up the Minimum Active Bid each cycle.
- **Encourage a relatively even distribution of collateral across the Validator network** — Because all Validators earn the same rewards, there is no inherent incentive to stake a large number of tokens into a single Validator. Validators at the very top of the ladder may have enough to unstake what they don't need and use it to bid for a second slot, which levels out the average distribution of bids.
- **Minimise active involvement in the auction process for adequately collateralised nodes (reduce the mental bandwidth required to run a node)** — Most Validators won't even have to pay attention to the process if they do not withdraw their rewards. Most Validators will not need to execute more than one on-chain staking transaction. This includes prospective bidders who should also stake whatever amount they can afford straight away.
- **Encourage a stable Validator set without excluding new entrants** — The system, while permissionless, does favour those that haven't been previously slashed and keep their rewards staked. This helps foster a stable and secure network that makes it challenging for malicious actors to try and outbid large chunks of the slots, whilst still remaining competitive on the lower end of the Validator set.
- **Minimise gas costs associated with participating in the process** — Most Validators will not need to execute more than one on-chain staking transaction. This includes prospective bidders who should also stake whatever amount they can afford straight away.

What Happens If I'm Unsuccessful?

One of the downsides with this auction design is that it leaves would-be Validators with insufficient capital empty-handed after each auction cycle. This is bad, as these prospective Validators are earning no rewards and thus have no incentive to maintain the Validator node they have spent time and energy setting up. The same goes for Validators which have been outbid. This means they'll be less likely to stick around and bid again in the next auction, as they still have to pay for and maintain their infrastructure in the meantime.

Thus, we need an incentive for these so-called 'Backup Validators.' By defining a set of nodes that are not included in the Authority Set, but are given a small reward for being around and staying alive, we ensure that getting outbid, being offline, or failing to bid enough to join the set could still be a profitable exercise for these operators. It also ensures that there's always a set of online nodes monitoring the state chain and ready to participate.

Backup Validators

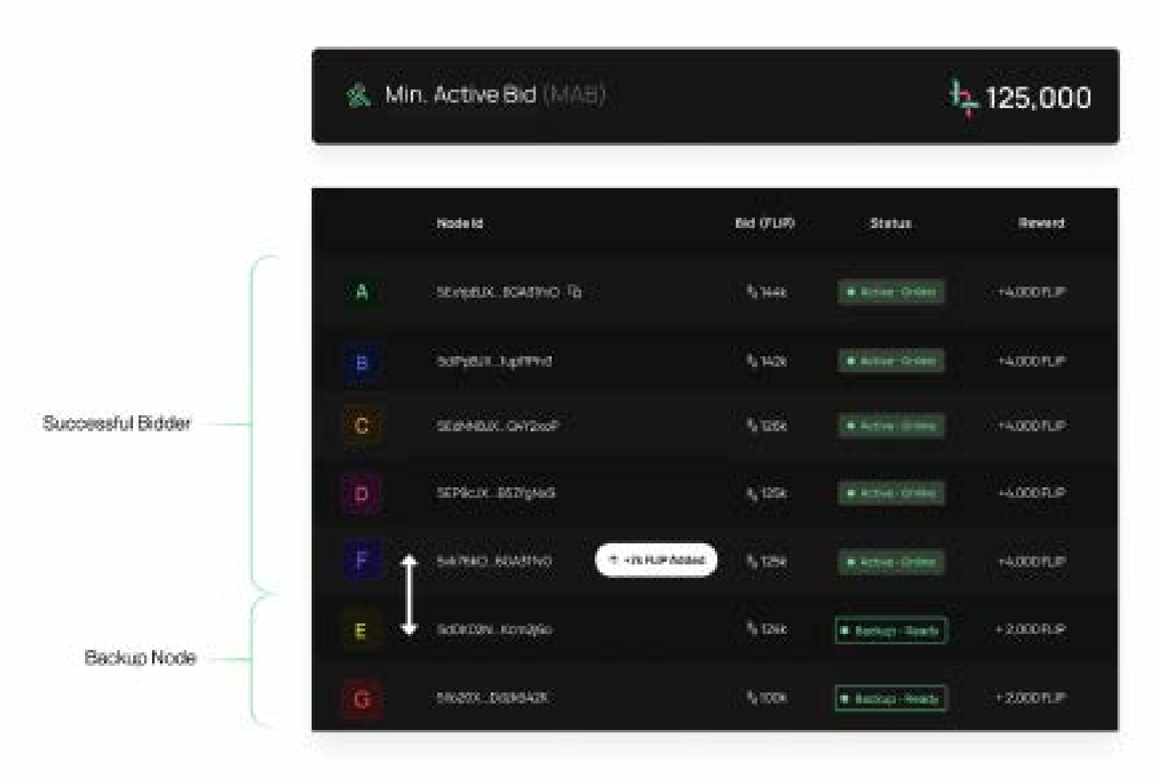
Instead of paying an equal reward to the Backup Validators, a fixed reward is distributed proportionally to Backup Validators based on their stake size. This is because we want to incentivise these Backup Validators to have the highest amount they can in case of an Emergency Rotation, in which the highest bidding backup nodes would be included first, and also to incentivise the nodes to hold onto their stakes and await the next auction.

For Backup Validators, we also allow bids to be placed outside of the normal auction cycle and immediately reflect increased bids for Backup Validators in the rewards they are paid. This provides a direct and immediate incentive to stake as much as possible as soon as possible, both increasing total bidding and increasing the likelihood that these more active and collateralised Backup Validators will be included in the next set.

The rules for Backup Validators are as follows:

- A fixed reward of LLL (much less than the Authority Set reward) is allocated to the Backup Validators for a given Epoch.
- There is a limit on the number of Backup Validator slots - 1/3rd of the current Authority Set size. Any bidding node outside this limit will not be a Backup Validator.
- So long as Backup Validator remains Qualified (including being Online), rewards will be paid to it based on their stake, proportional to their share of staked LLL in the total number of LLL staked in Backup Validators (It should be noted that individual Backup Validators will never earn more than Authority Validators).
- Any Backup Validator that goes offline will no longer be Qualified, and therefore gives up a Backup slot to another online Bidding node. However, they can come back online at any time, and provided they are still bidding enough to win back their Backup Validator slot, can immediately begin earning rewards again after the first successful heartbeat interval.
- In an Emergency Rotation, only the top third of Backup Validators will be included in the Emergency Set. This is to prevent mass deregistration events allowing large numbers of low-collateral nodes to form a superminority in the vast majority of cases.

Going back to our example, with the Backup Validator system in place, our third round might look something like this:



Min. Active Bid (MAB) 125,000

	Node Id	Bid (FLIP)	Status	Reward
A	5EvbBUX...8CAEYhO	144k	● Active · Online	+4,000 FLIP
B	5dPyBUX...1upfPhd	142k	● Active · Online	+4,000 FLIP
C	5EdhNBUX...QAY2xoP	126k	● Active · Online	+4,000 FLIP
D	5EP8cJX...B6ZfgAsG	125k	● Active · Online	+4,000 FLIP
F	5ek7hkO...6GA51VO	125k	● Active · Online	+4,000 FLIP
E	5dDKDJN...Kcm2j6o	124k	● Backup · Ready	+2,000 FLIP
G	5Nb20X...D6bM42K	100k	● Backup · Ready	+2,000 FLIP

As you can see, Validator E is earning enough rewards to stay close to the Minimum Active Bid. If one of the Authority Validators were to unstake or be deregistered, the Minimum Active Bid would drop to 124k $LLL, if no new bids are placed at all.
The design addition of rewards for Backup Validators solves the remaining problems with Launchledger's SSOD Auctions, better addressing the key goals of the design and ensuring a more stable and redundant network composition, whilst encouraging competition even among participants bidding below the Minimum Active Bid threshold.

In Summary

Launchledger's SSOD Auction system should produce a range of positive behaviour from Validators, with the effect of furthering the defined goals of the network. It provides a simple and effective framework for Validators to maintain long term positions within the network, whilst also naturally encouraging the reinvestment of rewards back into the Validator slots. It is fair and predictable to new entrants who, after the first few auction cycles, should be able to accurately estimate their performance in auction rounds before they start.

Coupled with additional rewards, the SSOD auction creates incentives for Backup Validators to remain active. This will help keep the minimum bid higher when other Validators drop out (or in the case of an Emergency Rotation), making for more sustainable collateralisation of the network.

Higher collateral means higher liquidity security. In turn, this maximises the number of assets that Launchledger can support and allows for better pricing for users. Better prices should mean more volume and more volume means more destroyed LLL. Destroying LLL should serve to increase the overall collateralisation of the network. You can hopefully see that this has the potential to form a strong positive feedback loop under the right conditions.

Governance & Security

Governance Processes

The Launchledger protocol will not be set in stone. A governance process must exist in order to upgrade the network to support new blockchains, new pools, update fees, and adjust rewards as needed. Further, governance processes can be implemented which could significantly reduce the potential impact of unexpected events such as supermajority attacks, ransom attacks, and software exploits.

However, in order to preserve the decentralised nature of the protocol, the following principles must be observed:

1. Governance processes in a decentralised protocol should not be able to be unilaterally executed by any one party.
2. An honest Validator network should be able to expel any nominated governance body.
3. Most importantly, effective control over user funds should never be given to any single party. Effective control means the ability to unilaterally control the flow of funds, or to permanently block access to funds.

In this section, we present a range of governance features of the Launchledger protocol designed to upgrade and protect it.

The Governance Keys

The Governance Keys are arbitrary keysets that are nominated implicitly by the Validator network. The keys themselves can be any multisignature scheme that is compatible with the governance pallet that Launchledger Validators run. What this means is that the Governance Keys could be managed by any Threshold Signature Scheme, standard n of m scheme, or even a single key (though this would likely never be voted for). Who is behind said keys is a social matter - DAOs, councils, companies or even single individuals could be elected as a governance keyholder with enough backing, though we will refer to them generally as "Councils." A supermajority of Validators can at any time propose to elect a new Council.

There are two Governance Keys in the Launchledger protocol, each held by different Councils and used for different purposes.

1. **The Governance Key:** This key holds many responsibilities, and is intended to be held by the primary developers of the protocol, or a body that represents them. Outside of managing the upgrades process, the 'Governance Council' can activate essential governance processes which aim to protect the protocol from attacks, and software exploits. However, these security features of the Launchledger protocol often require an additional signature from the Community Council to enable the full governance capabilities built into the protocol.
2. **The Community Key:** This key acts as a check-and-balance for the Governance Key. Intended to be held by a body of community members separate to the primary developers of the protocol, this key has no special abilities by itself, but it is required to allow the Governance Key to use the more powerful security and governance features of the protocol. If used correctly, this key prevents the Governance Keyholders from being a threat to the protocol, or exerting unilateral and effective control over the protocol through high-stakes governance actions.

The reason the role of check-and-balance isn't simply left to the Validators by default is because there may be scenarios where they aren't the best arbiters of community interests. This is especially true in the unlikely cases where some or all Validator keys have been compromised by hostile attackers. Further to this, Validators make up the heart of the protocol, but may not share the same views as users, liquidity providers, and developers that are also essential to any Web3 protocol's long term success. By forcing Validators to nominate a distinct and smaller governing body, the network can enact governance votes faster and with fewer conflicts of interest.

The keys are automatically deployed to Smart Contract Vaults and the State Chain Gateway contract by the Validator network, though a cooldown period is enforced to mitigate against attacks involving software exploits.

Voting in New Keys

Once the protocol is up and running, the Governance Council and the Community Council can be rotated only through a token-weighted vote on the state chain. Anyone with a token balance on the State chain can vote in proposals, even those currently staked into Validators. The minimum threshold for a successful proposal is a 2/3rd+1 super majority.

Anyone can create a new governance process to vote on the replacement of either one of the keys on the State Chain at any time. Voters must opt in to any key rotation proposal in order for a vote to be counted. The proposals last for 7 days, and the token balances are only counted at the end of the proposal timeframe. Anyone that is actively unstaking or undergoing claims at the end of the vote will not have their tokens counted towards the result.

In order to mitigate against a scenario where a hostile supermajority has overridden this process in an attempt to use the security features against the protocol and its users, a cooldown period is required. By enforcing State Chain rules which prevent the rotation of both governance keys for a 14 day period, a hostile supermajority will have to wait a week before the new key is installed and usable.

Liquidity providers should therefore receive as much as 2 weeks warning that the protocol is being taken over by a new party that may or may not be connected to the existing users and developers of the protocol. From there, they can decide if they wish to keep their assets in the system, or withdraw before any actions can be conducted by the new keyholders. Similarly, validators may wish to unstake during this cooldown window if they decide they do not support the holders of the new governance keys.

Thus, any party can become a Governance keyholder, but it requires not only support from a supermajority of Validators and other token holders but also backing from the liquidity providers in place.

Runtime Upgrades & Hard Forks

The Runtime Upgrade process, which is enabled by the Substrate blockchain framework, changes the default nature of protocol upgrades from the traditional forking system, one which relies on manual binary upgrades, into one whereby the Governance Council has the ability to upload changes to the blockchain code directly to each individual node, and at a specified b lockheight, h ave t he r esponding n odes s witch o ver t o t he new protocol rules seamlessly. This is a great feature to enable fast and lower-friction development, but doesn't work perfectly in Launchledger

Due to the Launchledger Engine, a sidecar binary that runs outside of the State Chain runtime, Launchledger Validators won't always be able to be seamlessly upgraded in this way. If any changes in a runtime upgrade also require a change in the Launchledger Engine, Validators that have not updated this secondary binary may be suspended by the other Validators when they can't meet the newly required functionality specified in the runtime upgrade.

Traditional forks are also still possible using the Grandpa finality module that ships with Substrate, although we suspect that the governance processes designed for the protocol will be a more appropriate method for both upgrades and for making major project direction changes amongst competing groups of participants by voting on new governance keys.

Security Considerations & Features

Decentralisation presents a challenging array of security considerations that centralised exchanges simply don't have to deal with. Previous attempts at secure cross-chain systems have proven to be challenging in the wider ecosystem, with losses from security incidents in the cross-chain sector easily reaching into the hundreds of millions of dollars. We have thought long and hard about Launchledger's protocol security, and take it very seriously. Here, we'll explore some of the security properties of the Launchledger protocol, the range of security risks we have identified, the assumptions that we've made, and the features designed to mitigate those risks.

Potential Protocol Risks

No security profile can be built without a firm understanding of the risks to which the protocol is exposed. The surface area for faults and exploits with bad outcomes for protocol participants is not infinite, but it is larger than most blockchain projects. To begin, let's enumerate where things could go wrong:

1. Liquidity could be lost or stolen through the AMM/State Chain logic - LPs and swappers could experience potentially serious or even complete losses if the AMM logic or other related accounting systems within the blockchain fail or are exploited. By undermining the intended logic of the Accounting Layer, Validators could distribute funds incorrectly, leading to incorrect payouts. Attackers, if such exploits are discovered, could use them to drain funds into their own wallets. This type of security issue has happened in THORChain, for example.
2. Liquidity could be lost or stolen through the vault management system - LPs and swappers who have deposited funds and are waiting to be paid out could experience potentially serious or even complete losses if the Launchledger engine is compromised or exploited in this way. Close attention needs to be paid to TSS ceremonies, broadcasting logic, and key rotation processes in the vault management system.
3. Validator private keys could be compromised, seriously undermining the security of the system as a whole if enough keys are compromised. Superminority and supermajority attacks could occur if this happens at a large enough scale. Individual Validators could also have all of their $LLL stolen if their Validator key is compromised.
4. Validator private keys could be simply lost, which if it occurs could result in the liquidity held by Validators being unable to be moved, potentially forever if enough keys can not be recovered.
5. The $LLL token market could collapse if the code which handles emissions, rewards, and claims is compromised or exploited.
6. External chains could be attacked or experience major reorgs which would likely break the Launchledger Accounting Layer if incoming transactions that have been witnessed are later rolled back on the external chain.

Protocol Level Security Features

Many of the above risk areas can largely be addressed by enforcing good engineering processes, a comprehensive integration and unit testing suite, comprehensive internal and external audits, and offering significant bounties to penetration testers. However, any complex software system that assumes that everything will work as intended, even if all of the correct precautions have been taken, has failed to develop a comprehensive security plan.

Redundancy is a key security principle and especially when applied in complex distributed systems with many moving parts like the Launchledger protocol. Our unique security features are designed to offer redundancy during many potential security incidents.

State Chain Safe Mode

As all of the above risks will impact the State Chain, and in fact most of the risks could arise from State Chain code, a system to pause all critical chain activity is essential in the case of a security incident.

The Safe Mode is a chain state where Validators temporarily agree to halt most core functions of the protocol. Blocks will continue to be produced, so that key governance extrinsics and events can be processed, but other functionality is heavily reduced. In Safe Mode:

- Witness extrinsics from the block at which the Safe Mode was activated are not accepted. Any ingress events that are not finalised will have to wait until after Safe Mode is deactivated to be rescanned and processed.
- Auctions and vault rotations are no longer automatically triggered.
- No Egress transactions are processed (No swap payouts, no LP withdrawals, and no new staking claims will be processed).
- Supply sync transactions are suspended.

Validators can choose to opt-in or opt-out of Safe Mode delegation. This is done during the registration process and can be changed by submitting an extrinsic to the chain at any time. Validators which opt-in to Safe Mode delegation defer their actions to that of the Governance Council, meaning if the Governance Council submits a “safe mode request” governance extrinsic to the chain, those validators automatically count as having voted to enter Safe Mode.

A threshold of only 50% of the Authority Set needs to vote in favor of Safe Mode to activate it. Turning it on gives Validators, developers, and the community time to analyse security incidents in a reduced risk state. During Safe Mode, runtime upgrades and other fixes c an b e d eployed t o t he n etwork, a llowing f or a measured recovery process.

Once the incident is resolved, the network can exit Safe Mode by a supermajority vote on a governance extrinsic that can be submitted by anyone. Once again, opted-in Validators delegate their vote to the governance key.

State Chain Gateway Protection

The State Chain Gateway is an Ethereum smart contract that allows holders of $LLL, an ERC20 token, to put up collateral on the Launchledger State Chain. This collateral is credited on the State Chain as an account balance which can be used to run Validators, Brokers, or perform active liquidity management activities. The State Chain Gateway contract also processes redemptions to send unlocked $ERC20 tokens back to account holders after receiving a valid redemptions certificate (a TSS 101 of 150 signature from the current Authority Set). The State Chain Gateway can call the $LLL Token contract to mint more tokens if required as well.

There are quite a few different ways this could be exploited. Namely, risks 1, 2, 3 and 5 are all relevant and could all cause losses if the State Chain Gateway contract is involved. To provide a layer of protection against risks 1, 2, 3, and 5, the State Chain Gateway comes with a feature which adds a 2-day delay before a user can actually claim tokens (executing a claim) from the contract after submitting a withdrawal certificate (registering a claim) to the smart contract. The delay itself changes little, but in combination with its second feature, it can play a major role in providing redundancy against attacks and losses related to bridging $LLL.

The second feature allows the Governance key to suspend the State Chain Gateway from executing redemptions, or register new ones. If an issue where fraudulent or incorrect registered redemptions are detected within the delay window, negative impacts from $LLL bridging-related incidents can be negated.

That being said, freezing claims indefinitely is not a solution. Any registered false redemptions would be able to be executed when the State Chain Gateway is unfrozen again if claims did not expire after some period of time. As the State Chain Gateway enforces an expiry time, which is set at the time of registering the redemption, the executions can not be processed by the State Chain Gateway contract after 3 times the length of the claim delay. In other words, on mainnet all unexecuted redemptions expire after 144 hours.

This means that in a security incident where the State Chain Gateway is frozen, a graceful recovery is possible if underlying issues on the State Chain can be resolved and the State Chain Gateway is unfrozen after all registered claims have expired, which should be within 144 hours.

During this time, the State Chain and Vaults could continue operating, unless they too have been affected by the incident, in which case Safe Mode is likely to also be active.

NOTE: The current iteration of the State Chain Gateway contract features a governance withdrawal function in tandem with the community key. It is unclear if this is necessary or desirable going forward, and its removal is slated for discussion.

Contract Vault Protection

Risks 1, 2, 3, 4, and 6 apply to the Vaults, and the losses that could be experienced due to these risks would impact liquidity providers, who could lose all of their funds, as well as swappers who are waiting for a payout.

Smart Contract Vaults also come with similar capabilities as the State Chain Gateway, however unlike the Gateway, there is no delay on payouts from the Vaults. This is necessary to ensure fast swapping times for users, but it does make the protections weaker than in the State Chain Gateway contract.

All smart contract based Vaults can be frozen by the Governance Council. Upon witnessing this function being called, the Validators will no longer be able to process egress transactions from that Contract Vault. It is assumed that Safe Mode may already be active at this time in most scenarios, so this could already be the case, but does add an additional layer of protection for the large pools of liquid funds held in these Vaults.

While frozen, the Community Council plays a very significant role. If it is assessed that the contract or the network could be irreversibly compromised, or that the aggregate key has been lost or corrupted for good, a final option exists to ensure that funds remaining in the Contract Vaults can be recovered and returned to users and liquidity providers.

Within the Contract Vaults, the Community Council can call a function that authorises the Governance Council to withdraw all funds in the vault. This check and balance ensures that the Governance key never has unilateral control of the funds and must first both freeze the vault contract and then achieve approval from the community key before this extreme recovery method can be used.

This protection feature is considered very important considering that it is expected that over 80% of all assets held on the AMM are likely to be held in Contract vaults, as opposed to Native Wallet vaults.

Security Ratio

Although assessed to be incredibly unlikely, even without this security feature, to prevent financially motivated attackers from benefiting from a collusion attempt the Security Ratio is enforced by the Validator network. As the chainstate is updated with ingress, egress, and trade information, it is possible to deterministically calculate the value of assets controlled by the validator network as well as the value of $LLL staked in validators.

Because only two-thirds of the validators are required to conduct a supermajority attack, the total value stored by the validators should not exceed the locked collateral of two-thirds of the validator network. This is the security ratio, where:

$Security\ Ratio = Liquidity\ TVL : Collateral * \frac{2}{3}$

For a variety of reasons, it is assessed that exceeding this ratio presents no immediate or significant danger, unless the ratio climbs well beyond 1:1. For example, a ratio of 2:1 would imply that the value of the stored TVL would yield a theoretical maximum attack profit of 100% return if a perfect supermajority attack was conducted. However, it is not clear that a one off 100% yield would be enough to justify executing a supermajority attack, given the opportunity cost of future validator returns, collateral appreciation, the risk of failure, complete loss, and real world criminal prosecution. The practical risk should be continuously assessed and the security ratio reviewed if it becomes a limiting factor in scaling the protocol.

In any case, the Security Ratio of 1:1 will be enforced by the validator network to prevent buildups of collateral not being fully utilised.

When the security ratio approaches or exceeds 1:1, the Validators will be required to automatically purge liquidity held by liquidity providers. Given that refund addresses have already been provided, it is possible to initiate an automatic egress return transaction for any liquidity provider at any time.

When the conditions for purging have been met, Validators will deterministically select liquidity used the least over the last 30 days. Any liquidity not yet deployed in pools would be the first to be purged. Secondly, liquidity held in unconcentrated ranges would be purged next, and so on, until the Security Ratio is reduced to below 1:1.

Deposits to liquidity positions will also be blocked when the ratio reaches 1:1. To prevent disruption to active liquidity strategies in the JIT AMM, only the least active liquidity providers and new liquidity providers will be blocked from making deposits until the ratio drops below 1:1 again. If the ratio exceeds 1.3:1, then all deposits will be blocked.

Given it is possible that large swaps by users could push the ratio above any of these thresholds temporarily, the security ratio is calculated using rolling averages at regular time intervals rather than instantaneous calculations. This also helps reduce the frequency of on-chain calculations required.

Supermajority Attacks

A hypothetical existential threat exists in the Launchledger protocol (and for that matter, all decentralised protocols): the Supermajority Attack. Launchledger requires a signature from 100 of the 150 Validators to produce valid transactions to move funds on external chains. If a malicious supermajority were to exist, they could easily steal the funds in all vaults. Likewise, a supermajority would be required to alter State Chain history, confirm false witness transactions, or other potential actions which could result in a loss of protocol funds.

The Launchledger protocol guarantees that as long as an honest superminority (a blocking vote of 50 nodes) exists in the Authority Set, and the protocol has not malfunctioned, all witnessing, State Chain execution, and egress transactions on other chains will be executed deterministically, with no scope for falsification.

If a supermajority of the Authority Set colludes or has their keys compromised, then it should be assumed that all funds could be stolen, and complete disruption, takeover, or failure of the network is possible. Launchledger provides no security guarantees in a supermajority attack situation, and in fact nor does any other credibly decentralised blockchain network of any kind. Correctly designed economic security systems are intended to prevent supermajority attacks, but the possibility of one can never be totally ruled out, and so is treated as an accepted risk, though we strongly believe it will never occur.

The risk of collusion is mitigated mostly through the enforcement of the Security Ratio, though without guaranteeing complete protection. It will always be possible in decentralised networks that if enough irrational actors exist, collusion can occur. However, we assume that there will always be a rational superminority in the Authority Set. Extensive reasoning on this subject is possible, though we don't think it's worth discussing here.

Another vector for a supermajority attack is a wide-scale validator key compromise event. However, in modern server architecture, it shouldn't be possible to compromise over 100 individual servers. Validator operators are also incentivised to secure their keys, as exposing them could result in all of their own staked $LLL being stolen. However, maintaining secure Validator nodes is essential, and software exploits (not just in Launchledger software, but also the operating system and other packages running on the machine) that allow for remote code execution or root access could potentially expose the keys of multiple Validators if they share the same exploitable architecture, which could potentially lead to a supermajority attack.

That being said, aside from setting sensible defaults for the software that is shipped and the support that is given, Validator server security isn't really protocol design and therefore must be considered another accepted risk. If a common exploit that could expose private keys existed across 100 servers running Launchledger Validators, such as a zero-day exploit in a common linux distribution, it's likely that the exploit could be used to target much bigger and easier targets, and would likely result in mass security breaches across dozens of high-value software protocols globally. Aside from encouraging Validator operators to keep up to date with security patches, there's not a lot Launchledger developers can do about that, and so it's an accepted risk that at least a superminority of Validators will remain both honest and secure.

Superminority Ransom Attack

An attacker that controls a superminority (>1/3rd) of the Authority Set can block all actions in the protocol, effectively freezing all funds, which could be held for ransom. This way, the attacker can extract value from the users without ever having to acquire the full 2/3rds of validators required to pull off a complete theft of the pool.

In practice, we consider this incredibly unlikely, especially considering that the Vault contract protections would allow the governance and community keys to extract the bulk of value out of the system (everything held on EVM chains, which includes all USD), limiting the upside potential of the attack. The downside is very likely the complete loss of LLL collateral used to acquire the superminority, which if the Security Ratio is enforced, will be worth more than the remaining funds being held for ransom by the attacker, making the attack unprofitable.

Incentive Design: Emission & Burning

Long-term token economics of the Launchledger protocol

Launchledger has an **elastic supply**. There is minting and burning which means that the protocol does not have a fixed or final token supply. The network genesis starts with 90m LLL tokens but will change over time based on demand. This is very similar to how Ethereum's token economics work after the EIP-1559 upgrade, which has been very successful in limiting the supply of ETH and rewarding validators and holders.

Launchledger aims to reward holders and Validators using a similar methodology, however in Launchledger's case, the token-burning mechanism is derived largely from swap fees rather than gas fees, which comes with some additional positive properties.

Emissions

Firstly, let's explore protocol emissions. Emissions are newly minted tokens given to various participants to incentivise the behaviour needed to secure the network and operate the protocol.

Purpose of Emissions

The rewards distributed to Validators are to ensure the economic security of the protocol. Launchledger requires Validators to stake tokens that can be slashed in order to ensure the safety of liquidity contained in the protocol.

This economic security is discussed in more detail on the Governance & Security page.

In the future, additional budgets may be altered or granted through governance to fund liquidity provision incentives or trading rebate programs to extend overall market performance.

Emission Rates

Launchledger has globally limited emissions caps based on an annual target rate. In reality, the emission rates are calculated every block, the per-block emissions multiplier is calculated such that compounding it over the course of a year hits the annual target rate. The actual rate of emissions fluctuates constantly due to burning, slashing, and so on. As a result, the projected supply is not predictable in the long term without making assumptions.

The emission targets are distributed among the following recipients:

Recipient	Target Rate	Distribution	Emissions in 1st Year
Authority Validators	7% annual	Equal Shares	6,300,000/90,000,000 LLL
Backup Validators	1% annual	Pro-Rata	900,000/90,000,000 LLL

Estimated Yields and Lockup Ratio for Validators

Due to the cap of 150 Authority Validators imposed upon the protocol due to FROST Signature Scheme scalability limitations, the Validator Auctions, Bonds & Rewards will determine the amount required to bid through market dynamics every few days.

Because the number of tokens being given to Validators as rewards is not dependent on the amount staked, the actual yield validators receive will be determined by competition in the auction market. We can predict the total staking equilibrium based on the performance of other protocols and their staking systems.

Predicted Token Yield and Minimum Bids for Authorities

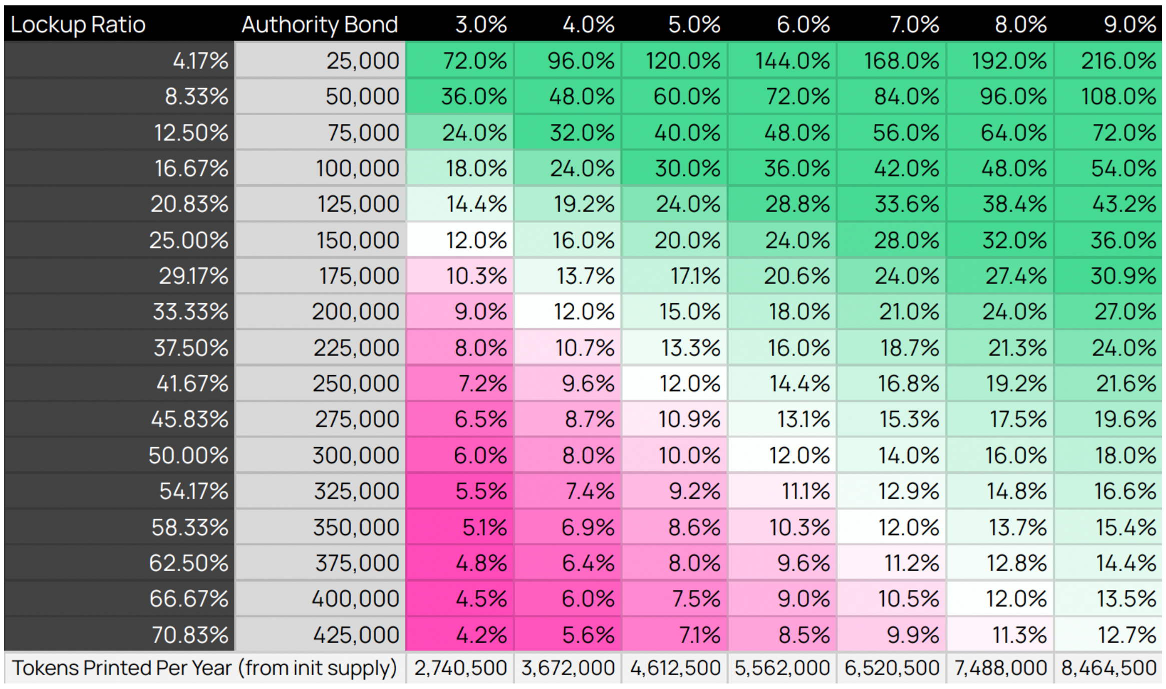
Many other proof-of-stake or similar systems show us the typical yield that Validator operators can expect and can be used to estimate how much the rewards will end up being based on how many tokens are staked globally in the auction system.

By analysing various staking rates and yields elsewhere in the ecosystem, we can see that reward rates on popular protocols range from 6% to 22% annually. As these protocols have matured, we can assume that for the most part, these are stable staking markets with changes taking place slowly over time.

In general, we can also say that the proof-of-stake systems with lower returns are typically very easy to operate or have lower-risk token models, whereas the ones with higher return rates are more difficult and/or expensive to operate or have riskier token models.

As Launchledger will have a higher adjusted rate of return considering the burning mechanisms, we would expect that the risk premium on Launchledger would be fairly low, but the cost and difficulty of operating nodes will be higher than typical protocols due to the requirement to observe and connect to many external chains.

Therefore, we would expect the auction market to stabilise when Validators are earning a reward of between 10% and 18% overall. In this matrix, you can compare the yields expected at different Authority bond levels and the derived staking ratio.



Lockup Ratio	Authority Bond	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%	9.0%
4.17%	25,000	72.0%	96.0%	120.0%	144.0%	168.0%	192.0%	216.0%
8.33%	50,000	36.0%	48.0%	60.0%	72.0%	84.0%	96.0%	108.0%
12.50%	75,000	24.0%	32.0%	40.0%	48.0%	56.0%	64.0%	72.0%
16.67%	100,000	18.0%	24.0%	30.0%	36.0%	42.0%	48.0%	54.0%
20.83%	125,000	14.4%	19.2%	24.0%	28.8%	33.6%	38.4%	43.2%
25.00%	150,000	12.0%	16.0%	20.0%	24.0%	28.0%	32.0%	36.0%
29.17%	175,000	10.3%	13.7%	17.1%	20.6%	24.0%	27.4%	30.9%
33.33%	200,000	9.0%	12.0%	15.0%	18.0%	21.0%	24.0%	27.0%
37.50%	225,000	8.0%	10.7%	13.3%	16.0%	18.7%	21.3%	24.0%
41.67%	250,000	7.2%	9.6%	12.0%	14.4%	16.8%	19.2%	21.6%
45.83%	275,000	6.5%	8.7%	10.9%	13.1%	15.3%	17.5%	19.6%
50.00%	300,000	6.0%	8.0%	10.0%	12.0%	14.0%	16.0%	18.0%
54.17%	325,000	5.5%	7.4%	9.2%	11.1%	12.9%	14.8%	16.6%
58.33%	350,000	5.1%	6.9%	8.6%	10.3%	12.0%	13.7%	15.4%
62.50%	375,000	4.8%	6.4%	8.0%	9.6%	11.2%	12.8%	14.4%
66.67%	400,000	4.5%	6.0%	7.5%	9.0%	10.5%	12.0%	13.5%
70.83%	425,000	4.2%	5.6%	7.1%	8.5%	9.9%	11.3%	12.7%
Tokens Printed Per Year (from init supply)		2,740,500	3,672,000	4,612,500	5,562,000	6,520,500	7,488,000	8,464,500

Given the budget of 7%, we would expect the staking ratio to end up between 37% and 66% of the total supply, aided by the existence of liquid staking products built for Launchledger We would also expect validators to eventually require between 200,000 LLL and 400,000 LLL tokens to make it into the authority set, though we expect this to be lower in the opening months of the auctions and gradually increase with competition for strong starting yields.

Backup Validator Rewards

Backup Validators are normally the top 50 bidding non-Authroities (see Auction Theory (SSOD)) and receive rewards from a limited budget of 1% per year target rate. The budget for rewards is distributed on a pro-rated basis, meaning those backup validators with a higher relative stake to the other Backup Validators will receive more rewards.

The system immediately reflects i ncreased b ids f or B ackup V alidators i n t he r ewards t hey a re p a id. This provides a direct and immediate incentive to stake as much as possible as soon as possible, both increasing total bidding and increasing the likelihood that these more active and collateralised Backup Validators will be included in the next Authority set.

In more precise terms, a given Backup Validator's rewards will be equal to 80% of the minimum between the Authority reward OR the Authority reward divided by the Minimum Active Bid squared, multiplied by the stake of the Backup Validator squared. This minimum check is performed to ensure that Backup Validators never earn more than 80% of the rewards of an actual Authority, even if they happen to be staking more than the minimum required bid.

If the total sum of all Backup Validator rewards exceeds an emission cap, then each of the Backup Validator's rewards will be multiplied by the 'capping factor' which is just the emissions cap divided by the total targeted rewards. Multiplying the rewards by the capping factor, ensures Backup Validators as a whole are collectively never paid more than what is budgeted for in the emissions schedule.

Yields for Backup Validators depend both on the stake of each validator as well as the number of tokens staked across the whole backup set and are therefore volatile and unpredictable, but overall capped at the target rate of 1% annual inflation.

Burn Mechanisms

At Launchledger we settled on an emission style that can broadly be likened to Ethereum's EIP-1559 token model.

Set emission rates are defined for Validator rewards, liquidity incentives, and other programmatic initiatives. These all create newly minted tokens which are allocated to the various operators of these systems.

On the LLL side, we have introduced mechanisms that also remove tokens from circulation forever. Burning is an underappreciated topic. While the narrative around 'deflationary' assets is strong, the reality is very few protocols even have the technical ability to be truly deflationary. Bitcoin, for example, can not be described as deflationary. At best, it can be described as supply neutral in its final form, with the only mechanism for circulating supply to go down is people losing their keys (a non-trivial proportion actually!).

Ethereum has been far from deflationary, having grown from 70m ETH in 2015 to 115m ETH today, but in relative terms, supply hasn't increased much more over time compared to Bitcoin, which has also minted heaps to miners over the years. With the new token model in place, Ethereum is now steadily decreasing its supply through burning: https://ultrasound.money/

Launchledger shares a similar model, with a few forms of burning mechanisms in play to encourage this burning and reflect captured value in the LLL market.

The Network Fee

The most important mechanism in play is the Network Fee. For every swap that passes through the Launchledger system, a small fee is taken from the user (in USDC) and is used to buy LLL tokens in the built-in
USD/LLL pool. This purchased LLL is automatically burned, removing it from the supply.
This is done by taking a 0.1% cut of a user's USD swap amount. This could be at the start, middle, or end of swap processing depending on the route. This always occurs at some stage in the swapping process as a function of having USD Denominated Pools.
For example:

- BTC > USD > ETH
- SOL > USD
- DOT > USD > ETH

All involve USD in the swap process. The ability of the network to automatically execute a buy-and-burn strategy with the proceeds of these fees is another unique advantage to the appchain architecture the network operates.

Impact of the Network Fee

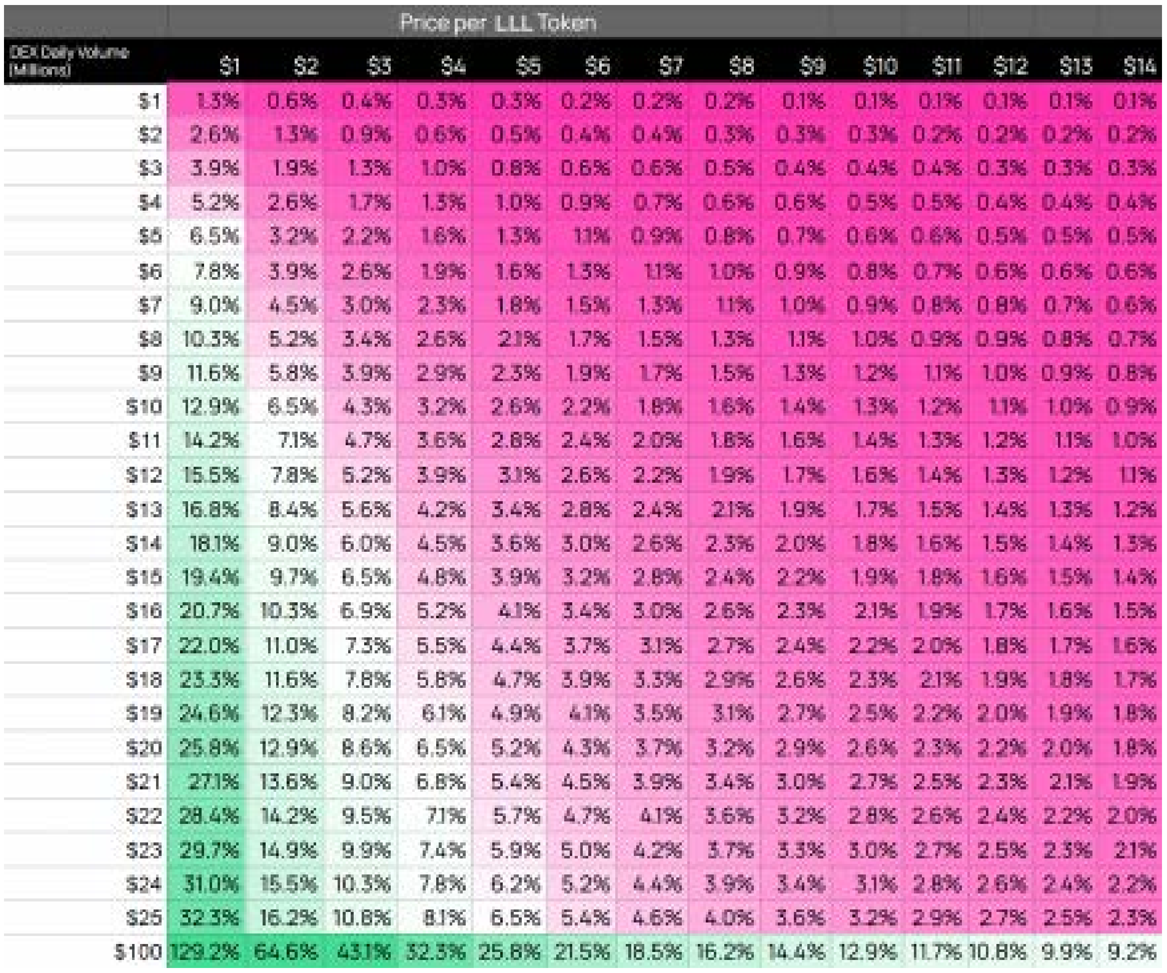
Exchanges are incredibly lucrative businesses. An exchange charging fees of 0.1% on trades with $500m in daily volume (a fraction of Uniswap's total average daily volume over the last year) should yield $182,500,000 in revenue every year. If all of those fees were directed into the purchasing of a token directly from the liquidity pool and then burning those tokens, the token holders would automatically benefit from this without having to lift a finger — it's simply automatically reflected in the token price.

If said exchange bought \$182m worth of tokens over a year, and if the average price was \$10 each (based on comparable DeFi tokens), that's 18.2 million tokens. Putting that in LLL terms, that would be over a fifth of the initial supply, meaning emissions would have to exceed 20% per annum to even come close to the number of tokens being removed from circulation. That means that if this did occur, the LLL supply would decrease over time based on current models, where we anticipate 8% inflation per year.

If we exclude all other types of burning and market mechanisms, We can actually model the projected impact of the network fee's design based on a wide range of average volumes, token prices, and net emission rates. In this model, the Y axis shows a range of daily volumes achieved across all Launchledger AMM pairs with a 0.1%fee, and the X axis represents the price of LLL. The resulting matrix shows the percentage of the LLL genesis supply that would be bought and burned annually at each of these levels.

A high percentage value would imply the LLL price is too low and the fees from the volume alone would cause the price of LLL to sharply rise to a stable level. Low percentages suggest that the buying and burning of LLL alone would not sustain this price. In this model, we estimate that the price would stabilise before the buy-and-burn offsets any natural churn from emissions being sold on the market. Given we expect most validators to retain their rewards to stay in the set, 5-6% is a reasonable estimate for realised inflation year to year.

Thus, we display high percentages in green where we would expect the price to rise naturally, and low percentages in pink where we would not expect the price to rise based on network fees alone.



	Price per LLL Token													
DEX Daily Volume (Millions)	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$10	\$11	\$12	\$13	\$14
\$1	1.3%	0.6%	0.4%	0.3%	0.3%	0.2%	0.2%	0.2%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%
\$2	2.6%	1.3%	0.9%	0.6%	0.5%	0.4%	0.4%	0.3%	0.3%	0.3%	0.2%	0.2%	0.2%	0.2%
\$3	3.9%	1.9%	1.3%	1.0%	0.8%	0.6%	0.6%	0.5%	0.4%	0.4%	0.4%	0.3%	0.3%	0.3%
\$4	5.2%	2.6%	1.7%	1.3%	1.0%	0.9%	0.7%	0.6%	0.6%	0.5%	0.5%	0.4%	0.4%	0.4%
\$5	6.5%	3.2%	2.2%	1.6%	1.3%	1.1%	0.9%	0.8%	0.7%	0.6%	0.6%	0.5%	0.5%	0.5%
\$6	7.8%	3.9%	2.6%	1.9%	1.6%	1.3%	1.1%	1.0%	0.9%	0.8%	0.7%	0.6%	0.6%	0.6%
\$7	9.0%	4.5%	3.0%	2.3%	1.8%	1.5%	1.3%	1.1%	1.0%	0.9%	0.8%	0.8%	0.7%	0.6%
\$8	10.3%	5.2%	3.4%	2.6%	2.1%	1.7%	1.5%	1.3%	1.1%	1.0%	0.9%	0.9%	0.8%	0.7%
\$9	11.6%	5.8%	3.9%	2.9%	2.3%	1.9%	1.7%	1.5%	1.3%	1.2%	1.1%	1.0%	0.9%	0.8%
\$10	12.9%	6.5%	4.3%	3.2%	2.6%	2.2%	1.8%	1.6%	1.4%	1.3%	1.2%	1.1%	1.0%	0.9%
\$11	14.2%	7.1%	4.7%	3.6%	2.8%	2.4%	2.0%	1.8%	1.6%	1.4%	1.3%	1.2%	1.1%	1.0%
\$12	15.5%	7.8%	5.2%	3.9%	3.1%	2.6%	2.2%	1.9%	1.7%	1.6%	1.4%	1.3%	1.2%	1.1%
\$13	16.8%	8.4%	5.6%	4.2%	3.4%	2.8%	2.4%	2.1%	1.9%	1.7%	1.5%	1.4%	1.3%	1.2%
\$14	18.1%	9.0%	6.0%	4.5%	3.6%	3.0%	2.6%	2.3%	2.0%	1.8%	1.6%	1.5%	1.4%	1.3%
\$15	19.4%	9.7%	6.5%	4.8%	3.9%	3.2%	2.8%	2.4%	2.2%	1.9%	1.8%	1.6%	1.5%	1.4%
\$16	20.7%	10.3%	6.9%	5.2%	4.1%	3.4%	3.0%	2.6%	2.3%	2.1%	1.9%	1.7%	1.6%	1.5%
\$17	22.0%	11.0%	7.3%	5.5%	4.4%	3.7%	3.1%	2.7%	2.4%	2.2%	2.0%	1.8%	1.7%	1.6%
\$18	23.3%	11.6%	7.8%	5.8%	4.7%	3.9%	3.3%	2.9%	2.6%	2.3%	2.1%	1.9%	1.8%	1.7%
\$19	24.6%	12.3%	8.2%	6.1%	4.9%	4.1%	3.5%	3.1%	2.7%	2.5%	2.2%	2.0%	1.9%	1.8%
\$20	25.8%	12.9%	8.6%	6.5%	5.2%	4.3%	3.7%	3.2%	2.9%	2.6%	2.3%	2.2%	2.0%	1.8%
\$21	27.1%	13.6%	9.0%	6.8%	5.4%	4.5%	3.9%	3.4%	3.0%	2.7%	2.5%	2.3%	2.1%	1.9%
\$22	28.4%	14.2%	9.5%	7.1%	5.7%	4.7%	4.1%	3.6%	3.2%	2.8%	2.6%	2.4%	2.2%	2.0%
\$23	29.7%	14.9%	9.9%	7.4%	5.9%	5.0%	4.2%	3.7%	3.3%	3.0%	2.7%	2.5%	2.3%	2.1%
\$24	31.0%	15.5%	10.3%	7.8%	6.2%	5.2%	4.4%	3.9%	3.4%	3.1%	2.8%	2.6%	2.4%	2.2%
\$25	32.3%	16.2%	10.8%	8.1%	6.5%	5.4%	4.6%	4.0%	3.6%	3.2%	2.9%	2.7%	2.5%	2.3%
\$100	129.2%	64.6%	43.1%	32.3%	25.8%	21.5%	18.5%	16.2%	14.4%	12.9%	11.7%	10.8%	9.9%	9.2%

This chart is derived from the circulating supply at month 6 of the vesting schedule, visible at https://Launchledger.io/token

State Chain Gas Fees

A lesser type of burning mechanism is the State Chain gas fees, which are automatically burned. All extrinsic submissions incur these fees, including liquidity provider updates, deposit channel requests, validator extrinsics, and so on. We do not expect these burning transactions to account for significant volumes of LLL unless network activity is extremely high.

For more information on these fees check how they apply to Brokers and Liquidity Providers.

Slashing

Finally, slashing and penalties for Validators are also burned. Hopefully, these penalties end up being insignificant, as a fully operational Authority set shouldn't ever incur these penalties if the operators are behaving correctly and maintaining their nodes.

To read more about these penalties, you can check out Reputation & Slashing.

Bitcoin Vault Design

Bitcoin is the oldest and most widely traded asset in the cryptocurrency industry. To support native cross-chain swaps between Bitcoin and the rest of the ecosystem, we have designed a unique Vault that allows the Launchledger Validator network to maintain continuous service over a pool of protocol-controlled funds.

Features of the Bitcoin Vault Design

1. Access funds using Schnorr Signatures as generated by our FROST protocol.
2. Deterministically generate an arbitrary number of Deposit Channel Addresses (Ingress Addresses) that are all controlled by the same key.
3. Ability to send a single transaction that sweeps funds from multiple Deposit Channel Addresses into a central vault.
4. Ability to perform a "key rotation" from one Authority Set to the next, whilst maintaining access to older ingress addresses, even after a key rotation has occurred.
5. Minimise the number of necessary signatures for any transaction.

Background on Bitcoin

Bitcoin behaves differently from most other cryptocurrencies. In Bitcoin, there exists no strict notion of an "address" as in other protocols. An address simply tells the sender of a transaction how to prepare that transaction so that the recipient may access the funds. Each transaction produces one or more "outputs" (Also called UTXOs = "Unspent Transaction Outputs"). Each output specifies an amount of Bitcoin that it contains, as well as a "locking script" (Bitcoin calls these locking scripts "ScriptPubKey"). The locking script is a small program, written in Script, which defines under what conditions the coins contained in the output may be spent (the conditions of this script "lock" the funds in the output). An address in Bitcoin is a short way to represent such a locking script, i.e. sending funds to a given address will always result in an output with the same locking script.

In order to spend the funds in an output, one must send a transaction that references an output to be spent, as well as another script, called the "unlock script" (Bitcoin calls this the "ScriptSig"). If the unlock script satisfies the conditions set up by the locking script, the funds can be used. Typically, a Bitcoin address contains the Hash of a public key. The corresponding locking script takes as input a public key and a signature and demands that the hash of the provided public key matches the Hash in the address. It also demands that the provided signature signs the details of the output and matches the provided public key. The unlocked script simply provides these two inputs (public key and signature).

Accessing BTC Using FROST Protocol Schnorr Signatures

In November 2021, the Bitcoin protocol activated the "Taproot" feature, which consisted of several changes. One of these changes is BIP 340, which introduces support for Schnorr Signatures. By using Taproot addresses as our ingress addresses, we can unlock any output sent to one of our ingress addresses by providing a valid Schnorr Signature. Bitcoin puts some additional constraints on Schnorr signatures, specifically:

1. The nonce n must be chosen such that $r = n \times G$ has an even y-coordinate r_y. To "fix" a nonce, one can just substitute $n \to Qn$ where Q is the known group order of the secp256k1 curve. To fix the corresponding r, one can either recalculate it from the fixed nonce, or just LLL the sign of its y-coordinate.

   In the context of FROST multisig, the nonce n is never exposed to any party for security reasons, but r can be calculated. To implement this fix for FROST multisig, each participant calculates their share of the signature as in step 5 of the FROST signing protocol but using:

$$z_i = \begin{cases} \lambda_i \cdot s_i \cdot c + d_i + (e_i \cdot \rho_i) \bmod Q, & \text{if } r_y \text{ even,} \\ \lambda_i \cdot s_i \cdot c - d_i + (e_i \cdot \rho_i) \bmod Q, & \text{if } r_y \text{ odd} \end{cases} \quad (1)$$

2. Same thing for the private/public key pair: The y-coordinate of the public key must be even. This can be implemented using the same mechanism already used for providing "compatible" keys in our Ethereum implementation.

3. The signature is just the concatenation of the 32-byte padded, big-endian x-coordinate of r, and the 32-byte padded, big-endian representation of s.

Taproot addresses "look just like" normal bech32 addresses in bitcoin, so any wallet that correctly supports sending to bech32 (should be most wallets) will be able to send to our ingress addresses.

Deterministically Generating an Arbitrary Number of Deposit Channel Addresses All Controlled by the Same Key

Another change introduced by Taproot are "MAST" scripts ("Merkelized Abstract Syntax Tree"). Details are provided in BIP 341 and BIP 342. These changes allow replacing the locking script of an output with a hash value. Now there are two options: Either, the hash value is interpreted as a public key and the funds may be spent by providing a valid signature for this public key, or the hash value is the hash of an (arbitrarily complex) locking script which needs to be satisfied in order to spend the funds. The unlock script trying to spend the funds needs to select one of these options and then either provide the required signature, or the "real" locking script that produces the hash together with the requirements to unlock it. Furthermore, any code branches in the locking script (if-else, etc.) will result in a binary tree of code paths, and the code of each possible path is hashed separately, resulting in a "Merkle tree". Only the root hash of this Merkle tree is used as the hash value in the locking script. The unlock script only needs to provide the code for the path that will be traversed during the unlocking (this results in additional privacy, as well as reduced size of the unlock script). We can now define a locking script that essentially looks like this:

```
if false {
let constant = 1337; // Some salt
} else {
let publicKey = ...; // Our Schnorr pubkey
check_provided_signature_for(publicKey);
}
```

This script has two code paths, but the first one will never be executed, because the condition is always false. Inside it, we can specify a different salt for each ingress address. The other code path is always the same and uses our public key to check the provided signature. Because of the salt, the Merkle hash will be different for each salt, resulting in a different ingress address, even though the public key is always the same. In practice, we do not use the Merkle tree feature, because of some additional complications. Instead, we use scripts like this where the salt is chosen as 3 in this example and the hex blob is our public key:

```
OP_3 OP_DROP 59B2B46FB182A6D4B39FFB7A29D0B67851DDE2433683BE6D46623A7960D2799E
OP_CHECKSIG
```

In general, the script consists of pushing the salt value onto the stack, dropping it, pushing our public key onto the stack, and then running OP_CHECKSIG.

Sweeping Inputs from Different Despoit Channels Into the Central Vault

Because Bitcoin has a different notion of what an "address" is, this feature is provided trivially in Bitcoin. Any transaction can specify multiple UTXOs to be used as "inputs" to that transaction. As long as the locking scripts of all UTXOs are satisfied, the single transaction can now spend the combined funds of all the referenced UTXOs. We would prepare a single transaction that uses the UTXOs of all our different ingress addresses and spends them into our vault.

Bitcoin Vault Rotations

Because our Vault is simply another Bitcoin address, a key rotation would just consist of a single transaction of all the funds from the old vault to a new address, which is controlled by a different key.

However, there is no way to change the locking script of an existing UTXO, so funds that are sent to an old ingress address (such as from an open despoit channel created during the previous Epoch) must be unlocked using the corresponding (old) key. If a validator set changes and the mutual key is replaced, the old key will not be accessible going forward. To handle this, we can't just "abandon" any funds sent to older addresses. We could force validator members to keep their old key shares for a certain period of time, or we find a way to "hand over" the control of old ingress addresses to the new key. For various reasons, we chose the "handover" option.

As one validator set transitions to the next, most members will probably stick around, so the set of old/new members that need to perform the handover is going to be quite small in practice. Nevertheless, this procedure would also work when completely replacing a validator set entirely.

Launchledger Keyshare Handover Protocol

We developed a modified version of the FROST keygen ceremony, in which both new validators and (some fraction) of old validators take part. This is the keyshaing protocol that we have implemented in our Bitcoin vault and can be applied across many more chains.

Let V_1 denote the set of existing validators, and let V_2 denote the set of new validators. Note that V_1 and V_2 are not necessarily disjoint. To simplify notation lets assume $|V_1| = |V_2| = n$ (generalising the protocol for $|V_1||V_2|$ should be relatively straightforward). The goal of the protocol is for V_2 to obtain a new set of shares for the t/n key that V_1 collectively hold.

The implementation will be based on the existing keygen/re-sharing implementation with the following modifications:

- We will select t validators (denoted V_S) from V_1 that will participate in the ceremony together with V_2. In practice, we expect most (if not all) of V_S to also be in V_2, but we make no such assumption in the protocol. The ceremony will thus have $n' = |V_S V_2|$ participants, and we will use n' "votes" (rather than n) when determining the outcome of the ceremony: e.g. we will require at least 2/3n reports against a particular party before penalising them.
- An additional stage will be added (Stage 0) in which V_S will broadcast public key shares of every party in V_S corresponding to the original key. These are the keys that will be used in Stage 4 to check that V_S have set their secret correctly. Participants not in V_S will assume that the values broadcast by the majority from V_S to be correct (using the same principle as described in Broadcast Verification section of https://www.notion.so/LaunchledgerFROST-implementation-44643d46165c4893aeb998517430bbae).
- In Stage 1 only participants from V_S will set their secret to their existing shares. Other participants will set their secrets to 0. Note that this is already supported by the current key re-sharing implementation.
- Just like in Key-Resharing, in Stage 4 (fast-forwarding past broadcast verification and hash commitments stages) all participants will check coefficient commitments against public key shares (known by V_S or received during Stage 0 by the rest of participants), aborting the ceremony if commitments are incorrect, reporting the misbehaving parties.
- In Stage 5 secret shares will only be generated and sent to parties from V_2. Importantly V_S will use a sharing polynomial corresponding to a t/n key (rather than t/n).
- From here onward the ceremony will proceed exactly like a regular keygen ceremony with the exception that only complaints (sent in Stage 6) from V_2 will be taken into account. At the end of the ceremony we expect V_2 to collectively hold n shares (with threshold t) of the key originally held by V_1.

Minimising the Number of Necessary Signatures for Any Transaction

Bitcoin requires a separate signature (over different data) for every single UTXO spent by a transaction, even if all the UTXOs use the same locking script. This means that we need to do a FROST signing for each ingress payment, even if we ultimately only submit a single transaction (As explained above). Numerous suggestions exist on how this requirement could be changed so that a single signature is sufficient (see BIP 118, this article, BIP 131 etc.), but none of these have been implemented in Bitcoin yet. Currently, we would need to sign multiple pieces of data through FROST in order to sweep funds from our ingress addresses. FROST is only "slow" when generating new keys, though. Signing is pretty fast, and we estimate that we can handle 30+ BTC deposit sweeps every minute without impacting the wider protocol.

Some Additional Learning Resources (written for macs for now)

- Read the book “Mastering Bitcoin” for some nice details on how Bitcoin works. It’s available for free here.
- To play around with Bitcoin without needing a full node, you can run an instance of the regtest network, which is a local test network with your computer as the only node:

On Mac, download a suitable version of the bitcoin-core application from here. Open the file, install by dragging the Bitcoin icon into the application folder. Start a terminal and run:

/Applications/Bitcoin-Qt.app/Contents/MacOS/Bitcoin-Qt -regtest

If it doesn’t start, go to “System Settings” → “Privacy Security”, scroll down. There should be an entry asking about the Bitcoin software. Allow it to be started and retry the above command.

In the Bitcoin software, click on “Bitcoin Core” in the menu bar and select “Preferences”. Click on “Open Configuration File” → “Continue” and add this line to the config file:

regtest=1

Save and close the editor. Now the Bitcoin app will always start in regtest mode, even when started via Spotlight or similar.

Blocks are not mined automatically in regtest, so we need to do this by hand. Create a new wallet and a “receive” address. Copy the address into your clipboard, then press cmd + T to open a terminal. run generatetoaddress 107 YOUR_ADDRESS_GOES_HERE (replace with the address from your clipboard, obviously) in order to mine 107 blocks, with all rewards going to your provided address. This is necessary, because the mining rewards for the first 100 blocks are not spendable in Bitcoin (weird bug), and the wallet will only consider funds available if they have 6 blocks of confirmation. If you later want to send some transactions, you can rerun this command with a lower blockcount (even 1) to my new blocks and include your submitted transactions.

To try some taproot features, go here and get the source code for **btcdeb**, the BitcoinDebugger. Follow the instructions for building it. If you use an M1 Apple machine and the build fails, comment out the line that says libbitcoin_a_LIBADD = $(LIBSECP256K1) in Makefileåm (should be line 89), run make clean and retry. Now you can use the “tap” executable to follow this tutorial.

Future Work: “Hub and Spoke” Cross-chain Liquidity Network of Networks

This paper has covered the core protocol in detail, but it has also alluded to the idea that there are limits on the number of assets that can be supported natively by one cross-chain liquidity network, such as the main Launchledger network. There are three limiting factors that constrain the maximum number of blockchains that can be supported:

1. Complexity - As more blockchains are added, Validators need to add more light client connections to remain in sync and witness events on each of those blockchains. Maintaining dozens of client connections to maintain their Online status is expected to become increasingly challenging, particularly as smaller and less secure chains are added to the network. There are also more transaction types and more advanced logic required to handle these new chains, increasing complexity, which inevitably increases the security risk surface area.
2. Economic Security - Another limiting factor is the collateral put up by validators to secure the liquidity in the system. If the security ratio is reached, no further liquidity can be stored in this current design. We have devised ways to handle this and safely extend the liquidity capacity of the network, but those solutions add considerable complexity to the vault management system. As more chains are added, more liquidity is required to support each of the new chains, using up the limited amount of collateral that can be in the system at any given time.
3. Signing Scheme Scalability - As more chains are added, the frequency of expected signing ceremonies required to be performed by the Validators increases, thus also increasing the hardware and bandwidth requirements of Validator operators. This is not expected to be a problem in and of itself, but a system where a Launchledger Validator needs to be run on multiple machines instead of a single server instance would not be desirable.

To overcome these challenges, one potential solution is to create ‘subnetworks’ - secondary instances of the Launchledger protocol that are collateralised separately to the main network, support a different range of assets, but still share a common USD pairing. This way, the USD can be routed between the main Launchledger network and the subnetworks as a means of completing swaps between assets supported by the different networks, in the same way that swaps are all done through USD on the main network anyway.

There are two possible ways of achieving this:

1. Where the main and subnetworks treat each other as separate and use the native swapping functions built into the protocol to achieve the swaps. This would keep the networks totally separate in the case of a security incident, but it would mean that swaps would be more expensive to execute if the route crosses networks, and would require an on-chain USD transfer from network to network for each swap between the networks.
2. Where the main and subnetworks have a shared security model which allows for the virtual routing of USD between them, avoiding the need for on-chain transfers every time swaps occur, but also introducing more security risks by exposing both networks to each other’s assets.

It is not yet known what a sustainable number of assets on the main network is. Time will tell, but it will likely become self-evident to the protocol developers and community when a subnetwork is required. In any case, this future work presents the opportunity to extend the functionality of the Launchledger product to support as many blockchains and assets as the centralised exchanges are capable of.

Conclusion

Launchledger offers a protocol for native cross-chain swaps that brings the average user experience of decentralised on-chain trading much closer to that of centralised exchange offerings, while maximising the benefits of decentralisation for the benefit of the user.

Given its generalised nature, the protocol can be extended to support almost any blockchain network, and offer users, protocols, and product access to markets that would otherwise only be reachable through centralised offerings.

It offers extreme capital efficiency possibilities for Market Makers, greatly improving the average user experience, whilst also being less reliant on large amounts of liquidity in order to function.

We hope that these qualities will drive forward the Launchledger protocol as the best chance of replacing centralised exchanges as the primary method of spot trading within the Web3 industry.

References

[1] Sergio Demian Lerner (2012) P2PTradeX: P2P Trading between cryptocurrencies. Available at: https://bitcointalk.org/index.php?topic=91843.0.

[2] @hagaetc (2022) DEX metrics. Available at: https://dune.com/hagaetc/dex-metrics.

[3] @Trayvox (2022) Layer Zero Thread. Available at: https://twitter.com/trayvox/status/1508734174705987586

[4] Prabhjote Gill (2022) One of the largest cryptocurrency swapping platforms just lost 1.3 million as users failed to update approvals. Available at: https://www.businessinsider.in/investment/news/one-of-the-largest-cryptocurrency-swapping-platforms-just-lost-1-3-million-as-users-failed-to-update-approvals/articleshow/88992186.cms

[5] Rekt.news (2022) Wormhole REKT. Available at: https://rekt.news/wormhole-rekt/

[6] Rekt.news (2021) Polynetwork REKT. Available at: https://rekt.news/polynetwork-rekt/

[7] vitalik Buterin (2022) We are the EF's Research Team. Available at: https://old.reddit.com/r/ethereum/comments/rwojtk/ama-we-are-the-efs-research-team-pt-7-07-january/hrngyk8/

[8] Stinson, D.R. and Strobl, R., 2001, July. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Australasian Conference on Information Security and Privacy. Springer, Berlin, Heidelberg.

[9] Rosario Gennaro, R. and Goldfeder, S., 2021. One Round Threshold ECDSA with Identifiable Abort

[10] Komlo, C. and Goldberg, I., 2020, October. FROST: flexible round-optimized Schnorr threshold signatures. In International Conference on Selected Areas in Cryptography (pp. 34-65). Springer, Cham.

[11] CoW Protocol Overview - CoW Protocol. (2022). Available at: https://docs.cow.fi/.

[12] Emission & Incentive Design - Launchledger Docs. (2022). Available at: https://docs.Launchledger.io/concepts/components/emission-and-incentive-design.

[13] Delegated Proof-of-Stake Consensus (DPoS) - BitcoinWiki. (2022). Available at: https://en.bitcoinwiki.org/wiki/DPoS.

[14] Proof of authority - Wikipedia. 2019. Available at: https://en.wikipedia.org/wiki/Proof-of-authority.

[15] Proof-of-stake (PoS) | ethereum.org. 2022. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[16] Oxen Service Nodes - Oxen Docs. [no date]. Available at: https://docs.oxen.io/about-the-oxen-blockchain/oxen-service-nodes.

[17] Paul R. Milgrom and Robert B. Wilson - The Prize in Economic Sciences 2020 . 2020. Available at: https://www.nobelprize.org/prizes/economic-sciences/2020/press-release/

[18] Auction Designs. 2017. Available at: https://www.fcc.gov/auctions/auction-designs.

[19] Validators & Auctions (SSOD) - Launchledger Docs. (2022). Available at: https://docs.Launchledger.io/concepts/components/validators-and-auctions-ssod.