by Tariz from Radius

Summary

In designing a structure to achieve atomic execution between rollups, inspired by FlashLoan's trustless smart contract. The design verifies the trust assumptions needed between rollups to facilitate atomic execution. Three phases of proving:

1. Proposal Block Sequencing: Bundle transactions are included in all rollups' blocks.
2. State Execution: The execution of transactions by all rollups is valid.
3. Finalized Block Sequencing: All rollups have correctly reconstructed the block.

[

Architecture Overview

1920×631 71.1 KB

](https://ethresear.ch/uploads/default/original/2X/3/3d5b29d8c97c323737213d164506555cbacf6a90.jpeg)

Motivation

Rollups are an effective technology for improving the speed and cost of transactions on DApps. However, few teams have deployed their services using rollups, because it requires giving up the robust service operation experience provided by Ethereum's smart contracts. In other words, once they opt for rollups, they must consider aspects like censorship resistance, liquidity composability, liveness guarantees, and transaction execution.

We continuously explore and share methods to leverage the scalability of rollups while retaining the service operation experience of smart contracts, aiming to enhance these approaches.

Firstly, achieving Atomic Execution.

Trust Assumptions in Rollups

The difficulty in achieving Atomic Execution between rollups stems from the challenge of establishing trust assumptions between them

. From a user's perspective, rollups might seem to offer a superior experience compared to smart contracts, but this perception is ironically due to the trust assumptions between users and the rollup's Sequencer.

The Sequencer communicates the transaction outcomes (state) to users, who proceed with their next transaction under the premise that the state's validity is trustworthy.

Sequencer Role

User's Trust Assumptions

Sequencing

Receives transactions from users and decides which block to execute.

The Sequencer will determine the block without censorship, sandwiching, or front running.

Execution

Executes the determined block.

I will receive a valid execution result that adheres to the block determined in the Sequencing step.

Finalizing

Publishes the executed results on Ethereum for verification.

The result I receive will be finalized on Ethereum.

Creating the trust assumptions between rollups is impractical, given they are managed by different entities. This leads to a significantly inferior UX in cross-rollup bridges compared to smart contracts, resulting in liquidity fragmentation.

Concept

Let’s delve into the prerequisites for making assumptions verifiable to achieve Atomic Execution. Atomic Execution ensures that a bundle transactions included in each rollup’s block either all succeed or none are executed at all. Flash loans on Ethereum serve as a prime example of this concept. A flash loan is a set of transactions (a bundle) executed across multiple smart contracts on Ethereum.

Consider a Flash Loan involving two transactions executed in the smart contracts of Uniswap and AAVE: 1) Transaction A in Uniswap, 2) Transaction B in AAVE. The process of achieving atomic execution and the concept of trustless assumptions are as follows:

Trustless assumption: This refers to the notion that there’s no need for mutual trust. For instance, Uniswap does not need to trust the operation of AAVE smart contract.

Flash Loan Execution Process

Uniswap’s Trustless Assumption

Once a Flash Loan is included in a block by Ethereum’s validator, all transactions within the bundle are submitted to the designated smart contracts.

Transaction B is submitted to AAVE’s smart contract.

Each transaction is executed by the smart contracts according to precise logic.

The outcome of executing Transaction B is valid, regardless of failure or success.

If any transaction fails, other smart contracts that have already executed will revert their actions.

1) If Transaction A fails, the AAVE contract reverts Transaction B.

2) If Transaction A succeeds, the AAVE contract finalizes Transaction B.

Our design goal is to ensure that rollups get to the Uniswap’s trustless assumptions. The requirements to verify this are:

1. Bundle transactions are included in all rollups’ blocks.
2. The execution of transactions by all rollups is valid.
3. All rollups have correctly reconstructed the block.

We add a critical requirement to this list: Trustless Sequencing. This involves making the previously mentioned Sequencing Assumption trustless. In other words, it guarantees block determination without censorship, sandwich attacks, or front running, thereby strengthening trustlessness between users and rollups.

Before detailing the design, we assume rollups are structured as follows for efficiency:

[

Rollup

1635×1141 61.6 KB

](https://ethresear.ch/uploads/default/original/2X/6/6b242d9114636c9d159799dccb195e9c368938f8.png)

1. Rollups separate the Block Sequencing and Execution layers.
2. (Shared) Sequencer: A stateless node responsible only for block sequencing.
3. Executor: A full node solely responsible for block execution.
4. (Shared) Sequencer: A stateless node responsible only for block sequencing.
5. Executor: A full node solely responsible for block execution.
6. Rollups share a sequencer for interoperability, referred to as a shared sequencer.

Architecture Overview

[

Architecture Overview

1920×631 71.1 KB

](https://ethresear.ch/uploads/default/original/2X/3/3d5b29d8c97c323737213d164506555cbacf6a90.jpeg)

Phase 1. Proposal Block Sequencing

The Proposal Block is an initial block created by the shared sequencer to request execution from the executor. This block ensures that transactions from the promised user's bundle are included in all related rollups' blocks. During this process, all blocks are composed without censorship, front running, or sandwich attacks. This phase guarantees Atomic Inclusion

.

1. Commit: Promise the execution order of encrypted transactions.
2. Reveal: Decrypt the transactions.
3. Sequencing: Arrange transactions in the promised order to create a Proposal Block.
4. Proving: Prove the validity of the Proposal Block.

The Shared Sequencer has included all transactions within the bundle in the related blocks, following the promised order.

Phase 2. State Execution

The state is the result of the Executor's execution of the Proposal Block. This phase serves as the basis for proving the validity of the execution and deciding whether to revert the bundle

.

1. Execute: Execute the Proposal Block to compute the executed state.
2. Proving: Validation of the state's integrity.

The Executor has validly executed the Proposal Block.

Phase 3. Finalized Block Sequencing

The Finalized Block is reconstructed based on the execution results of the Proposal Block. It excludes (reverts) all failed transactions, including only those that were successfully executed. This block inherits the Proposal Block's resistance to censorship, sandwich, and front running. This phase guarantees Atomic Execution

.

1. Sequencing: A Finalized Block is created, excluding all transactions that failed in the Proposal Block, including those from any failed bundles.
2. Proving: Prove the validity of the Finalized Block.

Failed transactions were excluded from the Finalized Block in the committed order.

Phase 1. Proposal Block Sequenicng

1. Trustless Sequencing

Trustless sequencing refers to the process where blocks are determined without the need to trust the shared sequencer. The key aspect is that the sequencer preconfirms transactions to be included in the block while they are still encrypted.

This approach effectively makes attacks such as censorship, front running, and sandwiching, which require prior knowledge of transaction contents, virtually impossible. Thus, regardless of who becomes the shared sequencer among the rollups, it prevents the misuse of block creation privileges for unfair personal or collective gain, ensuring the most neutral sequencer. The process unfolds as follows:

[

Trustless Sequencing

3826×1322 247 KB

](https://ethresear.ch/uploads/default/original/2X/9/9d3c5a1059bbea395d190a9c34cca9e9853ee951.png)

1. Encrypt: Users encrypt their transactions based on a Timelock puzzle.
2. Commit: The Sequencer preconfirms the encrypted transaction.
3. Pre-confirmation: The order of transactions to be included in the Proposal Block.
4. Includes enc_tx, order, sequencer_signature.
5. Sends preconfirm_tx to individual users or stores it in a DA.
6. Pre-confirmation: The order of transactions to be included in the Proposal Block.
7. Includes enc_tx, order, sequencer_signature.
8. Sends preconfirm_tx to individual users or stores it in a DA.
9. Decrypt: The Sequencer solves the Timelock puzzle to decrypt the transaction.
10. Sequencing: Transactions are arranged in the promised order to form a block.

Trustless sequencing can incorporate various cryptographic techniques. When selecting encryption technology, we consider two key attributes:

1. Delay Encryption

: This attribute allows for the decryption to be delayed until the Sequencer is ready to commit publicly to the transactions.

1. In-protocol

: The Sequencer must have the capability to decrypt transactions independently, without relying on third parties, including users.

To achieve these attributes, we utilize the 'Timelock puzzle.' This cryptographic mechanism ensures that the decryption key can only be found after a specified time has elapsed, enhancing the security of transaction encryption and creating an environment where trust in the Sequencer's ordering actions is unnecessary.

Timelock Puzzle-based encryption could potentially lead to sequencing delays by malicious users. To maintain Sequencing Liveness, Radius's Practical Verifiable Delay Encryption (PVDE) allows users to generate a zero-knowledge proof (zkp) for the decryption key, enabling the Sequencer to identify transactions that could lead to decryption failures, thus preventing them. More details can be found on Ethereum Research Forum, and the Curie Testnet allows for exploration of implementation details and usability.

Our previously released Encrypted Mempool Open Source encompasses all functionalities of Trustless Sequencing. For more information, refer to the blog.

2. Bundle Sequencing

To ensure Atomic Inclusion, we describe how the shared sequencer distributes transactions from bundles across different blocks. When the shared sequencer preconfirms a user's bundle, it is a promise to include all transactions within the bundle in the rollup's block. This means that users and the rollup's Executors can obtain an inclusion promise for transactions that will be executed across multiple blocks solely through communication with the shared sequencer.

[

Trustless Bundle Sequencing

4278×1266 482 KB

](https://ethresear.ch/uploads/default/original/2X/7/74cc62677959e2e74b4cbafc9ec2219362669555.png)

Bundle Sequencing proceeds in the same order as the previously described Transaction Sequencing, thereby inheriting resistance to censorship, sandwiching, and front-running attack.

1. Encrypt: Users encrypt individual transactions and bundle them together.
2. Users can specify the encryption scope, where the rollup's address is not encrypted.
3. Users can specify the encryption scope, where the rollup's address is not encrypted.
4. Commit: The shared sequencer creates individual preconfirmations for the encrypted transactions, then bundles these preconfirmations into a single Bundle Preconfirmation.

5. Bundle Pre-confirmation: {preconfirm_tx_A, preconfirm_tx_B, sequencer_signature}
6. This serves as proof of the relationship between the bundle and transactions.
7. Bundle Pre-confirmation: {preconfirm_tx_A, preconfirm_tx_B, sequencer_signature}
8. This serves as proof of the relationship between the bundle and transactions.
9. Reveal: The shared sequencer solves the Timelock puzzle to decrypt individual transactions.
10. Sequencing: Sequences them in the promised order in each rollup's block.
11. Here, the commitment from the bundle preconfirmation is also included to identify bundle transactions.
12. Here, the commitment from the bundle preconfirmation is also included to identify bundle transactions.

3. Proposal Block Proving

Proposal Block Proving is crucial because it allows Executors to verify that all transactions from bundles preconfirmed by the shared sequencer are included in the rollup blocks.

[

Proposal Block Proving

1920×1086 132 KB

](https://ethresear.ch/uploads/default/original/2X/2/28ccaa3d9f88fc82395b7f545d295a34d8adbdc3.jpeg)

The shared sequencer stores the bundle preconfirmation in the DA during the commit phase. Then, in the sequencing phase, it stores the Proposal Block in the DA and records the Proposal Block's commitment on Ethereum. Executors can validate the Proposal Block's authenticity by comparing the information stored in the DA and on Ethereum.

1. Executor A verifies the preconfirm_bundle contained in Block A.
2. Searches the DA to find the preconfirm_bundle and verifies the preconfirmed transactions within the bundle.
3. Checks if Block B includes the preconfirmed tx_B.

Executors can directly verify preconfirmations on Ethereum with Vector commitments. This algorithm commits data composed of index-value pairs, ensuring the integrity between the index and value of committed data. Let's assume the promised order of preconfirmed tx_B is i

. If the vector commitment C

, which includes preconfirmed tx_B, is published on Ethereum before the verifying phase, the Executor can verify whether tx_B is included in the block in the promised order as follows:

[

Proposal Block Proving - vector commitment

2127×1188 168 KB

](https://ethresear.ch/uploads/default/original/2X/8/8e74fe6eaa139839b66b368984942c4a9ee87db0.png)

1. Executor A requests and receives a proof \pi

from the shared sequencer for tx_B. Here, \pi

is a proof that can verify tx_B's value is committed at index i

in the vector commitment.

1. Executor A sends {\pi

, i

, tx_B} to an Ethereum smart contract designed to perform the verification of the vector commitment.

1. The smart contract follows the predefined verification logic and uses a single pairing operation to verify that tx_B is committed in the $i$th order in Block B.

2. If the verification fails, the shared sequencer is subject to slashing.

Phase 2. State Execution

The state is the result of execution a Proposal Block. It's crucial because it verifies that a counterpart rollup has executed the transaction validly, determining whether the bundle should be canceled.

If the Proposal Block and State are public, participants can re-execute and verify them. However, re-execution for every block could lead to excessive computational costs for verification. Let's consider what participants must do to verify the state's validity:

1. Verify that the Proposal Block committed to Ethereum matches the existing commitment by comparison.
2. Execute a valid Proposal Block and compare the resulting State for consistency.

[

Re-Execution for Verifying State

3702×701 111 KB

](https://ethresear.ch/uploads/default/original/2X/0/02c03bdd435ee0a35158057d2e67474284a23fb0.png)

Since states are sequential, participants must repetitively perform this process from State #0

to #100

to verify the validity of State #100

.

One idea to reduce the complexity of verification is to use Recursive Proof. If a recursive proof is created for every State, verifying two statements, participants only need to verify the most recent recursive proof for validating the 100th state.

Considering the costs of off-chain re-execution versus generating recursive proof, off-chain re-execution might be more economical. However, this recursive proof could potentially be used in the finalized phase to validate the final state, possibly reducing the cost of verifying all blocks that occurred in that slot on-chain.

Phase 3. Finalized Block Sequencing

1. Finalized Block Sequencing

The Finalized Block is sequenced according to the following rules, inheriting the Proposal Block's resistance to censorship, sandwich, and front running attacks:

1. All failed transactions from the Proposal Block are excluded from the block.
2. The Pre-confirmation of the Proposal Block is maintained.
3. If any transaction within a bundle fails, all related transactions are excluded from the block.

For instance, assume the second transaction in the Proposal Block fails. The Finalized Block is then determined as follows: Because the Pre-confirmation is maintained, it inherits resistance to censorship, sandwich attacks, and front running.

Proposal Block

Finalized Block

1

1

2

3

3

The sequencing of the Finalized Block for bundle transactions proceeds as follows:

[

Finalized Block Sequencing

2928×1033 315 KB

](https://ethresear.ch/uploads/default/original/2X/9/9074afddfa73f8881bc908cca6b71913fd27f5bd.png)

1. The Shared Sequencer receives executed state A from Executor A.
2. If tx_A fails, it sends a Second Proposal Block to Executor B, excluding tx_B.
3. A proof demonstrating the failure of tx_A is also provided.
4. The Second Proposal Block is sequenced following the same rules as the Finalize Block Sequencing.
5. A proof demonstrating the failure of tx_A is also provided.
6. The Second Proposal Block is sequenced following the same rules as the Finalize Block Sequencing.
7. Executor B delivers the executed state B' of the Finalize Block B to the shared sequencer.
8. The Shared Sequencer finalizes the Finalized Blocks for rollups A and B.

Due to the exclusion of successful tx_B from the Proposal Block, Rollup B might face a cascading failure of subsequent transactions due to the changed state. However, it's important to remember that this isn't a case of intentional execution failure. Moreover, transactions should not be recomposed according to the Finalized Block Sequencing rules. Therefore, in the scenario described, the Finalized Block can be determined with just one additional communication.

[

Finalized Block Sequencing - 2

1821×722 48.8 KB

](https://ethresear.ch/uploads/default/original/2X/a/ad7cf00acc9e9634ddb52c495e430c3f9fdde3f2.png)

2. Finalized Block Proving

The ability to verify that a rollup has correctly reconstructed a block is crucial. The reconstructed block represents a set of transactions that were finally executed off-chain by the rollup and will be ultimately verified on Ethereum. Thus, this proving step ensures Atomic Execution.

[

Finalized Block Proving

4174×2376 499 KB

](https://ethresear.ch/uploads/default/original/2X/f/f559158b9c4e5d7ce83944dd438e8a1688ae9c36.png)

Continuing with the example, let's assume tx_A in Block A fails. For Atomic Execution, the Shared Sequencer confirms that tx_A has failed during the 'state proving' phase and removes tx_B from Finalized Block B before sending it to Executor B. Executor B then verifies the failure of tx_A, executes the Finalized Block, and generates state B' along with its proof. The commitment for Block B is published on Ethereum, and state B, state B', proof, proof', and the Finalized Block are published in the DA.

Executor A can infer from this information that tx_B has been validly removed from Block B and that tx_B is not included in Ethereum's finalization.

This process is efficiently verifiable when combined with last year's publication,'Efficiency-Improved Inter-Rollup Transfer System Leveraging Batch Settlement Methods'. This system uses batch settlement and zk-proofs to increase asset transfer efficiency between rollups. This design will be covered in a future post.

Conclusion

While recognizing rollups as a pivotal technology for enhancing the transaction speed and cost of DApps, we have examined the trust assumptions set for surpassing the user experience of smart contracts. Accordingly, we proposed a design that minimizes trust assumptions between rollups for seamless interaction while maintaining their independent execution environments. This design is preliminary, and we continue to explore ways to preserve the cost efficiency and speed improvements offered by rollups.

In future posts, we will introduce the development direction of Trustless Sequencing, a key requirement for rollups mentioned earlier. This concept is crucial not only for eliminating the possibilities of censorship, sandwiching, and front running but also for allowing any node within a rollup to technically become the most neutral shared sequencer.

We hope that DApps created with rollups will achieve the robust service operation experience provided by smart contracts. We always welcome all feedback and opportunities for research collaboration.