Kairos: Zero-knowledge Casper Transaction Scaling

Requirements

Marijan Petricevic, Nick Van den Broeck, Mark Greenslade, Tom Sydney Kerckhove, Matthew Doty, Avi Dessauer, Jonas Pauli, Andrzej Bronski, Quinn Dougherty, Chloe Kever

Tuesday November 28, 2023

Contents

1. Introduction

The Casper blockchain's ecosystem is in need of a scaling solution to achieve a higher transaction throughput and continue to stay competitive. As a first step towards providing a trustless scaling solution, the goal of the initial version 0.1 of the Kairos project is to build a zero-knowledge (ZK) *validium* [1] [2] for payment transactions in a second layer (L2). This system will both enable a higher transaction throughput and lower gas fees. Here, *validium* refers to a rollup where the data, such as account balances, are stored on L2 rather than on the Casper blockchain directly (L1).

Additionally, Kairos V0.1 serves two other purposes:

- It is the first step towards a cheap and frictionless NFT (non-fungible token) minting and transfer system aiding Casper to become *the* blockchain to push the digital art industry forward.
- The conciseness and complexity of its scope allow us to explore the problem space of L2 solutions that leverage zero-knowledge technology and integrate with Casper's L1. Furthermore, it allows the team to collaborate and grow together by building a production-grade system.

Kairos V0.1 will support very few simple interactions and features. Users will be able to deposit, withdraw, and transfer funds. These interactions will be serviced by the L2 and verified and stored by the L1, leveraging zero-knowledge technology. In the remainder of this document, we will detail the requirements of such a system.

In Section 2 (Product Overview) we describe the high-level features that Kairos V0.1 will support. Next, Section 3 specifies the requirements based on the described interactions and features. We conclude the document with a glossary, which clarifies the terminology used throughout this document.

Additionally, this document is accompanied by several blog posts detailing some of the design considerations in more detail, as listed in the bibliography [3].

2. Product Overview

To have a common denominator on the scope of Kairos V0.1, this section describes the high-level features it has to support.

2.1. User Characteristics

The target audience comprises users familiar with blockchain technology and applications built on top of the Casper blockchain.

2.2. Product Constraints

- The product's backend will be deployed on modern powerful machines equipped with a powerful graphics processing unit (GPU) and a large amount of working memory as well as persistent disk space.
- The operating machines will have continuous access to

the Internet.

- The CLI will be deployed on modern, potentially less powerful hardware.
- The product should be a centralized solution for this initial version 0.1.
- The applied proving system should be zero knowledge.

2.3. Features

The features in this section are associated with a tag, which is used as a prefix of the tags in Section 3 (Requirements).

2.3.1. Deposit Tokens Into L2 System

[tag:F00]: Users should be able to deposit tokens from their L1 account to their L2 account at any given time through a command line interface (CLI).

2.3.2. Withdraw Tokens From L2 System

[tag:F01]: Users should be able to withdraw tokens from their L2 account to their L1 account at any given time through a CLI.

2.3.3. Transfer Tokens Within the L2 System

[tag:F02]: Users should be able to transfer tokens from their L2 account to another user's L2 account at any given time through a CLI.

2.3.4. Query Account Balances

[tag:F03]: Anyone should be able to query any L2 account balances at any given time through a CLI. In particular, users can also query their personal L2 account balance.

2.3.5. Query Transaction Data

[tag:F04]: Anyone should be able to query any L2 transactions at any given time through a CLI.

2.3.6. Verification

[tag:F05]: Anyone should be able to verify deposits, withdrawals, or transactions either through a CLI or application programming interface (API), i.e. a machine-readable way.

3. Requirements

Based on the product overview given in the previous section, this section aims to describe testable, functional requirements the system needs to meet.

3.1. Functional Requirements

3.1.1. Deposit Tokens Into L2 System

- **[tag:F00-00]** Depositing an amount of `tokens`, where `tokens >= MIN_AMOUNT` must be accounted correctly: `new_account_balance = old_account_balance + tokens`
- **[tag:F00-01]** Depositing an amount of `tokens`, where `tokens < MIN_AMOUNT` must not be executed at all
- **[tag:F00-02]** A user depositing any valid amount (condition stated in F00-00) to their `L2 account` must only succeed if the user has signed the deposit transaction
- **[tag:F00-03]** A user depositing any amount with a proper signature to another user's account must fail
- **[tag:F00-04]** A deposit request shall not be replayable.

3.1.2. Withdraw Tokens From L2 System

- **[tag:F01-00]** Withdrawing an amount of `tokens`, where `user's L2 account balance >= tokens > MIN_AMOUNT` must be accounted correctly: `new_account_balance = old_account_balance - tokens`
- **[tag:F01-01]** Withdrawing an amount of `tokens`, where `tokens < MIN_AMOUNT` must not be executed at all
- **[tag:F01-02]** Withdrawing an amount of `tokens`, where `tokens > user's L2 account balance` should not be possible
- **[tag:F01-03]** Withdrawing a valid amount (condition stated in F01-00, F01-02) from the user's L2 account must be possible without the intermediary operator of the system
- **[tag:F01-04]** Withdrawing a valid amount (condition stated in F01-00, F01-02) from the user's L2 account must succeed if the user has signed the withdrawal transaction
- **[tag:F01-05]** Withdrawing any amount from another user's L2 account must not be possible
- **[tag:F01-06]** A withdrawal request shall not be replayable.
- **[tag:F01-07]** A withdrawal request must prevent double spending of tokens.

3.1.3. Transfer Tokens Within the L2 System

- **[tag:F02-00]** Transfering an amount of `tokens` from `user1` to `user2`, where `user1's L2 account`

`balance >= tokens > MIN_AMOUNT` must be accounted correctly: `new_account_balance_user1 = old_account_balance_user1 - tokens` and `new_account_balance_user2 = old_account_balance_user2 - tokens`
- **[tag:F02-01]** Transfering an amount of `tokens`, where `tokens < MIN_AMOUNT` must not be executed at all
- **[tag:F02-02]** Transfering an amount of `tokens`, where `tokens > user's L2 account balance` must not be possible
- **[tag:F02-03]** Transfering a valid amount (condition F02-00) to another user that does not have a registered L2 account yet must be possible.
- **[tag:F02-04]** Transfering a valid amount (condition F02-00) to another user should only succeed if the user owning the funds has signed the transfer transaction
- **[tag:F02-05]** A transfer request shall not be replayable.
- **[tag:F02-06]** A transfer request must prevent double spending of tokens.

3.1.4. Query Account Balances

- **[tag:F03-00]** A user must be able to see their L2 account balance when it's queried through the CLI
- **[tag:F03-01]** Anyone must be able to obtain any L2 account balances when querying the CLI or API
- **[tag:F03-02]** Account balances must be written by known, verified entities only
- **[tag:F03-03]** Account balances must be updated immediately after the successful verification of correct deposit/withdraw/transfer interactions
- **[tag:F03-04]** Account balances must not be updated if the verification of the proof of the interactions fails
- **[tag:F03-05]** Account balances must be stored redundantly [4]

3.1.5. Query Transaction Data

- **[tag:F04-00]** A user must be able to see its L2 transactions when they are queried through the CLI
- **[tag:F04-01]** Anyone must be able to obtain any L2 transactions when querying the CLI or API
- **[tag:F04-02]** Transaction data must be written by known, verified entities only
- **[tag:F04-03]** Transaction data must be written immediately after the successful verification of correct deposit/withdraw/transfer interactions
- **[tag:F04-04]** Transaction data must not be written if the verification of the proof of the interactions fails
- **[tag:F04-05]** Transaction data must be stored redundantly [4]

3.1.6. Verification

- **[tag:F05-00]** Anyone must be able to query and verify proofs of the system's state changes caused by deposit/withdraw/transfer interactions at any given time

3.2. Quality of Service

3.2.1. Performance

- **[tag:QS00]** The CLI should respond to user interactions immediately.
- **[tag:QS01]** The L2 should support a high parallel transaction throughput[1]

3.2.2. Security

- **[tag:QS02]** The application must not leak any private or sensitive information like private keys

3.2.3. Reusability

- **[tag:QS03]** The L2's API must be designed in such a way that it's easy to swap out a client implementation
- **[tag:QS04]** The whole system must be easy to extend with new features

3.2.4. Usability

- **[tag:QS05]** The CLI should be designed in a user-friendly way

4. Glossary

Validium Please refer to [1] and [2]

L1 The Casper blockchain as it currently runs.

L2 A layer built on top of the Casper blockchain, which leverages Casper's consensus algorithm and existing infrastructure for security purposes while adding scaling and/or privacy benefits

Zero knowledge proof (ZKP) Is a proof generated by person A which proves to person B that A is in possession of certain information X without revealing X itself to B. These ZKPs provide some of the most exciting ways to build L2s with privacy controls and scalability. [6]

Bibliography

[1] Ethereum, "Validium." [Online]. Available: https://ethereum.org/en/developers/docs/scaling/validium

[2] Nick Van den Broeck, "ZK validium vs. Rollup." [Online]. Available: https://github.com/cspr-rad/kairos-spec/blob/main/blogposts/validium-vs-rollup.md

[1] Read [5] for more insight into parallel vs. sequential transaction throughput.

[3] Nick Van den Broeck, “ZK provers: A comparison.” [Online]. Available: https://github.com/cspr-rad/kairos-spec/blob/main/blogposts/compare-zk-provers.md

[4] Nick Van den Broeck, “Data redundancy.” [Online]. Available: https://github.com/cspr-rad/kairos-spec/blob/main/blogposts/data-redundancy.md

[5] Nick Van den Broeck, “Sequential throughput of l2 ZK validia.” [Online]. Available: https://github.com/cspr-rad/kairos-spec/blob/main/blogposts/sequential-throughput.md

[6] “Zero knowledge proof.” [Online]. Available: https://en.wikipedia.org/wiki/Zero-knowledge_proof