Implementation of Account Abstraction for Boonty End Users

Abstract

To ensure Boonty is user-friendly to the masses while maintaining its Web3 ideologies, I propose the implementation of Account Abstraction. This feature enables new users to create a Web3 wallet (and a Boonty-usable account address) using their existing Web2 accounts, removing Web3 related obstacles and covering all related fees for reward transactions. This document outlines the processes required to implement the EIP-4337 (account abstraction) functionality and the benefits it brings.

This document details the processes required to implement the EIP-4337 (account abstraction) functionality, and the pros of doing so. It simplifies the process beyond Boonty's current API that requires an existing wallet address and completely avoids the need for consensus-layer protocol changes.

Introduction

Boonty aims to reach the masses and introduce them to Web3 by effectively trojan horsing the general public with a completely Web 2 interface. The current link between Web3 and Web2 is through an API that requires a wallet address. By implementing account abstraction, Boonty can completely obscure the Web3 layer, making it truly effortless for basic users to interact without any knowledge of extra Web3 steps, thus removing barriers to entry and providing easier adoption for the general population and businesses.

To implement this, this document introduces a higher-layer pseudo-transaction object called a *UserOperation*, which simplifies the process and avoids the need for consensus-layer protocol changes. Users send *UserOperation* objects into a separate *mempool*. A special class of actor called bundlers package up a set of these objects into a transaction making a *handleOps* call to a special contract, and that transaction then gets included in a block.

Motivation

The primary motivation for implementing account abstraction in Boonty is to simplify the user experience and encourage adoption among the general population, and the business that serve it. Requiring a Web3 wallet isolates a large part of the population, hindering adoption. By utilizing EIP 4337 (account abstraction), Boonty can tout the benefits of Web3 based rewards, such as data ownership and secure email access, while having a UX that is already second nature for the general Web2 crowd.
Additional use cases for account abstraction include privacy preservation and the ability to sponsor transaction fees for users - if they have to pay (even tiny amounts) for transactions it is just another obstacle against Boonty and Web3 integration. Furthermore, this feature

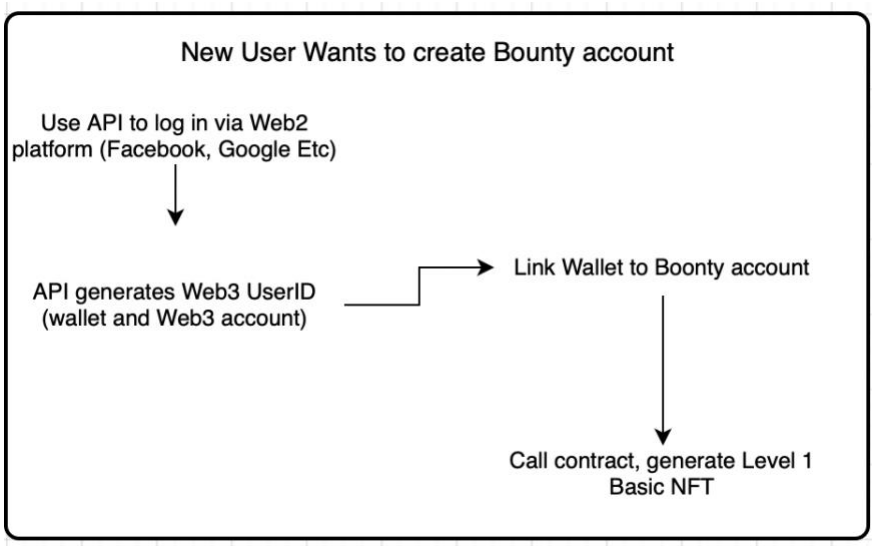
offers an easy solution for users who have forgotten their wallets or passwords, providing a seamless experience for account management and recovery. All of these pros benefit not just the user, but any Boonty stakeholder as it allows for a larger user base, and makes Boonty an easier sell to potential partners.

Specification

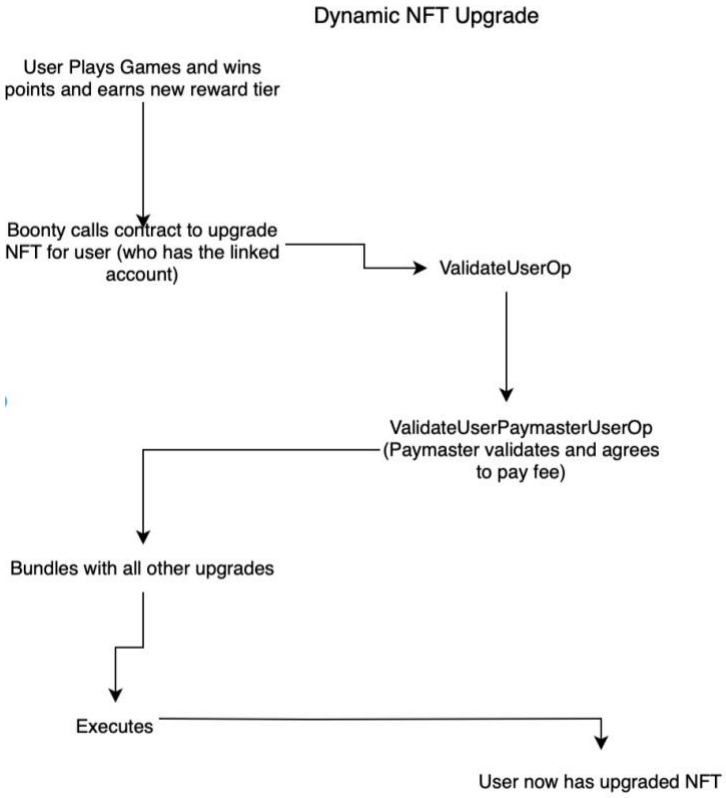
- **UserOperation** - a structure that describes a transaction to be sent on behalf of a user. To avoid confusion, it is not named “transaction”.
    - Like a transaction, it contains “sender”, “to”, “calldata”, “maxFeePerGas”, “maxPriorityFee”, “signature”, “nonce”
    - unlike a transaction, it contains several other fields, described below
    - also, the “signature” field usage is not defined by the protocol, but by each account implementation
- **Sender** - the account contract sending a user operation.
- **EntryPoint** - a singleton contract to execute bundles of UserOperations. Bundlers/Clients whitelist the supported entrypoint.
- **Bundler** - a node (block builder) that can handle UserOperations, create a valid an EntryPoint.handleOps() transaction, and add it to the block while it is still valid. This can be achieved by a number of ways:
    - Bundler can act as a block builder itself
    - If the bundler is not a block builder, it MUST work with the block building infrastructure such as mev-boost or other kind of PBS (proposer-builder separation)
    - The bundler can also rely on an experimental eth_sendRawTransactionConditional RPC API if it is available.
- **Paymaster** - a helper contract that agrees to pay for the transaction, instead of the sender itself – Might be helpful to set different paymasters per business
- **Aggregator** - a helper contract trusted by accounts to validate an aggregated signature. Bundlers/Clients whitelist the supported aggregators.

Architecture Diagrams

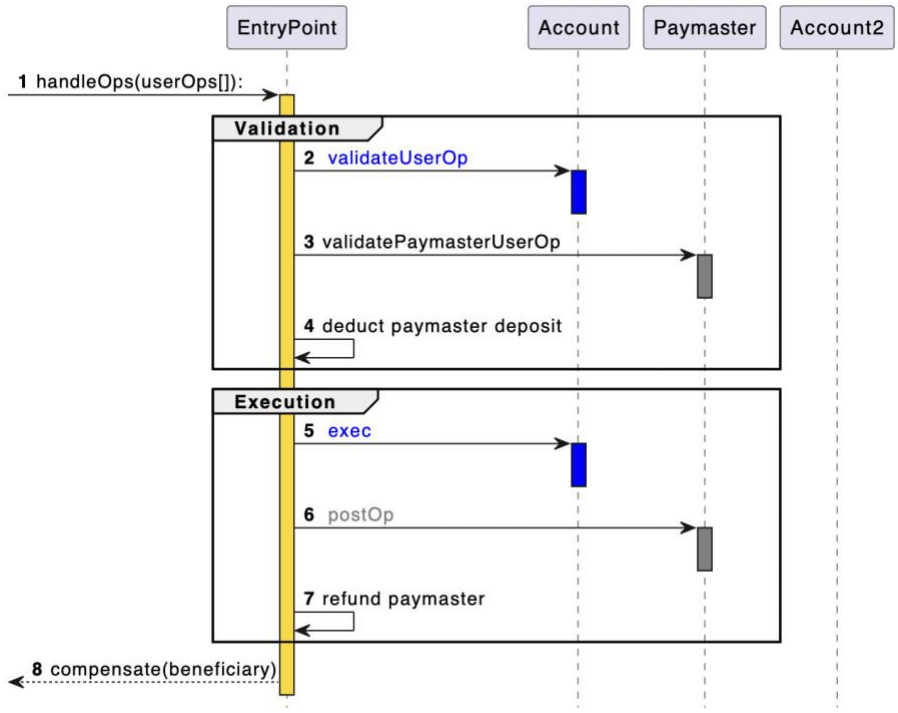
- ACCOUNT ABSTRACTION

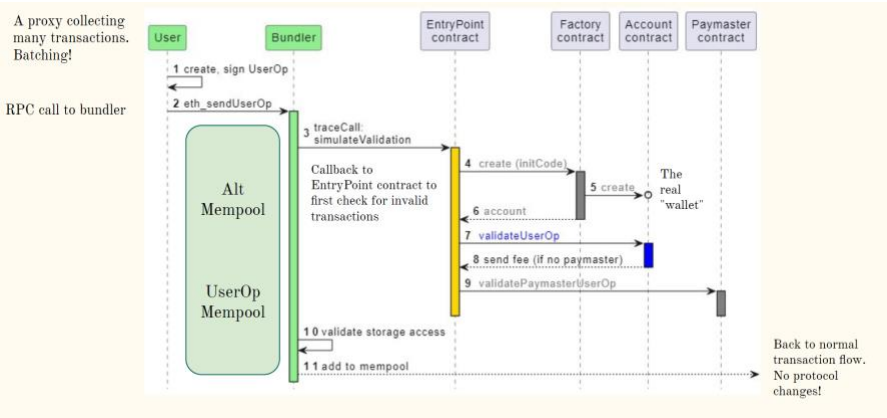


NFT Upgrade Logic



Technical Execution – from EIP 4337





Security considerations

- **Safety against arbitrary hijacking**: The entry point only calls an account generically if `validateUserOp` to that specific account has passed (and with `op.calldata` equal to the generic call's calldata)
- **Safety against fee draining**: If the entry point calls `validateUserOp` and passes, it also must make the generic call with calldata equal to `op.calldata`

Reference Implementation

This document is heavily based on the original EIP 4337 https://eips.ethereum.org/EIPS/eip-4337

Backwards Compatibility

EIP 4337 may not be easily compatible with pre-ERC-4337 accounts. Accounts lacking a validateUserOp function may need to be modified by creating an ERC-4337 compatible account that re-implements the verification logic as a wrapper and setting it to be the original account's trusted op submitter. However, for those customers with outdated wallets, creating a new wallet shouldn't take much effort.

Conclusion

Implementing EIP4337 is a crucial step in opening up the Boonty ecosystem to general Web2 users. It massively improves the ease of use by eliminating wallets, storing private keys securely and allowing for a smooth onboarding process. 'Gasless' or sponsored transactions allow Boonty to mimic more classical Web2 rewards platforms with all the usual Web3 benefits, which opens up the business to a wider clientele and end user base.