docs

Paymasters

Introduction

**Introduction**

One of the main reasons why the user experience of using EOAs is so difficult is because the wallet owner needs to find a way to get some ETH before they can perform any actions on-chain.

With paymasters, ERC-4337 allows abstracting gas payments altogether, meaning ​​someone other than the wallet owner can pay for the gas instead.

Gas abstraction offers plenty of benefits, such as:

* The need to acquire ETH before performing on-chain actions creates a significant friction point for users who are not crypto-savvy.
* Users don’t pay AWS fees for using web2 apps, so paying gas fees for using dapps could feel foreign and wrong to them. Paymasters allow dapps to sponsor those fees instead. Alternatively, dapps can instead allow the user to pay for gas in some ERC-20 token other than ETH, for example with USDC.
* To preserve privacy, a user can interact with assets in their account (for example, claimed tokens), without having to dox themselves by sending ETH from another KYCed wallet or CEX to pay for gas fees.

docs

Wallets

Supporting 4337

To support ERC-4337, wallets must implement a smart contract that is required to have two functions:

* validateUserOp, which takes a UserOperation as input. This function is supposed to verify the signature and nonce on the UserOperation, pay the fee and increment the nonce if verification succeeds, and throw an exception if verification fails.
* An op execution function, that interprets calldata as an instruction for the wallet to take actions. How this function interprets the calldata and what it does consequently is completely open-ended. However, we expect the most common behavior would be to parse the calldata as an instruction for the wallet to make one or more calls.

The ERC-4337 core team has implemented [SimpleAccount.sol(opens in a new tab)](https://github.com/eth-infinitism/account-abstraction/blob/develop/contracts/samples/SimpleAccount.sol), a sample minimal account that extends [BaseAccount.sol(opens in a new tab)](https://github.com/eth-infinitism/account-abstraction/blob/develop/contracts/core/BaseAccount.sol), which implements the [IAccount(opens in a new tab)](https://github.com/eth-infinitism/account-abstraction/blob/develop/contracts/interfaces/IAccount.sol) interface.

docs

FAQs

Reputation, throttling and banning

# Reputation scoring and throttling/banning for global entities

UserOperation's storage access rules prevent them from interfering with each other. But "global" entities - paymasters, factories and aggregators are accessed by multiple UserOperations, and thus might invalidate multiple previously-valid UserOperations.

## Staking

To prevent abuse, we throttle down (or completely ban for a period of time) an entity that causes invalidation of a large number of UserOperations in the mempool. To prevent such entities from carrying out [sybil-attacks(opens in a new tab)](https://en.wikipedia.org/wiki/Sybil_attack" \t "_blank), the protocol requires them to stake with the system, and thus make such DoS attacks very expensive. Note that this stake is never slashed, and can be withdrawn at any time (after an unstake delay period). When staked, an entity is also allowed to use its own associated storage, in addition to sender's associated storage. The stake value is not enforced on-chain, but specifically by each node while simulating a transaction. The stake is expected to be above MIN\_STAKE\_VALUE, and unstake delay above MIN\_UNSTAKE\_DELAY. The value of MIN\_UNSTAKE\_DELAY is 84600 (one day). The value of MIN\_STAKE\_VALUE is determined per chain, and specified in the "bundler specification test suite"

## Un-staked entities

Under the following special conditions, unstaked entities still can be used:

* An entity that doesn't use any storage at all, or only the sender's storage (not the entity's storage - that does require a stake)
* If the UserOperation doesn't create a new account (that is initCode is empty), then the entity may also use [storage associated with the sender](https://www.erc4337.io/docs/bundlers/running-a-bundler.md)
* A paymaster that has a postOp() method (that is, validatePaymasterUserOp returns “context”) must be staked