OST
Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

Account Abstraction - Demo

Thomas Bocek

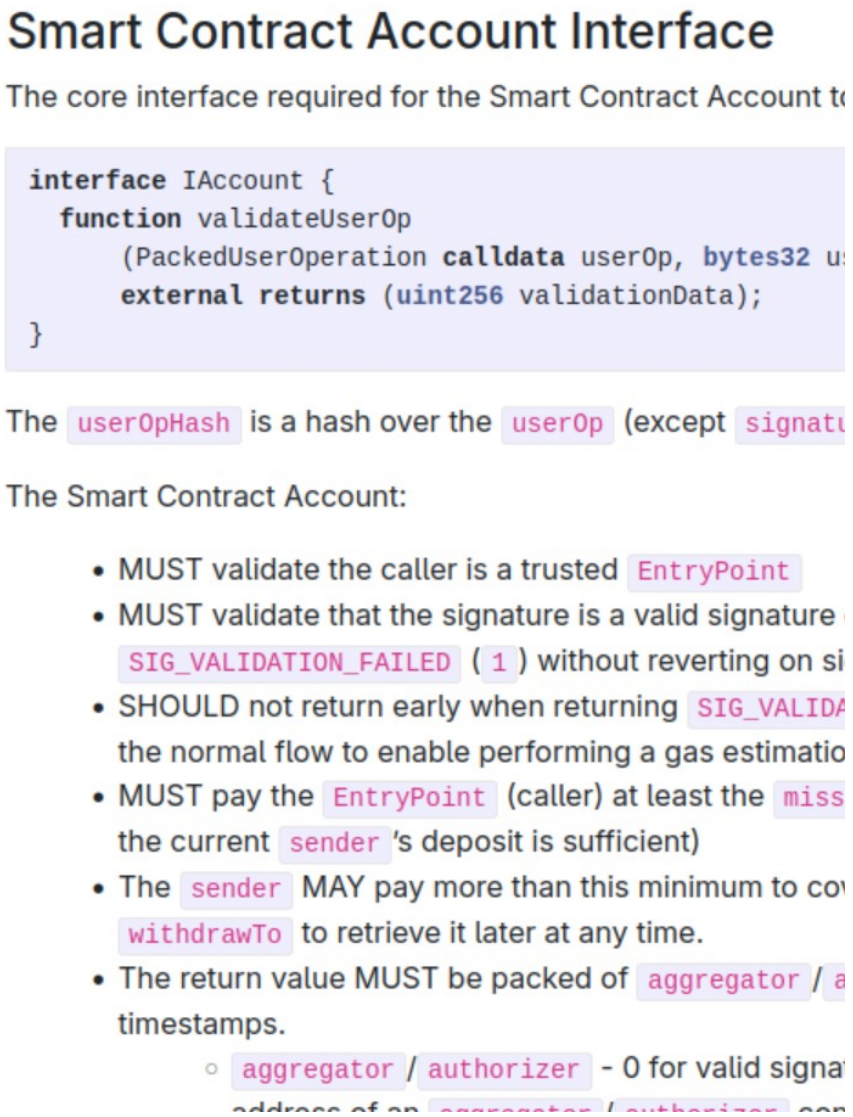
05.10.2025

Learning Goals

- Lecture 4
  - Demo of ERC 4337 Account Abstraction

ERC? What?

- Ethereum Request for Comments
  - Technical standard for Ethereum blockchain
  - Defines rules for smart contracts and tokens
  - Community-driven proposal system
  - E.g., ERC-20 (tokens), ERC-721 (NFTs), ERC-1155 (multi-tokens)
    - List: Final, Last Call, Review, Draft, Stagnant, Withdrawn
  - Solidity Interfaces
    - https://eips.ethereum.org/EIPS/eip-4337



Smart Contract Account Interface

The core interface required for the Smart Contract Account t

```
interface IAccount {
  function validateUserOp
      (PackedUserOperation calldata userOp, bytes32 u
      external returns (uint256 validationData);
}
```

The `userOpHash` is a hash over the `userOp` (except `signatu`

The Smart Contract Account:

- MUST validate the caller is a trusted `EntryPoint`
- MUST validate that the signature is a valid signature
  `SIG_VALIDATION_FAILED` (`1`) without reverting on si
- SHOULD not return early when returning `SIG_VALIDA`
  the normal flow to enable performing a gas estimatio
- MUST pay the `EntryPoint` (caller) at least the `miss`
  the current `sender` 's deposit is sufficient)
- The `sender` MAY pay more than this minimum to co
  `withdrawTo` to retrieve it later at any time.
- The return value MUST be packed of `aggregator` / `a`
  timestamps.
  - `aggregator` / `authorizer` - 0 for valid signa

OST

ERC-4337: Account Abstraction

- Ethereum's Path to User-Friendly Wallets
  - A loooong way
  - Implementation Without Protocol Changes
- Current Ethereum Account Limitations
  - EOAs require ETH for gas fees
  - Poor user experience for beginners
  - No transaction batching
- Core Concept
  - Smart contracts as primary accounts
  - Programmable transaction validation, authentication logic
  - Separation of signing and paying

```
interface IAccount {
    function validateUserOp(
        UserOperation calldata userOp,
        bytes32 userOpHash,
        uint256 missingAccountFunds
    ) external returns (uint256 validationData);
}
```

- Every ERC-4337 smart contract wallet must implement this
- Check if the UserOp signature is valid according to the wallet's logic

OST

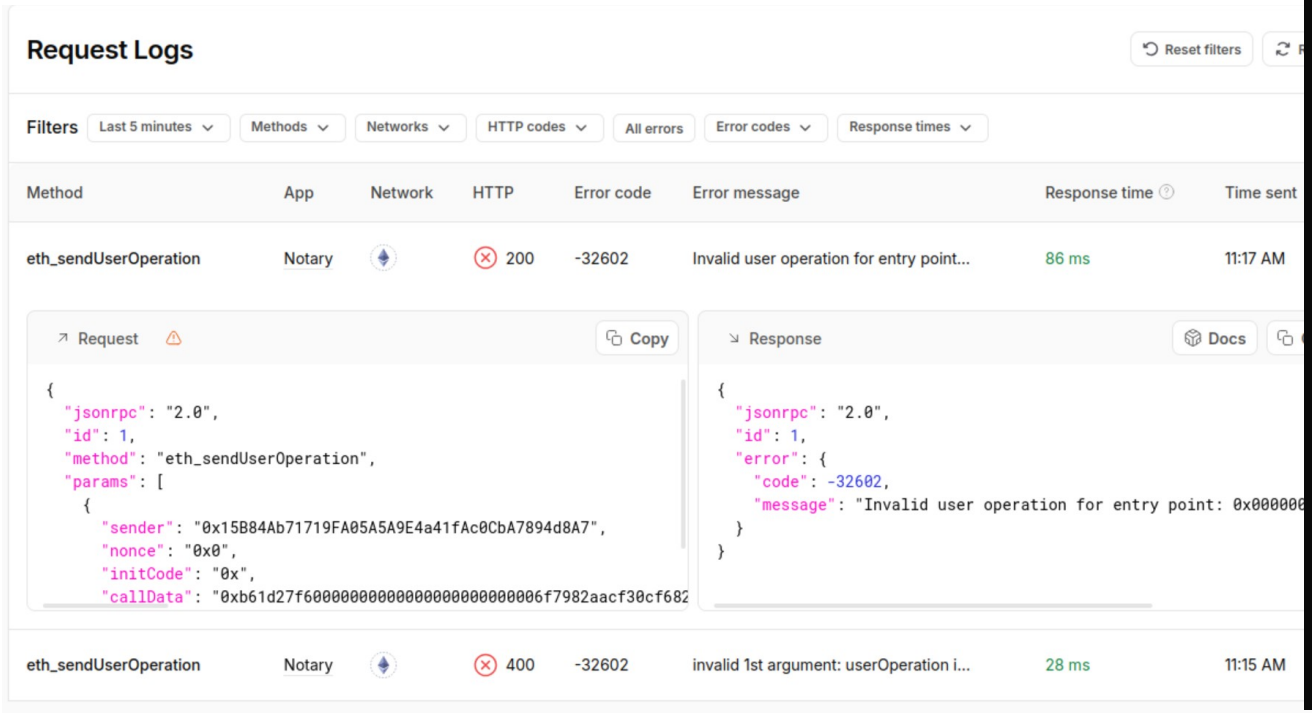
EIP-7702

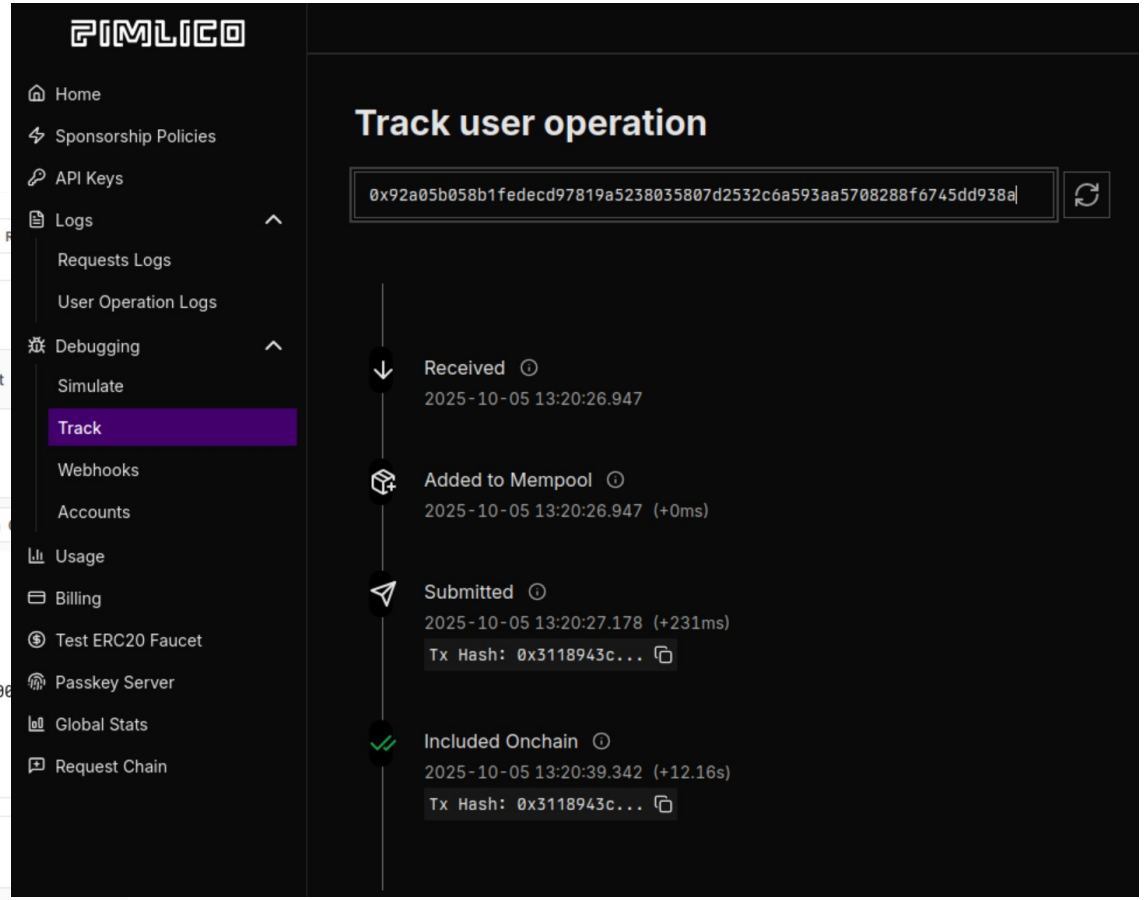
- ERC vs EIP
  - ERC (Ethereum Request for Comments): Standards for application-level specs (tokens, contracts)
  - EIP (Ethereum Improvement Proposal): Protocol changes/standards proposals
- What is EIP-7702
  - EOAs can temporarily set smart contract code by signing an authorization pointing to a delegation address
  - Attach a smart contract to your EOA
    - Your EOA runs that code but stays at your EOA address
- Key mechanics
  - New transaction type (0x04) adds contract_code field
  - Your EOA address stays the same, runs delegated code
  - No smart wallet deployment required
  - No per-user deployment needed
- ERC-4337 compatibility
  - Fully compatible with ERC-4337 infrastructure
  - EOAs can submit UserOps like smart contract accounts
- Security: **KNOW WHAT YOU SIGN**
- This demo: without EIP-7702

OST

Demo

- Notary Example
- Using 3rd part accounts
  - Pimlico / Alchemy





OST