**BÁO CÁO ĐỒ ÁN 1: NGHIÊN CỨU ACCOUNT ABSTRACTION**

Phạm Minh Đăng – 20194737

**I. Tổng quan về Account Abstraction trong Ethereum:**

- Ý tưởng về account abstraction dựa trên việc giúp cho người dùng có thể tương tác với mạng blockchain mà không cần thông qua ví như metamask, coinbase. Thay vào đó họ có thể dùng chính tài khoản google, facebook, etc để đăng nhập.

- Về cơ bản thì chúng là 1 higher level nằm trên kiến trúc blockchain của Ethereum sẵn có mà không làm thay đổi logic bên trong Ethereum

**II. Cơ chế hoạt động:**

**A blue and white diagram

Description automatically generated**

**Overview:**

- Mỗi người dùng sẽ sở hữu cho mình một **Smart Account** (đây là 1 contract), contract này sẽ thay thế người dùng trong việc thực hiện transaction, cầm các tài sản phi tập trung như **NFT**, **Coin**.

- Khi người dùng muốn thực hiện một transaction, họ sẽ tạo ra một **User Operation**, và các User Operation này sẽ được đưa vào 1 **mempool**. Ở đó có các bundler sẽ nhặt một số lượng **User Operations** và đóng gọi lại thành 1 bundle và gửi lên **Entry Point** để thực hiện nó.

- Ngoài ra người dùng có thể tích hợp thêm dịch vụ **Paymaster** để thực hiện **gasless transaction**.

**Smart Account:**

- là tài khoản của người dùng cuối. Ở mức tối thiểu, nó cần kiểm tra xem nó có chấp nhận **UserOperation** trong **Verification** hay không.

- Các tính năng bổ sung để hỗ trợ các chức năng khác của tài khoản như khôi phục mạng xã hội và đa thao tác cũng có thể được thêm vào đây.

**Paymaster:**

- là một thực thể có khả năng tài trợ phí gas của **UserOperation**. Cần phải làm hai việc:

- Kiểm tra xem nó có chấp nhận **UserOperation** trong vòng xác minh hay không.

- Trả bất kỳ logic phí bắt buộc nào trong **Execution**.

**User Operation:**

**-** Về cơ bản nó giống như một object, minh họa cho thông tin của một transaction. Dưới đây là các trường bên trong **User Operation:**

A screenshot of a computer

Description automatically generated

**Entry Point:**

**-** Về cơ chế **Aggregators**, như em được biết thì nó là một cơ chế để **validate signature** cho multiple smart accounts. Tuy nhiên thì về cơ chế này hiện tại em chưa nghiên cứu, trong report này sẽ hoàn toàn là về một **smart account flow** cơ bản.

- Về cơ chế **Verification,** đầu tiên nó sẽ check xem là **smart account** này đã tồn tại chưa, nếu như chưa tồn tại thì nó sẽ deploy 1 contract ở cái địa chỉ đó. Đây là cách nó tạo **smart account** vào đúng địa chỉ đó thông qua **create2**:  
 + Đây là 1 đoạn code trong thư viện “userop” của stackup, đầu tiên nó sẽ tạo ra 1 đoạn initCode và gọi lên **entryPoint** để tìm sender, sau đó sẽ lấy được address (kể cả chưa deploy contract) và nếu như **nonce** bằng 0 thì **initCode** = “0x”, còn không thì initCode sẽ là 1 đoạn hexify để thực hiện quá trình tạo 1 smart account mới ở địa chỉ đó

A screen shot of a computer program

Description automatically generated

+ Đây là đoạn code được thực hiện ở **EntryPoint**:

A computer screen shot of white text

Description automatically generated

A computer screen with text

Description automatically generated

A computer screen shot of a program

Description automatically generated

+ Sau khi account mới được tạo từ bunder thì sẽ gọi 1 hàm “initialize” để set lại owner mới và hàm này chỉ được gọi duy nhất 1 lần:

A screen shot of a computer

Description automatically generated

- Về cơ chế **Execution,** về cơ bản nó sẽ là việc thực hiện các hàm của **User Operation** thông qua calldata và gọi bằng phương thức **call** trong solidity

A computer screen with text on it

Description automatically generated

Ví dụ, ở đây em sẽ thử phân tích những bước hoạt động, khi em thực hiện send **0.01 ether** sang 1 ví khác và thông qua 1 **User Operation** (smart account này chưa được tạo) với smart account có địa chỉ là **0xfb2Ca0074ce9cB496316e63E5452A924A92A6F37** và chuyển 0.01 eth sang địa chỉ **0x0deB52499C2e9F3921c631cb6Ad3522C576d5484.**

A screenshot of a computer

Description automatically generated

Ở đây, 1 smart account sẽ được tạo và được entryPoint gọi hàm **initialize** để set lại **owner** mới.

A screenshot of a chat

Description automatically generated

Như ở đây mình có thể thấy là smart account send 0.01 eth cho địa chỉ **0x0deB52499C2e9F3921c631cb6Ad3522C576d5484.** Và 1 phần phí gas sẽ được gửi cho Entry Point ở address **0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789** và entry point sau khi thực hiện transaction sẽ trả lại 1 phần phí để gọi hàm cho bundler address **0x25Df024637d4e56c1aE9563987Bf3e92C9f534c0**

**Sequence Diagram của Entry Point:**

**A blue screen with text

Description automatically generated**

**III. Quá trình nghiên cứu:**

**-** Trong quá trình nghiên cứu, em đã research các tài liệu về EIP 4337.

- Ở giai đoạn đầu tiên em đã nghiên cứu một bên cũng triển khai chuẩn này là **Biconomy.** Trong quá trình nghiên cứu bên này thì em đã tìm hiểu được cụ thể về việc sharding private key trên **Torus node,** mặc dù trên README em cũng note khá nhiều nhưng em sẽ để lại những file README mà em đã học hỏi được từ giai đoạn nghiên cứu này (**Private key:** <https://github.com/perfogic/ERC4337/blob/main/private_key.md> và **Torus node:** <https://github.com/perfogic/ERC4337/blob/main/torus_architecture.md>)

- Ở giai đoạn sau, em đã tìm hiểu đến một bên khác là **Stackup.** Sau khi em nghiên cứu được cách họ triển khai các thư viện như **userop** thì em đã triển khai demo một ứng dụng về mua bán NFT trên marketplace sử dụng chuẩn **Account Abstraction** cùng với sự giúp đỡ của 2 thư viện chính là **userop** và **web3auth.**

+ **userop** là một thư viện sdk được viết bằng typescript, hỗ trợ em trong việc tạo **User Operation** và send nó lên bundler một cách dễ dàng.

+ **web3auth** là một thư viện sdk hỗ trợ em trong việc đăng nhập bằng google, facebook. Sharding và Recovery lại private key thông qua **Torus node.**

**IV. Demo sản phẩm:**

**- Login bằng web3auth:**

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**- Giao diện sau khi login:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**- Thử tương tác với marketplace, ở đây em sẽ chọn là listing một item.**

A computer screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* Sau khi listing xong, nft nhảy sang phần listing và nft cũ biến mất.

**- Transaction trên goerli:**

A screenshot of a computer

Description automatically generated

**V. Công nghệ sử dụng trong sản phầm demo:**

- Frontend: em sử dụng Next.js cho việc xây dựng giao diện

- Contract: em sử dụng hardhat cho việc testing contract và deploy contract lên mạng blockchain

- Indexer: em sử dụng Subgraph cho việc sync các dữ liệu mua bán trong contract.

- Third party: Moralis để sử dụng cho việc fetching toàn bộ nft thuộc về một user.

Github: <https://github.com/ERC4337-Dapp-Demo>, link website: <https://erc4337-five.vercel.app/>