**5장. 지갑 서비스 구축**

1. 온라인 지갑과 오프라인 지갑의 차이

- 지갑 : 계정들의 모임 , 계정 : 주소와 이에 연관된 개인 키의 조합

- 온라인 지갑 : geth내부, 어떠한 웹/DB에 저장된 지갑 => 위험성 있음

=> 제 3자를 신뢰해야하는 경우가 생길 수 있음

- 오프라인 지갑 : 지갑이 인터넷에 연결되어 있지 않음(USB,종이,Text File)

=> 훔치기 위해선 물리적으로 접근해야한다.=>온라인 지갑보다 안전

=> 문제점 : 실수로 지워지거나 잊혀짐 => 종이 형태로 금고에 보관

2. hooked-web3-provider

- sendTransaction()메소드의 모든 예제는 이더리움 노드에 존재하는 from

주소를 사용했으므로 브로드캐스팅 전에 트랜잭션을 서명할 수 있었음.

=> 지갑의 개인 키가 다른 곳에 저장되어 있으면 geth는 개인키를 찾을

수 없음 => 이경우는 web3.eth.sendRawTransaction() 메소드를 사용

- sendRawTransaction() => 원시 트랜잭션을 브로드캐스팅하는데 사용

=> 원시 트랜잭션을 생성하고 서명하기 위한 코드를 작성해야 함

=> 데이터 부분, 원시 트랜잭션을 생성하고 트랜잭션 서명이 필요함

- hooked-web3-provider 라이브러리는 HTTP를 사용해 geth와 통신가능

=> 이 공급자는 우리의 키를 사용해 컨트랙트 인스턴스의

sendTransaction 호출을 서명할 수 있게 해줌

=> 데이터 부분 생성 필요x, 메소드 구현 부분만 재정의하면 됨

3. ethereumjs-tx 라이브러리

- 이더리움 트랜잭션과 연관된 다양한 API를 제공하는 라이브러리

- 원시 트랜잭션을 서명하고, 트랜잭션이 올바른 키로 서명 됐는지 검사

- hooked-web3-provider와 ethereum-tx는 Node.js 및 클라이언트 측 자바

스크립트에서 모두 사용가능

4. 계층적인 결정적 지갑

- 시드라는 단일 시작 지점으로부터 주소와 키를 얻어낼 수 있는 시스템

- 만약 같은 시드를 사용하면 같은 주소와 키를 생성할 수 있음

- 계층적 : 주소와 키가 같은 순서로 생성됨 => 개별 키와 주소를 저장하

지 않고 시드만 저장하면 다수의 계좌를 백업하고 저장할 수 있음

5. 키 유도 함수

- 비대칭 암호화 알고리즘: 연관된 키가 필요 (ex RSA : 결정적)

- 대칭 암호화 알고리즘: 키의 크기만을 정의 키를 생성하는 것은 우리에

게 달려 있음 (ex KDF 키 유도함수)

- 키유도함수는 비밀 값(마스터 키,비밀번호,비밀구문)으로부터 대칭키 get

ex) bcrypt, crypt, scrypt, HKDF …

- 비밀번호 기반 키 유도 함수 => 비밀번호를 받아들여 대칭키 생성

6. LightWallet

- 트랜잭션을 생성하고 서명한 후, 이를 사용해 생성된 주소 키를 이용함

으로써 데이터를 암호화/복호화할 수 있는 API를 제공한다.

- keystore, signing, encryption, txutils라는 네 개의 namespace로 구성됨

=> singning : 트랜잭션 서명 , encryption : 비동기식 암호화,

txutils : 트랜잭션 생성 ,

keystore : 시드와 암호화된 키를 저장하는 객체

- keystore는 찾을 수 있는 주소에 대해 자동으로 트랜잭션 생성,서명 가능

7. HD 유도경로

- 다수의 암호화폐, 블록체인, 계좌 등을 처리하기 쉽게 만들어주는 문자열

- 원하는 만큼 많은 매개변수를 가질 수 있고 매개변수 다른 값으로 서로

다른 그룹의 주소와 연관된 키를 생성할 수 있다.

- 기본적으로 LightWallet은 m/0’/0’/0’ 유도 경로를 사용

8.

9.

10.