첫 이더리움 스마트 콘트랙트 개발 이야기

스포카 · 캐리 프로토콜
홍 민희

작년까지 하던 일

서버 프로그래밍

- 범용 프로그래밍 언어 (파이썬, C/C++, 하스켈)
- 비동기 네트워킹
- 관계형 데이터베이스 (PostgreSQL)

운영

- 리눅스
- 스크립팅 (bash, 파이썬)

캐리 프로토콜

2018년 6월 토큰 프리세일
첫 스마트 콘트랙트 개발

알고 있던 것들

- 튜링 완전한 기계 위에서의 프로그래밍 일반
- 암호학적 프리미티브에 대한 개괄적 이해
  - 암호학적 해시 함수 아이디어
  - 비대칭 암호화 아이디어
  - 서명, 검증, 암호화, 복호화 등의 아이디어
- 주력 프로그래밍 언어의 툴체인
  - Python: pip, virtualenv 등
  - Haskell: Cabal, Stack 등
- 애자일 프랙티스

도움이 된 것들

- 튜링 완전한 기계 위에서의 프로그래밍 일반
- 암호학적 프리미티브에 대한 개괄적 이해
  - 암호학적 해시 함수 아이디어
  - 비대칭 암호화 아이디어
  - 서명, 검증, 암호화, 복호화 등의 아이디어
- 주력 프로그래밍 언어의 툴체인
  - Python: pip, virtualenv 등
  - Haskell: Cabal, Stack 등
- 애자일 프랙티스

새로 배워야 했던 것들

- 튜링 완전한 기계 위에서의 프로그래밍 일반
- 암호학적 프리미티브에 대한 개괄적 이해
  - 암호학적 해시 함수 아이디어
  - 비대칭 암호화 아이디어
  - 서명, 검증, 암호화, 복호화 등의 아이디어
- 주력 프로그래밍 언어의 툴체인
  - Python: pip, virtualenv 등
  - Haskell: Cabal, Stack 등
- 애자일 프랙티스

생각을 바꿔야 했던 것들

- 튜링 완전한 기계 위에서의 프로그래밍 일반
- 암호학적 프리미티브에 대한 개괄적 이해
  - 암호학적 해시 함수 아이디어
  - 비대칭 암호화 아이디어
  - 서명, 검증, 암호화, 복호화 등의 아이디어
- 주력 프로그래밍 언어의 툴체인
  - Python: pip, virtualenv 등
  - Haskell: Cabal, Stack 등
- 애자일 프랙티스

이더리움 가상 머신 (EVM)

익숙한 특성들...

- 튜링 완전
- 고수준 언어로부터 바이트코드로 컴파일한다
  (예: 솔리디티)

이더리움 가상 머신 (EVM)

생소한 특성들...

- 결정적(deterministic)이다.
  → 같은 입력(nonce 포함)에 대해서는 항상 같은 결과
  → 암시적인 맥락에 의존적이지 않다
- 네트워크에 붙은 분산된 노드들에 의해
  중복되어(redundantly) 실행된다
  → 이러기 위해 결정적

이더리움 가상 머신 (EVM)

다른 비용 모형...

- **가스**: 공공 자원인 분산 컴퓨팅을 이용하기 위한 수수료
  → EVM은 물리적인 성능의 편차가 큰 다양한 컴퓨터에서 돌아가므로, 컴퓨팅 비용에 대한 일관적 모형이 필요하므로, 가스 같은 추상적 단위를 정의할 필요가 있음
- **가스 가격**: 1 가스 소모에 몇 이더(주로 gwei 단위)를 지불할지 정한다
- **가스 리미트**: EVM에서 돌아갈 스마트 콘트랙트 프로그램 코드가 얼만큼의 가스 이하로만 소모할 거라는 보장
  → 리미트를 초과하여 가스를 태우려고 하면 프로그램이 종료
  → 튜링 완전한 EVM에서는 정지 문제(halting problem)를 풀 수 없으므로 생긴 우회적 제약 개념
- 트랜잭션이 실패해도 가스 비용은 소모된다
- 컴파일러들은 수행 시간과 메모리 공간이 아니라 가스 소모를 최적화한다

개발 환경 차이

"로컬에서 충분히 개발하고 테스트한 다음 스테이징 서버에 올린다"

- 로컬에서 돌리기 위한 개발용 이더리움 싱글 노드 네트워크를 이용
  (예: 가나슈)
  → 기본 제공되는 계정들이 존재
  → 각 계정에 보통 100 이더 정도가 들어있음
  → 로컬넷이라도 가스비는 발생
- 스테이징 용도로 퍼블릭 메인넷과 테스트 네트워크를 이용
  (예: 롭스텐)
  → 필요한 이더는 각종 포시트에서 얻을 수 있다
  → 테스트넷이라도 가스비는 발생

풀 노드 (full node)

한국 평균 인터넷 속도가 주어졌을 때,
평범한 랩탑 컴퓨터(예: 최신 맥북 프로)에서,
퍼블릭 메인넷의 풀 노드를 구성하는 데에 한 달이 걸린다.

인푸라 (Infura)

- 클라우드로 제공되는 풀 노드 HTTP API 서비스
- 이름과 이메일 정도만 입력하는 것으로 가입 및 이용 가능
- 실제로 계정당 하나의 풀 노드를 제공하는 것이 아니라,
  풀 노드들 앞에 로드 밸런서가 있는 방식
  → 배포를 하거나 스크립트를 실행할 때 이전 트랜잭션을 만들 때 붙은 풀 노드와 이후 트랜잭션을 만들 때 붙은 풀 노드가 다를 수 있음
  → 두 풀 노드 사이에 동기화가 완료되지 않은 상태일 경우 nonce가 달라서 오류가 나기도 함
  → HTTP API 대신 웹 소켓 API를 쓰거나, 트랜잭션 사이에 시간 간격을 둬서 해결

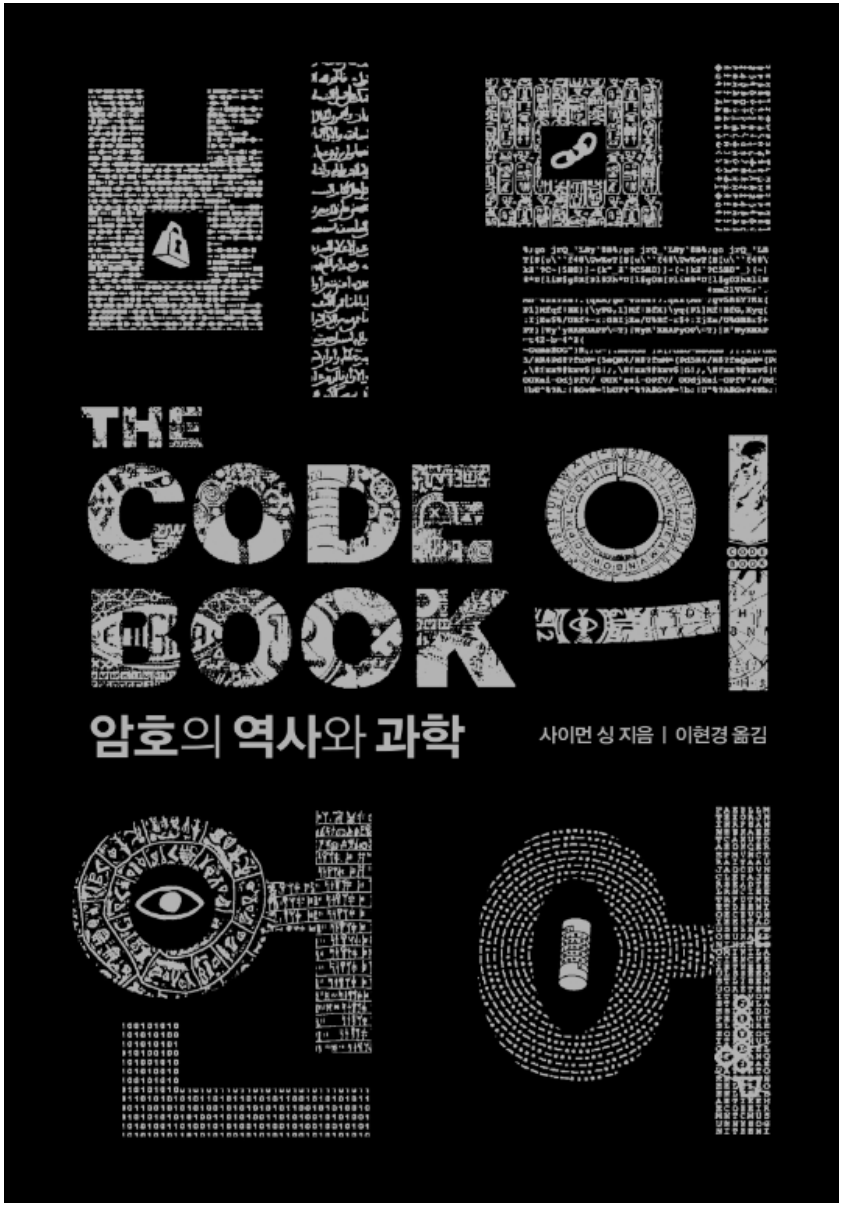
디버깅

- 스마트 콘트랙트 개발에서도 디버거는 있다
- 하지만 gdb 같은 전통적인 디버거보다 더 낫다
- 리액트 등의 시간 여행(time travel) 디버거에 더 가까운 개념
- 모든 스마트 콘트랙트의 실행은 트랜잭션 형태로 기록이 남고,
  입력으로부터 동작이 일과되게 동작되기 때문에 리플레이 가능
- 개념적으로는 더 편하지만, 아직 도구의 성숙도가 낮다는 단점은 존재

암호 화폐

- 블록체인은 암호학 프리미티브 위에서 만들어진 기술
- 이더리움도 여러 개념에 암호학 프리미티브가 녹아있다
- **계정**은 비대칭 암호화의 공개키 및 비밀키 관계
- **주소**는 공개키(로부터 유도된 다이제스트)
- **지갑**은 비밀키들의 집합
- **트랜잭션**은 디지털 서명

사이먼 싱《비밀의 언어》



솔리디티

https://github.com/ethereum/solidity

- EVM 바이트코드로 컴파일되는 고수준 언어
- 하지만 파이썬이나 자바스크립트처럼 고수준은 아니다
- 예를 들어 정수 자료형은 오버플로 등에서 자유롭지 않다
- `mapping(k => v)` 자료형은 딕셔너리가 아니다
  → `k` 에 대해 가능한 모든 값이 이미 존재한다는 아이디어
  → 따라서 길이 등을 가질 수 없고, 반복도 불가능
- 애초에 가스 때문에 추상화 수준을 한없이 올리긴 힘든 한계가 있다
- 오류가 나면 트랜잭션이 중단된다
  → 자바/파이썬 예외처럼 잡을 수는 없다

트러플 스위트

http://trufflеframework.com/

- 노드/자바스크립트 기반의 이더리움 개발 툴체인
- 솔리디티 소스 트리 통합 빌드
- 로컬 개발용 이더리움 싱글 노드 네트워크
- 테스트 프레임워크
- "마이그레이션" 기반 배포 시스템
- 애드혹 스크립트 실행기
- 디버거

오픈 제플린

https://openzeppelin.org/

- 웹 개발로 치면 장고, 레일스 같은 위치에 있는 프레임워크
- 현재 수요가 많은 다양한 케이스들의 구현을 제공
    - ERC 20/ERC 721 토큰
    - 토큰 세일
    - 접근 제한 (화이트리스트 등)
- 일부 동작을 커스터마이즈하기 쉽게 설계되어 있다
- 계약이라는 관점에서는 표준 계약서 같은 역할
- 직접 짜는 것보다, 조금 눈에 안 들어오더라도 오픈 제플린의 코드를 읽고 파악하여 활용하는 것을 추천
  → 보안 측면에서 이미 검증되어 있다
  → 계약은 당사자들 사이의 합의이므로, 알려진 코드가 합의하기 낫다
- 솔리디티 코딩의 베스트 프랙티스를 배울 수 있는 코드. 읽는 것 추천

아직 미숙한 개발 도구

역사가 짧기 때문에, 경쟁하는 언어·빌드 툴체인·프레임워크 등의 수도 적고, 아직 성숙도도 다소 떨어진다.

반면, 표준화가 시급한 부분에서는 미숙한 도구들이 난립하고 있기도.

유닉스 CLI 관례

```
truffle deploy --help
```

- 도움말 출력하는 대신 진짜 배포됨
- 트러플 스위트 공식 웹사이트의 문서를 찾을 것
- 문서화되지 않은 기능이 많기 때문에 현업에서는 코드를 읽는 것이 필수

산술 오버플로 (arithmetic overflow)

```
+ - * /
```

- (솔리디티) 산술 오버플로·언더플로 발생 가능
- 오픈제플린의 `SafeMath`를 쓸 것

테스트 픽스처 (test fixture)

- 트러플 스위트의 테스트 러너는 픽스처의 재현성을 보장하지 않음
- 이전에 실행된 테스트 케이스가 블록체인에 가한 변경이 사라지지 않음
- J유닛 등 일반적인 테스트 프레임워크의 테스트 픽스처처럼 repeatable하다고 가정하고 테스트를 작성하면 실패하므로 주의
- 예: 입금이 제대로 됐는지 확인할 때는 입금 직전의 잔고를 변수에 담고, 입금 후의 잔고와 차액을 계산해서 테스트한다.

자바스크립트

- 트러플 테스트 프레임워크는 테스트 코드를 솔리디티 또는 자바스크립트로 짤 수 있게 지원한다
- 솔리디티로는 표현력의 한계로 테스트하기 어려운 것들이 많다
- 자바스크립트로 테스트 코드를 짤 때는 자바스크립트에 정수형이 없고, `number` 타입은 부동소수점이라는 것을 염두에 둘 것
- `number`는 쓰지 말고 `web3.BigNumber`를 최대한 활용한다

애자일?

- 종래의 프로그래밍에서 당연하게 생각하던 애자일 방법론을 그대로 적용하기는 힘들다
- 애자일의 가정은 "소프트웨어는 건축과 달리 수정 비용이 싸다"는 것
- 하지만 스마트 콘트랙트는 배포하고 나서는 고칠 수 없다 (어렵다)

수정 애자일

- 그렇다고 애자일의 모든 것을 버릴 필요는 없고,
  애자일의 일부 프랙티스를 다른 프랙티스로 바꾸거나,
  특정 프랙티스를 더 철저히 실천하는 것으로 극복 가능
- 더 많은 테스트 코드는 언제나 도움이 되지만,
  스마트 콘트랙트에 대해서는 하면 좋은 게 아니라 필수다
- 스마트 콘트랙트 개발은 동료의 코드 리뷰가 더욱 빛을 발하는 분야
- 스마트 콘트랙트가 구현하려 하는 명세에 대해서 팀 모두의 합의 수준이 높아져야 한다
- Solium, solhint 같은 정적 분석 도구도 많은 도움이 된다
- 배포하기 전까지 테스트넷을 최대한 활용할 것
  → 시간 뿐만 아니라 거액의 비용을 아낄 수 있다

질문과 답변

발표 자료 링크: https://bit.ly/blockparty2-hong