<2주차>
블록 체인 : 중앙에서 관리하는 역할의 부재 - 자기 스스로가 관리해야함
white paper? 성능 향상?

거래(Transaction) : k에서 p로 암호화폐 전송
거래에 대한 내용은 장부(Block)에 저장됨. 브로드캐스팅 방식이 적용됨
장부안에 여러개의 거래가 존재.
거래에 대한 기록이 만들어짐 = 블록이 생성됨.
블록을 만드는 컴퓨터 = 채굴 노드(블록체인에 관한 모든 기능을 가지고 있음)
여러 컴퓨터중 가장 빨리 계산을 끝낸 컴퓨터가 채굴노드가 되는 것
(계산 시간에 대한 경쟁.)
생성된 컴퓨터에 프로그램적으로 자동으로 코인이 발부됨
계산을 뒤늦게 끝낸 컴퓨터에 대한 보상은 없음
이때 계산을 Mining(채굴)이라고 함.
블록은 결국엔 자료구조(헤더+바디) : 바디에 거래(트랜잭션)들이 들어감
모든 컴퓨터가 똑같은 트랜잭션을 모으는 것은 아님 = 컴퓨터마다 보유한 트랜잭션의 수가 다를 수 있음
블록생성 완료 -> 브로드캐스팅으로 결과 전송 -> 다른 컴퓨터(계산을 늦게 끝낸/못 끝낸)에서는 해당 계산이 올바르게 됐는지 검증을 하게됨
트랜잭션, 메모리 풀(Pool)
블록의 크기를 키워서 계산속도가 빨라짐??
Pool : 그룹 단위로 계산을 해서 분산시켜서 계산 처리 속도를 향상시킴
블록은 연결하는 체인 : 이전에 거래된 장부의 내용이 현재의 블록에도 포함되어있다.(예전의 기록들은 사라지지 않음) 실제로는 해시값
===p 1.16
Unconfirmed : 장부에 기록되지 않은 .. ex) 없는 코인을 전송하려고 하는 것 :
모아진 트랜잭션에 대해 해시값으로 블록에 반영함
현재 블록에서 해시값으로 이전 블록과 연결됨>?
Nonce(4바이트) : 0,1,2...대입해서 계산된 값이 어떤 값보다 작으면 계산이 완료된것. ->평균 계산시간을 구할수 있다.
계산 시간에 따라서 계산 난이도를 판별. 이를 조절해가면서 평균 10분(꼭)이 되도록 한다.
해당 과정을 통해서 블록이 생성되고 나면 Unconfirmed에서 Confirmed 트랜잭션으로 변경됨
Reward개념 : 비트코인을 계산한 컴퓨터 주소에 줌. POW : Proof Of Work
동작하는 규칙이 있음. 51:49이렇게 되면 49는 분리됨 (ex. BTc 와 BTC Cash)
POW=>POS(Proof Of Stacking) : 비트코인을 많이 생성한 컴퓨터에게 블록을 생성할 권리를 줌.
POS : 몇개 이상의 코인 있는 대상들로부터 랜덤으로 추출/?
===
p 1.18
비트코인 : address(본인의 잔고x) , 이더리움 : account(본인의 잔고에 대한 정보O -> smartcontract에 의해 증감-블록체인에 기록됨)
이더리움위에 어플을 올림. dApp 프로그램이 있다 = smart contract : 잘못된 부분에 대한 수정이 불가능함
블록체인 안에서 동작하는 프로그램
이더리움 플랫폼 : geth(Go 언어로 작성된 프로그램)가 설치된 컴퓨터//들이 블록을 생성해서 chain으로 연결됨.
smart contract : Solidity언어를 통해서 만들 수 있음.
dApp _ frontend : html.css.js에서 웹서버로 데이터를 전송
블록체인 안에 있는 정보를 서버를 통해 프론트로 보내야함. 웹서버와 블록체인 사이에 프로그램 필요(node.js 등등.. 이더리움에서는 web 3.0라는 라이브러리를 제공)
해당 라이브러리에는 거래, 블록체인 연결과 관련된 함수들이 존재함.
어떤 행위에 대해 보존이 필요한 정보에 대해서는 블록체인을 통해 저장하는 방식으로 사용가능
but 이를 기록하는 행위에는 비용이 발생한다.(smart contract 사용시 발생)
수정이 불가능하다는 단점이 있음.-> 무한루프에 대한 탈출이 불가능&코인 탕진
이더리움 사용전 test net을 통해서 미리 시험삼아 동작시켜볼 수 있음(테스트 툴 . remix 툴)

===week2
대칭키
비대칭키(공개키)
해시함수 sha256
====2.6
Encryption(암호화)를 하려면 키가 필요

암호화 : 보내는 정보가 변함.
바뀐 정보를 보는 수신자는 원래 상태로 복호화해야함
대칭키의 경우에는 암호화, 복호화에 동일한 키를 사용함 -> 속도가 빠름
대칭키 알고리즘 : DES, AES

===2.9
256비트에서 private 키 발행.
이를 통해 public 키 발행 / 까지는 시간이 많이 안걸림
how to find private key by ues public key? ->현실적으로 불가능
두개의 키.
===2.8
public, secret(private)키 중 하나로 암호화 남은 하나로 복호화
rsa 알고리즘(p 2.7) 비대칭키 : 시간이 오래 걸림
=> 빠른 속도가 필요하면 대칭키 사용
but RSA 사용이유? : 반대편 사용자에게 직접 키를 주는 방식으로 사용하면 중간에 해커가 침입해도 해당 데이터를 보는 것이 불가능하다.
공개키는 누구나 가지고 있음. 암호문은 해커가 가져가도 못품.수신자가 공개키, 비밀키 생성 -> 송신자가 수신자의 공개키를 사용해서
암호문으로 만들어서 전송. 해커는 이를 볼 수 있는 비밀키를 못 가지고 있음.
반대로 송신자가 자신의 비밀키를 통해 암호문을 생성하면 해커는 송신자의 공개키를 가지고 이를 볼 수 있음.
해당 행위에 대한 부정을 방지하는 기능
트랜잭션에 대해서 자신이 했다고 하는걸 알리는 행위가 된다. 발뺌이 안됨.ex) 자신의 서명 이라는 의미가 있다.
===2.12
해시 함수
해시 : 지문
Plain 텍스트 -> 해시함수 -> 해시된 텍스트(항상 같은 길이)
같은 값을 넣으면 항상 같은 결과를 반환
반대로 돌아가는 과정은 의미가 없다.(Pre-image Resistance)
다른 값을 넣었는데 같은값이 나올 가능성은 있지만 거의 불가능하다.

===p2.14
Plain text라는 문서를 암호화해서 보내려고 함
사이즈가 크면 암호화하는데 시간이 오래 걸림
문서에 대해 Digest Function(해시 실행해줌). -> 문서 내용이 변조 불가
나온 해시값을 Private키로 암호화 시킴(디지털 서명)
암호화된 문서를 원본 문서와 함께 첨부해서 보냄
수신자가 이를 받음
받은것 분리하고
송신자의 public키로 복호화함.->해시값이 결과값으로 나옴
plain text를 해시함수 시킨값과 비교 하여 같으면 정상적으로 데이터를 받은 것.(=원본 문서가 변조되지 않았음)
===p.15 디지털 서명
공개키로 풀린다 -> 본인의 비밀키를 이용한것 - 부인 방지
원본 문서의 내용이 조금만 바뀌어도 해시 결과가 바뀜 - 변조 방지
===p.17
검증하는 과정. 해시함수와 공개키가 필요함.
===p.19
서버를 믿을 수 있는가.
공개키는 전세상에 오픈되어있음.
다른 사람의 공개키를 사용하게 되는경우(또는 해커가 자신의 공개키를 가지고 보내는 경우)
즉, 보낸 사람의 공개키를 것에 대한 검증이 필요함.(보증, 인증과정.. by 국가)
디지털 인증서>>공개키가 들어가 있음. 소유자에 대한 정보도 들어가 있음.
보증해주는 사람에 대한 서명은 서명자의 Private키로 암호화 되어있음
실제로는 국가가 위임한 기관에서 이를 수행.
보증해주는 기관은 믿을만 한가.
===p.22
CA : 보증서를 발행해주는 곳. 보증에 대한 비용이 발생함.
CA의 Private키로 서명을 해줌.
송신자가 보증서를 믿을만 한지 보증해준 사람(CA)를 봄

이 CA에 대한 공개키를 통해 보증을 확인해서 실제 CA에서 발행한 보증이 맞는지를 확인
===p.33
이 과정이 root CA까지 올라가고 root CA에서는 Self 인증을 하게 된다.(더 올라가지 않음)
===p.34
https : 암호화된 데이터들을 주고 받기 때문에 보안성이 확보됨
===
네이버가 우리 브라우저에게 인증서를 보내줌. 각자에게 공개키 줌. 사용자는 자신의 비밀키를 가지고 브라우저에 접속?
브라우저에서 root CA를 보증해줌.
===p.32
디지털 인증서가 인증되는 과정에 대한 구조도
===p.33
X.509 : 인증서의 포맷
PKI : 인증에 대한 아키텍쳐

<3주차>
블록체인 기반 : 해시함수와 public key 알고리즘
공개키 알고리즘(public key) : 대칭키 & 비대칭키 2가지의
대칭키 : 암호키 = 복호화키
비대칭키 : 암호키 != 복호화키 // 비밀키(private key), 공개키
보증이 필요 -> 공인인증(PKI)
대칭키의 처리 속도가 비대칭키보다 빠름(task 속도가 빠름)
해시함수 : one-way : 보낸과정의 반대과정을 통한 추측이 불가능함
비밀키로 암호화 : 본인확인 할 수 있음.(보낸사람의 비밀키로 암호화 하면 그사람의 공개키로만 이를 복호화 할 수 있기 때문)
블록체인이기 때문에 보낸 과정에 대해서 수정.삭제 불가능.
===3.4
해시함수 : sha256 : 256비트(32바이트)의 결과값(문서의 지문)을 반환해줌.
===3.7
비트코인 : private -> public 과정으로 생성된 public키를 통해 주소를 생성
===3.8
수신자의 공개키를 통해 암호화 하고 수신자가 본인의 비밀키를 통해 복호화
===3.9
보내고자 하는 문서(블록체인 : 거래, 코인 송금)
Tx(transaction) 해시함수를 통해 지문(해시값) 만듬
해시값을 송신자의 비밀키로 암호화 = 본인 서명
해시값과 문서를 같이 보냄
그 후 수신자의 공개키를 통해 다시 암호화 -> 수신자만 풀 수 있음
수신자가 본인의 비밀키로 이를 풀고 나면 원본문서와 해시값이 나옴
송신자의 공개키를 통해 문서를 다시 hash함수를 적용하면
총 2개의 해시값이 존재. 이를 비교하여 같으면 원본문서에는 수정된 내용이 없이
잘 전달된 것을 알 수 있다.
===3.10
주소 생성에 대한 중앙통제가 없음 : 해당 주소가 누구의 것인지 알 수 없음
비트코인의 주솟 생성은 대칭키의 알고리즘.
256비트 -> 경우의 수 2의 256승 에서 1개를 선택한 것이 비밀키
비밀키를 통해 공개키를 만듦. 공개키를 통해 다시 해시함수를 거치는 등등의 과정을
거치고 나서 비트코인 주소가 만들어짐

거래소에는 내 비트코인 주소를 그대로 쓸 수 없음
거래소에 등록 => 거래소에서 다시 주소를 생성. 이는 누구의 것인지 다 알 수 있게됨
거래소에서 생성했다 = 비밀키를 거래소가 보관하고 있다.(거래소의 DB에)
거래소에 등록되지 않은 주소(거래소 밖에 있음)
===3.11
장부가 공개됨.
발행 : 채굴자(블록을 생성함 COM)에게 비트코인의 주소를 발행해서 채굴자의 지갑에 지급(=Reward)
LEDGER (=Block)
Tx-> ->Full Node(빨간원) // 평균 10분단위로 장부를 기록 해놓은 컴퓨터(Full Node)

bitcoin-core 프로그램(보통 리눅스에 설치) : 비트코인에 참여하고 있는 Full Node가 어디있는지 찾음
해당 블록을 다운로드 받아서 본인 컴퓨터에 Full Node 구성 가능
===3.12~13
암호화폐 지갑 프로그램
- 비트코인 : 지갑에 잔액 안보임(balance라는 잔고 영역/필드 가 존재하지 않음)
- 이더리움 : 지갑에 잔액(balance : 잔고 영역) 보임
비밀키를 통한 서명. 디지털 서명된 작업 ~ 비트코인 네트워크에 브로드캐스트 방식으로 전달됨
올바른 transactio인지를 검증 : node에서 공개키를 통해 확인(노드가 검증_이를 블록에 쌓음 mining/계산)
비트코인 결제는 신용카드와 달리 10분(블록 생성 시간)의 결제대기 시간이 있음
1시간(5~6개 블록)후에 안정적으로 결제가 이루어졌는지 확인이 가능함.

b'₩xe6₩x7f[z₩x19₩x1fB₩x15₩x18₩xb59₩xc6|₩x94₩x0f₩xb9' 16진수

e67f5b7a191f421518b539c67c940fb9 10진수

블록의 난스 값에 어떤값을 넣어야
내가 보내고자 하는 내용의 해시가 0000으로 시작할까 - 를 찾는 과정이 '채굴'
0000으로 시작하는 해시값이 발견되면 채굴됨.
하지만 해당 조건을 만족하는 난스 값은 하나가 아니라 여러개일 수 도 있음
이중에 가장 빠르게 찾는 사람에게 권한과 보상을 부여하는 것
난이도가 어렵다 : 10분이상
난이도가 쉽다 : 10분 이하
난이도를 높이는 방법 : 만족해야 되는 0의 개수를 늘리는 방법

===3.15
헤더 부분에 난스(미지수) 존재
prev black hash : 이전 블록의 해시값
timestamp : 블록을 만든 시간 - 장부의 순서가 바뀔 수 없도록 해줌
Tx : transaction 들 / 전체에 대해 해시함수를 통해 나온 결과가 Merkle root
Merkle root : 전체에 대한 해시값 들어 있음
Nonce 외에는 모두 고정된 값.
Mining - 채굴 / 무작위의 값을 모두 대입해보는 연산
===3.16
난스 값을 구하는 것 : pow(proof of work) : 보상을 준다.
이더리움과 비트코인은 블록의 구조가 다름
GPU(데이터를 빠르게 처리해줌)
이더리움 : proof of stake(pos : 지문 증명)으로 변경ing
지문 증명 : 블럭(장부)를 만듦. 누가 만듦?(pow :계산 빠름, pos:'이더'라는 지문을 많이 가지고 있는 사람에게)
무조건 많은 것만 보는 것은 아님.
if 30개이상에게 준다. => 70은 1, 35+35로 쪼개면 2.. 랜덤으로 블록생성 권한을 준다.
앞으로 : 아케텍쳐의 구현보다는 어떤 구성으로 아케텍쳐를 만들 것인가가 주.
===3.17
Bits : 앞에 붙을 0의 개수를 결정하는 = 난이도를 결정하는 요소
Coinbase Tx(transaction : 주소가 있는 작업) : 나의 비트코인 주소로 보내라는 내용을 적어놓은것(받는주소(나의 주소)만 있음/ 따로 송신자의 주소는 들어있지 않음)
화폐거래소에 대한 규제.
===3.18
체인으로 연결 : 앞번 블록의 해시값을 전달 받음
제네시스 블록(맨 처음 블록)에는 이전 해시값이 존재하지 않기 때문에 0으로 입력

===3.16
블록 : ledger : 거래장부 -> transaction(거래)
거래들을 정해진 양을 모아서 블록으로 만듦(누구나 만들 수 있지만 등록을 시키는 것은 consensus - 전체가 동의하는 규칙에 의해서만 가능
consensus : 가장 먼저 계산한(해시값을 찾은) 사람에게 블록 생성의 권한을 부여
reward라는 개념이 항상 따라다님. 가장 먼저 만든 사람에게 보상하는 것 (=권한 부여)
분산 장부 : distributed ledger : 분산 컴퓨팅(100%, 100%, 100%) & 중앙통제x
참고 문헌 : satoshi nakamoto - "bitcoin a peer-to-peer electronic cash system"
블록체인 전자 화폐 : 익명성 보장 / 인플레이션 문제 해결을 위해 시스템적으로 총 생산량을 제한함.(유한성)
'비트코인'이라는 플랫폼. : 암호화폐의 이름.중앙 통제 주체 없이 누구나 참여할 수 있음.(=p2p) : public block chain
<=>private block chain(ex. 기업 내부에서만 사용하도록 하는 경우)
비트코인, 이더리움은 모두 public block chain
최초의 블록 : Genesis Block(원래는 보통 0부터 시작함)
public : 조회는 모두 가능. wallet을 통해 나에게 속한 비트코인만 옮길 수 있음
그러면 장부란 어떻게 생긴것인가
장부블록 : 헤더와 몸통(바디)로 구분
<헤더>
Prev Block Hash : 이전 블록의 해시값
Consensus : 누가 장부에 기록을 할 것인가? = 누가 블록을 생성할 것인가? 라는 것에 대한 합의가 필요함 - 비트코인 외에는 생성자에게 문의를 통해서 수정이 가능하나 비트코인은 누군지 알 수 없다.
POW : Proof Of Work : 해시값을 구하는 것. 이 미지수가 Nonce
Nonce의 값을 구하는 것을 mining한다고 함.
Nonce 외의 값은 모두 알고 있음
Merkel Root Hash : transaction들의 해시값(body의 내용)을 저장하고 있음
주어진 조건을 만족하는 Nonce의 값은 여러개이다. 그중에서 먼저 찾은 사람에게 장부에 기록할 수 있는 권한을 가지게 되는것
장부는 공개가 되어있기 떄문에 누구나 분석할 수 있음.
쪼개지는 것 : fork(부모 -> 자식)
soft fork : 마이너하게 버전을 바꾸는 것
hard fork : 호환성이 없는 것. 구조 자체가 바뀌는 것.
0번에서 hardfork 되어서 BCH : 비트코인 클래식 / BTC 로 나뉨
genesis 블록의 prev hash 값은 0
genesis 블록에 대해 계산한 hash 값이https://www.blockchain.com/explorer/blocks/btc/0에
해시값 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f이 있다.
Nonce : 2,083,236,893 이 넌스 값을 가지고 계산을 하면 해시값이 나옴? : 앞에 나오는 0이 10개가 되는 값을 찾아내는 것인데
Nonce값을 넣어서 계산해보니 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f가 나왔다.
mempool.space
난이도가 올라가는 것 : 앞에 와야하는 0의 개수를 늘리는 것
걸리는 시간을 보면서 적절한 시간을 넘기는 경우에는 0의 갯수를 줄인다.
===3.17 헤더에 관한 정보가 정리된 사이트
Height : 블록의 번호
Merkle Root :
Minted : 발행된 액수 (시스템 상으로 2100만개로 제한되어있음)
210,000이 될 때마다 반감기 설정(주는 비트코인의 수를 반으로 줄임 - 50 - 25 -12.5 -....현재 : 6.25 ..3.125...점점 0으로 수렴. 채굴이 끝남
Bits : 난이도에 대한 정보.
Miner : 채굴한 사람
==번외
a번째 블록의 해시값은 a+1번째의 블록의 헤더에 기록됨
===블록의 내용을 볼수 있는 사이트

===4.3
블록의 헤더 : 이전 블록 해시값, nonce(pow)
블록의 바디를 모드 hash를 해서 Merkle root
모든 Full node는 블록을 구성함
만든 블록이 블록체인에 등록되는 것? 가장 처음에 만들어진 것만!
time stamp : 블록을 구성할 때의 시점
bits : 난이도 (평균10분에 1개의 블록 생성을 목표로 설정) : 0의 개수(target value)
모든 블록은 최소 1개의 transaction을 가지는데 이를 coinbase transaction이라고 함
transaction : 거래 (누가 누가에게 얼마를 송금하는지) 정보 기록
은행의 시스템과는 달리 중앙통제 역할이 존재X
따라서 transaction에 중앙통제의 역할을 대신하는 것이 필요(=> 암호)
블록체인의 장부 : 모두에게 공개되어있음
===4.4~4.6
블록의 정보(Deatil / blockchain.com)
hash : 32byte= 256bit
merkle root
difficulty
nonce
bits
minde on
height
transaction = 4라고 하면 (1개의 coinbase transaction. 나머지는 거래에 관한것)
Minted : 발행된 것
fees
reward ; 보상(minted+fees)

header_hex = ("01000000" #버전 정보 거꾸로 읽어 온 것(00/00/00/01)4바이트 리틀 에디안 방식 +
"81cd02ab7e569e8bcd9317e2fe99f2de44d49ab2b8851ba4a308000000000000" +# Prev Block (125551) Hash
"e320b6c2fffc8d750423db8b1eb942ae710e951ed797f7affc8892b0f1fc122b" +# Merkel Root
"c7f5d74d" + #원래 값 : 4dd7f5c7 # Timestamp
"f2b9441a" +# Bits = 0x1A44B9F2 = 440,711,666
"42a14695")

버전정보-이전 블록의 해시값-merkle 값-블록의 구성시간

125551번째 블록(이전블록)의 해시값 : 00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048 또한 2개씩 끊어서 역순으로 정보가 들어가있음

merkle : 2b12fcf1b09288fcaff797d71e950e71ae42b91e8bdb2304758dfcffc2b620e3

https://www.epochconverter.com/

4dd7f5c7 : GMT: 2011년 May 21일 Saturday PM 5:26:31 / Your time zone: 2011년 5월 22일 일요일 오전 2:26:31 GMT+09:00

난이도(125551) : 440,711,666 (HEX 변환)

nonce : 42a14695 (HEX 변환)

===4.7
비트코인 블록 구조
transaction의 구조:
누가 누가에게 얼마를(value or amount),공개키 알고리즘(☆public : for locking, private : 서명의 용도)
-> 기록된 장부에 대한 소유권 주장시 private key로 unlocking
Hard Fork 분기를 통해서 bitcoin, bitcash로 분할
bitcoin에만 Marker와 Flag 정보가 존재.
standard tx, legacy tx / segwit tx

===5.2
블록체인 : 거래기록의 장부 - ledger
거래 : transaction-> 블록의 바디부분에 들어가있음

pow : 가장 먼저 nonce값을 찾는 블록을 채택함

트랜잭션도 체인으로 되어있다. : 이전 블록의 바디에 존재하는 transaction의 해시값도 가지고 있음.

A로 서명함 = hash
===5.5
ownership : locking :unlock을 할 수 있다. : 소유권이 있음

utxo : 사용하지 않은 거래를 모아두고 있음.
(unspent Tx output)

거래를 할 때 input으로 들어온 utxo(lock되어있는 상자 / 보내는 사람의 공개키로 잠겨있는 블록)을 보내는 사람의 비밀키로 unlock해야함
unlock하는 순간 stxo(spen tx output = 빈 상자)이 된다.
그 다음 받는 사람의 공개키로 이를 다시 locking
===5.5
utxo의 input에 unlocking script가 있음
여러곳으로 들어가는 input , 여러 곳으로 output이 갈 수 있음

보내고 남은 코인은 나에게 오는 output으로. - 남은 금액에 대해서는 다시 utxo로 보관

===5.11
tb1xx : 테스트
비트코인 커뮤니티

===5.12
비밀키 : 256비트 = 32바이트, 가짓수가 2^256
컴프레스드, 언컴프레스드 종류에 따른 공개키
공개키를 가지고 hash를 2번함.

비트코인 소유
1.광질
2.교환(구매)
빗썸- 암호화폐 거래용 계좌
비트코인 지갑 : 비트코인 주소를 가지고 있음(레거시, 위트니스, 네이티브)
빗썸에 자체 지갑이 있음.
빗썸이 사용자의 수만큼의 비트코인을 소유(사용자 지갑에 들어있는게 아님)
소유한 비트코인에 대한 기록을 DB에 저장해놓고 있음
내 코인을 내가 소유하기 위해서는 외부로 가져와야 함.
원화로 바꾸기 위해서는 거래소로 내 코인을 가져와야함.

혹은 다른사람과 실거래.
거래소로 가져오려면 비트코인 주소가 필요함.
->빗썸에 주소(private key)를 생성해달라고 요청.
빗썸이 만든 것이기 때문에 우리가 private key는 알수가 없음
자신이 주소를 잃어버리면 거래소인 빗썸에서도 도와줄 수 가 없음(개인의 요청에 의해서 만들어진것)
==========
이더리움

public bc : 누구나 이용 가능(비트코인, 이더리움)
private bc : 특정 그룹/회사가 모여서 서비스를 이용하기 위한(hyper ledger fabric)
이더리움 : 플랫폼에 어플리케이션을 개발해서 올릴 수가 있다.

이더리움 화이트 페이퍼
IPO(Initial Public Offering) : 주식 시장에 주식을 상장함.
ICO (Initial Coin Offering) : 투자하면 이더리움 코인을 발행해 주겠다.(Presale)
비트코인 : POW
이더리움 : POW이기는 하지만 비트코인의 방식과는 조금 다름. // Ethash 방식 ~> POS로 바뀌는 중(proof of staking:지분 점령? 증명?)
smart contract : D웹(웹개발+smart contract: ★solidity라는 언어를 통해서 개발함) 개발시 들어감.
크립토 커렌시 : BTC / ETH
해시 알고리즘 / 이더리움 : public private 모두 있지만 balance라는 은행 잔고와 비슷한 시스템이 들어감
새성 시간 : 이더리움 평균 15초 당 1개의 블럭이 생성됨.

ASIC 기반 채굴 : 해시 계산만 빠르게 하도록 만든 칩.(인텔칩이 아님)
GPUs 기반 채굴 :
scalable : 더 많은 사람이 더 빠르게 처리가 되는가> POS 방식으로 넘어갔을 때를 기준.

이더리움 : 거래 기반의 상태 머신(=컴퓨터 = VM)
geth -> 동작 -> 컴퓨터
이더리움에서의 표현 : World-Wide-Computer : 전세계에 대해 하나의 컴퓨터가 존재한다고 생각하는 방식.
이더리움이 실행하는 프로그램 : smart contract 코드 / 컴파일. 링크 후 실행파일/
solidity라는 언어로 스마트 컨트랙프로그램을 만듦. test.sol 과 같은 파일을 컴파일하고 내부적으로 링크를 함.
그 과정을 거치면 실행파일이 생김
test.java와의 차이점
java는 컴파일하고 링크를 하면 실행파일이 생김. 해당 실행파일은 기계어로 생성되는 것이 아님. java Byte code로 생성됨
이를 실행하기 위해서는 java VM(JVM)이 필요함.
solidity도 실행파일은 eturium byte code가 생성됨(기계어가 아님) 이를 기계어로 바꾸기 위해서는 이더리움 가상머신이 필요함(EVM)
상태(state) : 내가 이더리움을 얼마나 가지고 있는가에 대한 balance
거래를 하면 state가 변하는 것.(계좌의 balance가 바뀜)
상태가 변하게 된 원인은 이더를 송금한것(=거래가 발생함) => 거래를 기반으로 상태가 변하는 것 - 이를 프로그램을 통해서 관리를 하는 것이 이더리움이다.
이더리움 : 거래가 송금도 되고 프로그램이 실행되게 하는 요청이 될 수도 있다!
이더리움 거래 : 상태를 변화시키는 것!!

===evm.pdf

state : 이더리움에 참가하는 모든 컴퓨터는 동일한 state를 가짐.

world state0 -> Transaction -> world state1
Transaction들이 모여서 블록을 구성하는 것은 동일
POW 핵심 : nonce 값

이더리움 지값 > 이더리움 주소(20바이트)
key, value // dictionary 타입 : 주소가 key, 상태가 value : balance (key - value : mapping) 주소 - account의 상태값 들을 모아 놓은 것이 world state
level db
거래가 발생하면 거래 당사자들의 state가 변함.(EVM에서 처리하는 내용)
solidity에서 key-value mapping 기능 지원
어떻게 이를 구현할 것인가. (구현 관점 : 테이블, 객체 등등) 가장 최적화 되어있는 db가 Level DB(no SQL)
객체 관점 : Account라는 data structure가 존재하는 것.

```
typedef struct _abc{
        int balance
        char* hash.Root
        ...
} Account;
```

Account Alice; 로 객체 선언
Mapping view(spec) - 구현 - Table view

Account에 들어가는 정보(4)
  nonce : 이더리움에서 넌스는 2개 // 보내는 사람이 몇개의 transaction을 보냈는지에 대한 카운트, 난이도를 맞추기 위한 값이 아님
★balance
  storage hash (account storage : 데이터를 영구적으로 저장하기 위해 필요한 공간) :
  code hash (evm byte code)/smart contract : 코드가 1bit라도 수정이 되었는지를 확인하기 위한 hash 값.
        (블록체인에서 짠 코드는 한번 짜면 수정이 불가능함)

storage hash, code hash는 account의 종류에 따라서 있을 수도 없을 수도

smart contract가 이더리움 플랫폼에 deploy되면 모든 작업은 주소를 기반으로 동작함.
smart contract는 사람이 아니기 때문에 account를 가지는데 이는 주소로 되어있다.

Account의 종류 : 사람 / Smart Contract - 2가지
사람이 갖는 정보(EOA) : Externally Owned Address : storage hash, code hash = null
Smart Contract가 갖는 주소에 연결되는 account (CA) : Contract Account : storage hash, code hash != null
===p.19 (★EOA, CA의 차이 구분)
external actor : 지갑
contract account : smart contract를 만들어서 deploy 하면서 생긴 주소가 매핑된 account
비트코인 : 트랜잭션의 블럭을 찾으려면 해시값 사용
이더리움 : 이더 스캔 : 블럭과 트랜잭션(해시값), smart contract(deploy 했을 때 생성된 주소를 통해 소스코드를 볼

수 있음)

dApp은 smart contract 1개인가? (x) : dApp 은 smart contract + Web //일부
deploy : 블럭안에 들어간다(웹 코드는 블럭안에 안들어가고 smart contract 코드만 들어감) 웹서버 프로그램은 블록체인과 관련 없음
deploy를 한다 = Transaction으로 smart contract를 심음
만든 사람(sender)의 지갑을 가지고 Transaction을 만듦
설치 :

사람의 지갑으로 생성 : private->public key 생성 - hash -> 이더리움 주소 생성
contract account : deploy시 생성 : sender address() + nonce -> 이더리움 주소 생성
누군가가 지갑을 생성해도 state가 변함.(지갑 생성도 Transaction에 의해 생성된 것이기 때문)

dApp의 핵심은 스마트 컨트랙
비트코인 : 코인은 있지만 토큰은 없음 유틸리티 토큰>
이더리움 :
코인 : 코인 있음 / dApp은 코인 못만듬 / 코인은 플랫폼에서만 사용 /
토큰 : dApp에 의해서 동작 / 인센티브 발급을 위함(무한하게 가능?) : 많이 참여한 사람에게 / ERC-20 : FT, ERC-729 : NFT / 토큰을 몇개 발행할지 정할 수 있음

===원래 파일 17
비트코인의 블럭 구조 : prev blk hahs, none, time stamp, diff, 머클티리 해시값(트랜잭션 해시값)
비트코인 : script : input/output - 튜링불완전(반복문 사용 불가능) 스크립트 언어

이더리움 블럭의 구조
: 이전 블록의 해시값(parentHash), timestamp, diff, 트랜잭션과 관련된 해시하는 동작을 담당하는 트리?가 3개(state어카운트, transaction, receipts영수증 //비트코인에서의 해시는 transaction)
solidtity : 튜링 완전(반복문 사용 가능_while) 스크립트 언어 : 무한 반복 트랜잭션에서 발생할 수 있는 문제를 해결했음. how? evm 위에서 실행할 때 기름을 쓰듯이 사용. 기름=이더 / 이더를 다 쓰면 반복문이 끝남.
dApp을 플랫폼에 올리기 위해서는 Transaction을 발생시켜야함. 이때 이더리움도 이더 라는 수수료를 받음 ..
smart contract : 함수들의 모음 / 이를 실행시키기 위해서 이더를 사용함.
=>★블럭의 헤더에 gasLimit, gasUsed 가 존재하는 이유
smart contract = EVM byte code
add, subtract 등 실행에 필요한 자원들이 모두 정해져 있고 이를 미리 계산해서 알려줌. 반복문에 따라서 전체 사용량이 변함
-> 무한 반복문이지만 실제에서는 무한하지 않게 되어있음.
잘못 코드를 작성해도 수정이 안된다. 계좌의 balance가 0이 되면 멈춤 - 그동안 진행된 사항들이 reset 됨.

beneficiary : 혜택 / 채굴한 사람에게 감.
extradata : 데이터를 이더리움 플랫폼에 저장할 수 있음. : 데이터를 블록체인에 작성하게 해줌.
mixHash : 채굴과 관련된 내용/

여기까지 시험범위 (이더리움 헤더부분)

이더리움 evm p.19
state : account의 state / account : EOA해서 사용자가 갖게 되는 정보
스마트 컨트랙트 ..
nonce,balance, storage hash, code hash

모든 것은 Transaction에 의해서 일어남
actor : 사용자
smart contract도 Transaction을 생성함

contract 생성 : solidity언어 사용
contract 파일을 컴파일, 링크 -
smart contract 코드도 블록에 들어가있음

EOA : 사용자 account
메시지를 통해서 데이터, 이더 등을 보냄
CA : Contract Account
message call : smart contract안에 들어가있는 함수를 호출. = 함수 실행
함수 실행 -> (1) 실행해서 데이터 보여주기(기름/gas 불필요) (2) 데이터를 변화시키기(기름/gas 필요)
EOA - EOA : 송금
EOA - CA : 함수 실행

smartcontract(객체지향) 배포시 최초로 실행되어야 하는 부분 : 객체지향에서 constructor에 해당하는 부분 = init code(발행할 토큰의 개수를 지정할수 있음)
코인과 토큰
코인 : 플랫폼이 생성해주는 것 (화폐발행)
토큰 : 유틸리티 토큰 : 서비스를 위한 asset
비트코인 : 코인은 있지만 토큰은 없음 / 서비스를 생성할 수 있는 플랫폼이 아님
이더리움 : 코인도 있고 토큰도 있음 / 서비스를 생성할 수 있음 (by smart contract)
토큰을 몇개 발행할 것인가 : init code
배포가 되면 contract 주소가 생성됨, account 생성됨 (account의 상태에 code, storage 들어감)
smart contract : 함수들의 집합
함수 호출 by transaction & function name
함수의 인자에 해당하는 데이터가 들어갈 수 있음. input data로 들어감.
호출 후에는 state가 변경되는 것임
storage에 들어가는 데이터는 smart contract의 코드를 변경하지 못하는 것과는 별개로 변경이 가능하고 변경이 되면 state가 바뀌는 것임
gas와 이더는 다른 것.
wei : 1/10^18
v, r, s : 해당 transaction에 대한 유효성을 검증하기 위한 파라미터
to : 160bit : 20byte : 이더리움 주소
EVM이 world state를 변경시킴(transaction을 보내면(smart contract 생성 후 배포 등), 외부에서 내부에 접근 할 때 web 3를 사용함.)
이더리움 안에 있는 정보는 transaction에 의해서 변경됨 = web 3 interface 를 통해서 = js의 함수를 통해서
이더리움 설계도(white paper)에 따라서 코드를 작성 = 클라이언트 프로그램(클라이언트 프로그램의 종류 : geth, parity?)

week07pdf

jvm이 자바 프로그램을 컴파일 하고 링크해서 bytecode(생성코드) 만듦 = CPU에 종속적이지 않음 . 종속적인건 jvm이 가지고 있음
solidity - 컴파일 - 빌드 - 이더리움 바이트 코드(기계어x) 이를 실행하기 위한 프로그램 EVM
EVM 내부 구조
Program counter, Stack(코드는 스택기반으로 실행됨), memory, gas available(코드안에 사용되는 함수에 대해 얼마만큼의 gas가 필요한지 작성되어있음)
smart contract 실행 - byte code 실행 - 필요한 gas에 대한 대략적인 양이 gas available에 적혀있음
but 왜 대략적?? 정확하지 않은 이유 :
if / else 각자마다 실행됨에 따른 gas양이 다를 것이기 때문./ 어떤 부분이 실행될지를 아직 모르기 때문에 정확한 계산 불가능
for문 : 몇번 실행될지 정확하게 알 수 없음
=> 대략적인 수치는 알 수 있지만 정확하게 알지 못하는 경우도 존재할 수 있다.
실행 중 gas가 모자란 경우에는 smart contract의 실행이 멈춤.
-> 완전한 실행x = 지금까지 실행한 모든것이 무효 처리됨 - 소비된 gas는 반환해줌. 하지만 transaction 발생에 대한 수수료는 반환이 되지 않음
EVM code : immutable : 수정이 불가능함.
world state : persistent 없어지지 않음
memory에 저장된 것은 없어질 수 있음.

state transition
Tx 블록.
Block x -> Block x+1 : transaction
sequential 하게 처리가 되면서 state가 변함.

이더리움 블록
ommers list : uncle 블록 생성.
비트코인의 경우 : 거의 동시에 채굴이 된 경우 이들 모두가 유효한가? 더 길어지는 chain이 유효한 것으로 간주함.
이더리움 : 긴 것이 인정을 받기는 하지만 다른 쪽(uncle block)도 약간의 보상을 줌. - 이를 헤더에 반영을 함.(ommers list)
또한 uncle block과 관련한 내용이 헤더에 들어감.

비트코인 Merkle Tree = 이더리움 Transaction Root

이더리움
★stateRoot (MPT를 통해서 관리함)
receipts Root

Address -> account (key -> value 관계)
account 안에 balance가 있음
많은 데이터를 어떻게 효율적으로 관리할 것인가!

블록의 구조

비트코인 블록헤더 : prev hash, nonce, timestamp, bits(난이도), Tx hash //비트코인의 첫번째 tx : coin based transaction

이더리움 블록헤더 : parentHash, nonce, timestamp, difficulty, TransactionRoot //이더리움 : coin based Transaction 없음,

이더리움에만 존재하는 요소들
ommerHash : uncle block에 관한 hash
gas (Limit, Used):
number : count
beneficiray : coin based Transaction과 유사한 기능??
mixHash :

(비트코인은 먼저 계산하는 것이 유리 : ASIC칩으로 해시 연산에만 강한 성능을 가짐) | 이더리움에서는 ASIC으로 채굴하지 못하게 하여 공평하게. ASIC에 저항성을 갖도록 설계한 방법 : ethash(POW)
ethash : 반드시 메모리에 access하도록 되어 있음.(속도의 한계가 있음)
cpu안에 cache가 있는데 이를 ASIC에 집어 넣으면 이더리움도 채굴 빠르게 가능하지 않을까?=> cache를 써도 용량의 제한이 있음
이더리움 : 최소 2gb가 기본사양. ASIC 안에 2gb의 용량을 만들 수 없음. 반드시 DRAM을 거치게 되어있음
이더리움 방식에서 채굴을 빠르게 하기 위해서 사용하는 것이 그래픽 카드
이를 계산하는 과정에서 mixHash를 사용함.

state의 정보를 모두 hash한 값이 stateRoot에 저장됨.
state trie의 용량은 매우 클 것임.
비트코인 (평균 10~15분) 이더리움(10~15초)마다 state가 변경됨. 이더리움의 경우에 방대한 용량을 관리하기 위한 방법이 필요함(p.19)
key : value의 형태(data structure)로 사용이 됨.
실제로는 이를 저장할 때 level db(key : value 형태의 db)에 저장 // mysql, oracle : R-DBMS , level db : key : value에 특화된 db
n번째 - transaction - n+1번째 블록 (key : value)에 해당하는 데이터의 크기 지금까지의 모든 데이터를 저장받는 것은 불가능(10~15초마다 state가 변하는걸흠)
상태가 변한 사람과 변하지 않는 사람이 있는데 한쪽이 바뀐다고 양쪽 다 바꿔야 하는 것은 비효율적!
상태가 변한 것만 다음블록에 반영하자!(포인팅) / 상태가 변하지 않은 것은 이전블록에 그대로 유지시킴

Merkle Patriciea Trie(MTP)
Practical Algorithm To Retrieve Information Coded In Alphanymeric

Radix Tree // search
1~7 : key 첫번째 글짜부터 공통부분
각각의 노드에 대해서 hash
key : 이더리움 주소
value : account 주소??

총 4종류의 node
Branch node
마지막에 value가 들어가는 노드 : leaf 노드(state의 정보가 들어감)
extesion 노드 : 같은 노드가 있으면 확장.
니블 : 4비트
prefix :

ㅁ : odd nible's first

짝수개 : leafnode의 prefix에 2
홀수개 : leafnode의 prefix에 7
마지막 leafnode에서 hash256

변경되지 않은 것은 포인트 , 변경된 것만 바꿈(p.26)
state root에서 이전 블록을 포인트 : 바뀌지 않은 정보들
state root에서 현재 블록을 포인트 : 바뀐 정보들

Public Key : ECDSA-Secp256k1(Private Key)
비트코인 : 더블 해시
이더리움 : Keccak-256
 나온 해시값에서 끝의 20바이트 앞에 0x붙이면 이더리움 주소가됨