W2. Mechanics of Bitcoin

▼ Cryptography for Blockchain

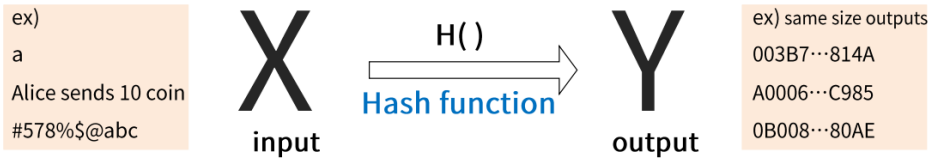
- 목차
  - 해시 함수
  - 블록체인을 이루는 자료 구조 - 해시 포인터, 리스트
  - 기본적인 암호화 기법
  - 디지털 서명 원리

1. 해시 함수

- 블록체인의 암호화를 위해 사용됨

<특징>

- **메시지 축약 기능**
  - 입력 값으로 어떤 크기의 데이터를 넣든, 어떤 타입의 데이터를 넣든 동일한 크기의 출력 값을 도출해냄

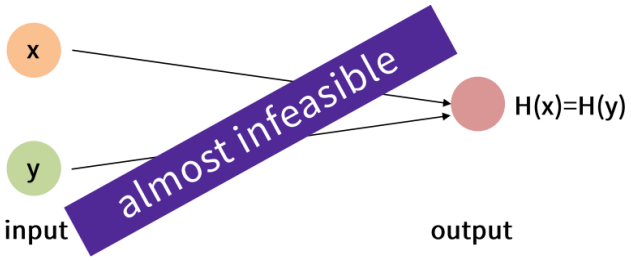


입력값 X를 해시 함수를 이용해 해싱을 하면 출력 값 Y를 얻을 수 있음

그림의 해시 함수 H를 SHA-256 해시 함수라고 가정할 때, 예시에 나와있는 것처럼 소문자 하나를 입력하든, 긴 문자열을 입력하든 32바이트의 동일한 사이즈의 output을 만듦

>> 용량이 큰 원본 데이터의 사이즈를 줄일 수 있음

- **Collision - free**
  - 서로 다른 두 입력에 대해 같은 해시 값을 생성(충돌)하지 않도록 하는 성질

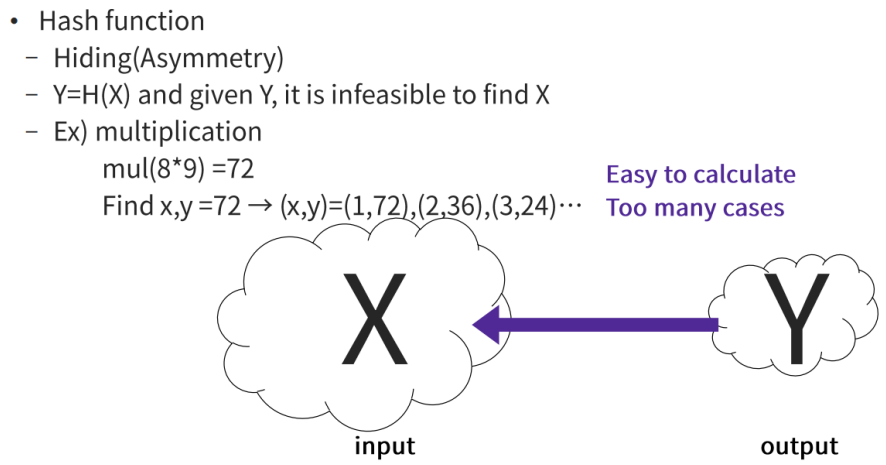


모든 입력 값을 SHA-256 해시 함수를 이용해 해시 하는 경우, 출력 값의 범위는 2^256으로 제한됨
그림의 오른쪽, output은 2^256의 가능성을 가지는 반면, 왼쪽의 input은 매우 다양한 입력 값을 가질 수 있음
왼쪽에 있는 모든 포인트가 화살표로 매핑되어 오른쪽에 있는 지점으로 이동한다고 생각해보면, 왼쪽의 포인트들이 오른쪽의 특정 포인트들에 여러 번 매핑될 수밖에 없음을 알 수 있음
무한대의 범위를 가지는 입력 값을 한정된 2^256 크기의 출력 값에 대응시키려면 충돌이 발생함

이때, 충돌이 적을 수록, 충돌을 찾는 것이 계산적으로 어려울 수록 좋은 해시 함수라고 말함

Why?
충돌이 많을 수록 서로 다른 두 입력이 동일한 해시 값을 가질 확률이 높아지기 때문임 데이터의 무결성과 보안성을 위해서는 충돌이 적어야 함

- **비대칭성**



- 해시 결과인 Y값을 이용해 역으로 X값을 구하는 것은 불가능함

  ex) 8과 9의 곱셈 결과를 계산하는 것은 매우 쉬우나, 72의 결과값을 가지는 x와 y의 조합을 찾는 것은 다양한 경우가 존재하므로 쉽지 않음

- 함수에서 추출된 값(Y)으로 원래 값(X)을 구할 수 없기 때문에 자신이 공개하길 원하지 않는 데이터를 다른 사람들이 못 보게 할 수 있음

  >> 공개키에서 주소를 만들 때 암호화 해시 함수를 이용하는 이유

- **해시 함수의 종류**
  - SHA (Secure Hash Algorithm)
    - SHA-1

      임의의 길이의 입력데이터를 160bit의 출력 데이터로 변환함

      이때 2^64보다 작은 입력 데이터를 다룸

    - SHA-256

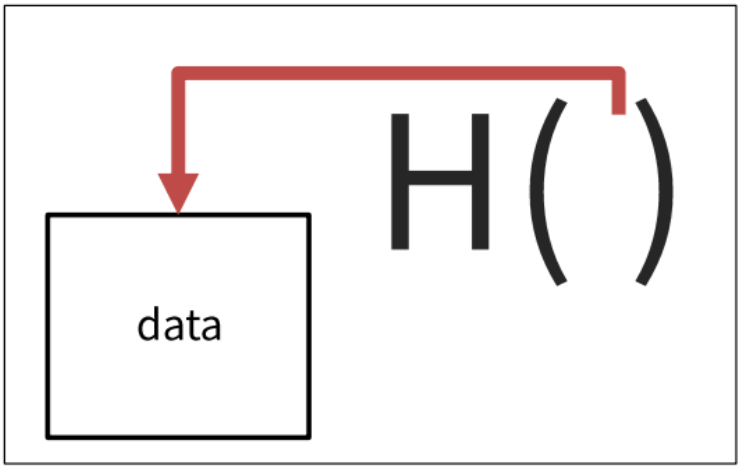
      비트코인 시스템에서 사용하는 해시 함수256bit의 데이터를 생성함

  - Keccak 256

    이더리움에서 채택한 해시 함수

    256bit의 출력 값을 생성하지만, 실제 이더리움에서 사용될 때에는 처음 96bit를 버리고 나머지 160bit를 활용함

2. 해시 포인터



- 정보가 저장된 곳을 가리키는 일종의 포인터이자 마지막으로 기억하는 데이터의 해시값

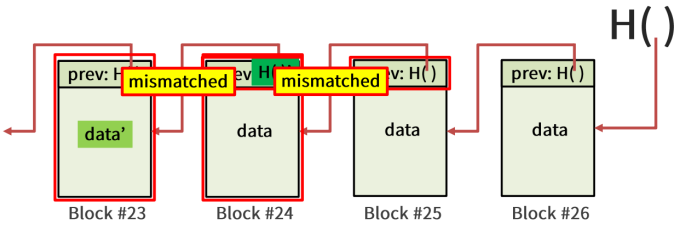
<활용처>

1. 사용자가 원하는 정보를 다시 요청할 수 있음
    - 포인터의 성질과 유사한 특징임

2. 해시가 가리키고 있는 데이터가 변경되었는지 검증하는데 사용할 수 있음
    - 해시 함수의 성질에 따라, 데이터가 변경되면 데이터의 해시 값이 변하기 때문에, 해시 포인터가 가리키는 해시 값과 데이터의 해시 결과가 일치하지 않게 됨

      >> 데이터의 위변조를 검사할 수 있음

- **해시 포인터를 이용한 링크드 리스트 구조**



- 링크드 리스트로 연결된 모든 블록은 리스트 내의 이전 블록을 가리키는 해시 포인터를 가지고 있음

  >> 리스트 내의 데이터 변조 감지 가능

  26번째의 블록이 생성된 와중에 악의적인 노드가 23번 블록의 데이터를 변조하려고 시도한다고 하자.

  23번째 블록 데이터를 변경하면 해시 함수의 collision free 성질에 따라 23번 블록의 해시 값이 변경됨
  따라서 24번 블록 내부의 23번 블록을 가리키는 해시 포인터 값과 실제 해시 값 사이에 불일치가 생김 더 이상 24번 블록 내의 해시 포인터가 실제의 23번 블록을 가리키지 않게 되는 것임

  만약 이러한 문제를 해결하기 위해 24번 블록 내부의 해시 포인터 값을 변경하면 어떻게 될까?

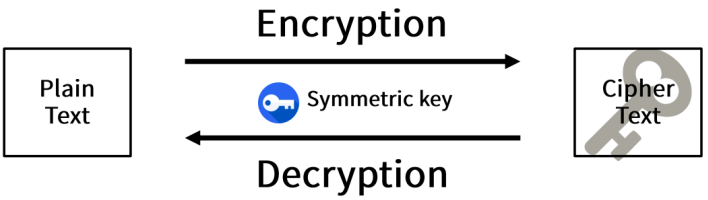
  악의적인 노드가 또다시 24번 블록의 해시 포인터 값을 수정해 실제 해시값과 일치시키려고 수정을 시도한다고 하자.
  하지만 이 경우, 24번 블록의 데이터가 변경됨 따라서 24번 데이터의 해시 값과 25번 내부의 해시 포인터 사이의 불일치를 감지하게 됨

  이러한 메커니즘에 의해 기존의 데이터를 추후에 변경시키는 것이 매우 어려움

3. 기본 암호학

- 암호화 체계는 암호화 키와 복호화 키가 일치하는지 여부에 따라 대칭형 키 암호화 기법과 비대칭형 암호화 기법으로 나뉨

1) 대칭키 알고리즘 (Symmetric Key Algorithm)



- 이 알고리즘에서는 하나의 키만을 이용함

  데이터를 암호화, 복호화할 때 같은 키를 이용해 데이터를 변환함

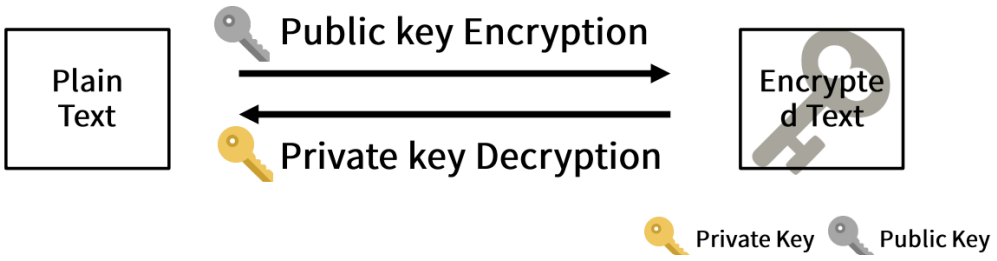
> 그림 왼쪽의 Plain Text는 아무 변조가 가해지지 않은 원본 데이터를 의미함 즉, 누구나 볼 수 있는 문서로, 읽을 수 있는 형태임
> 반면, 오른쪽의 Ciper text는 대칭 키에 의해 암호화된 문서로, 내용을 알아볼 수 없는 문서임

- 대칭 키 알고리즘은 암호화와 복호화를 할 때 동일한 대칭키를 사용하므로 비대칭 키와 비교해 계산 속도가 빠르다는 장점이 있음

- 대칭키 알고리즘의 대표적인 예로, 트리플데스(3DES)나 AES같은 알고리즘들이 있음

2) 비대칭 키 알고리즘 (Asymmetric Key Algorithm)

Public Key Encryption Algorithm이라고도 함



- 공개키(public key)와 개인키(private key)라고 하는 서로 다른 두 개의 키를 이용함

  공개키는 말 그대로 공개적으로 알려져 있는 키로, 누구나 이 공개키를 얻을 수 있음

  반면, 개인키는 개인 혼자만 소유하는 키로, 누구에게도 공유하지 않음

- 이 두 가지 키로 공개키 암호화 또는 전자 서명에 이용할 수 있음
  - **공개키 암호화**

    그림에 나와있는 과정처럼 공개키를 이용해 데이터를 암호화하고, 개인키를 이용해 원본 데이터로 복원함

    개인키를 가진 사용자만 공개키로 암호화된 데이터를 확인할 수 있기 때문에 문서를 특정 사용자를 위해 암호화하는데 사용됨

  - **전자 서명**

    특정 사용자가 문서를 보냈다는 것을 증명, 또는 검증하기 위한 시스템

    전자 서명을 위해 개인키로 암호화된 문서를 제공함
    공개키로 문서를 복호화해서 풀면 전달받은 데이터가 해당 사용자가 보낸 문서임을 증명할 수 있게 됨

- 대표적인 비대칭 알고리즘으로 RSA가 있음

4. 디지털 서명 기술

블록체인 네트워크 상에서 거래를 할 때, 거래를 검증하기 위해 사용하는 기술로, 문서에 하는 인감 날인 또는 사인에 해당하는 기능을 컴퓨터 세계에서 실현함

- **디지털 서명이 가져야 할 보안 요구 사항**

  1. **위조 불가**

     디지털 서명은 본인 이외에 누구도 흉내내서는 안 됨

  2. **인증**

     해당 서명이 본인의 것인지 확인할 수 있어야 함

  3. **재사용 불가**

  4. **변경 불가**

     서명된 문서의 내용을 변경할 수 없음

     만약 내용이 변경되었다면, 서명할 당시의 데이터와 다르다는 것을 알아낼 수 있음

  5. **부인 방지**

     서명자가 자신의 문서임을 입증하기 위해 서명한 뒤 나중에 부인할 수 없음

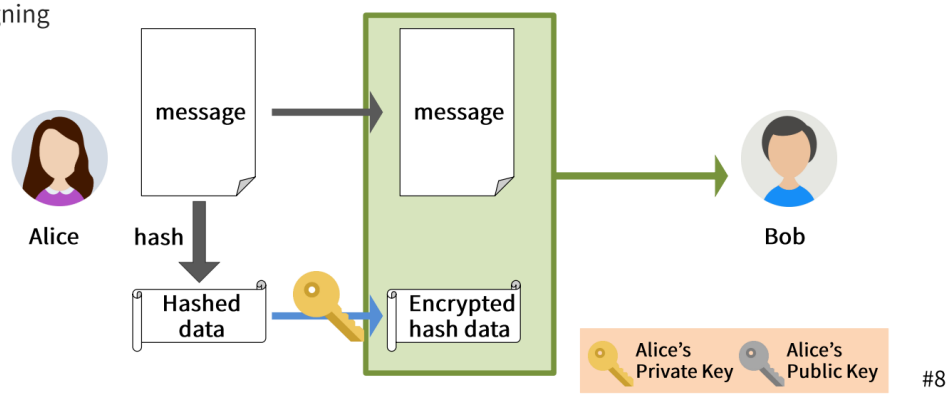
- **디지털 서명 방식을 이용해 문서에 서명하는 과정**

  디지털 서명은 크게 **서명(Signing)**과 **인증(Vertification)** 두 가지 과정으로 나뉨

  - 서명 : 문서가 검증되었음을 알리는 과정
  - 인증 : 독자가 해당 문서에 서명이 되었는지를 확인하는 과정

  1. **서명 (Signing)**

- Process of Digital Signatures
  - Signing



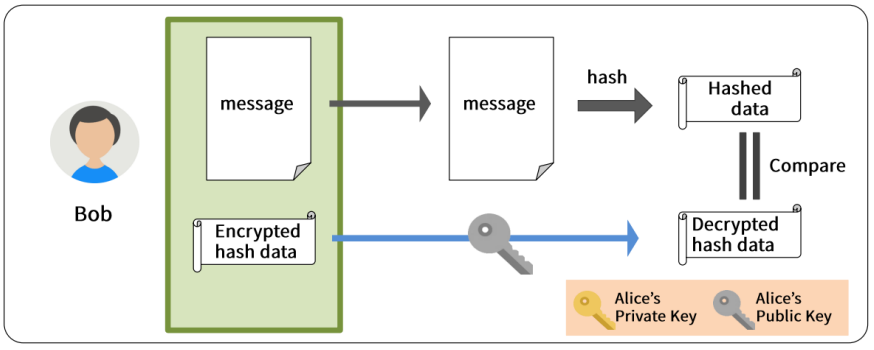
앨리스가 밥에게 디지털 서명 방식을 이용해 메시지를 전송하고자 할 때, 엘리스는 자신의 개인 키를 이용해 메시지에 서명한다.
이때, 원본 데이터 자체를 암호화하는 것이 아니라 해시 함수를 이용해 해시한 메시지에 개인 키로 서명을 함으로써 암호화한다.

앨리스의 개인 키로 서명된 문서는 앨리스의 공개키를 이용해서만 복호화할 수 있다.

앨리스는 암호화한 해시 데이터와 원본 메시지를 함께 밥에게 보낸다.

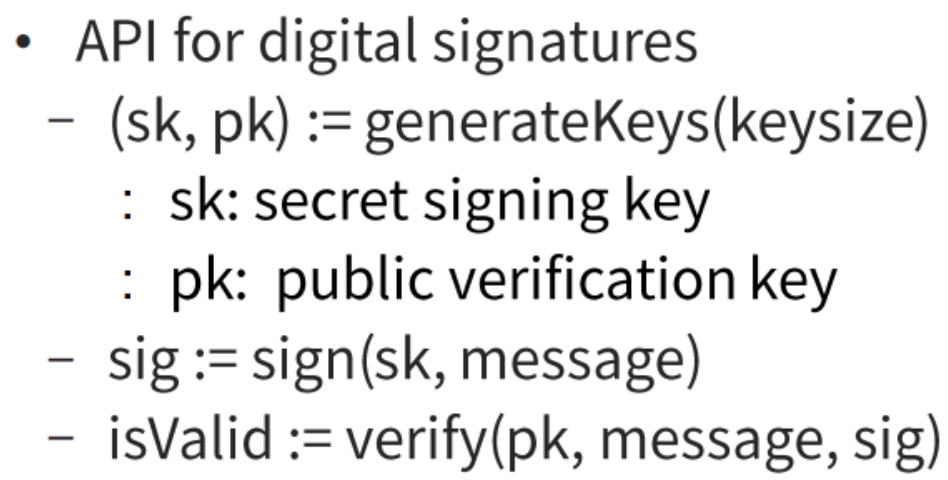
2. 인증 (Vertification)

- Process of Digital Signatures
  - Verification



밥은 앨리스로부터 받은 문서가 진짜 앨리스가 서명한 문서인지 검증해야 한다. 검증을 위해 서명 부분을 따로 떼내어 앨리스의 공개키(Public key)로 복호화(Decryption)한다.
그리고 서명된 문서와 함께 전달받은 원본 데이터를 앨리스가 사용한 것과 동일한 해시 함수를 이용해 해싱한다. 원본 데이터의 해시값과 앨리스의 서명을 앨리스의 공개키를 이용해 복호화한 값이 일치하는지 확인한다.
두 값이 일치하면 해당 문서는 앨리스에 의해 서명된 문서이고, 그렇지 않으면 서명되지 않은 문서이다.

- **전자 서명을 위한 API (Application Programming Interface)**



  - generateKeys
    - 개인 키, 공개키로 이루어진 키 쌍을 만들 수 있는 함수
    - sk (secret signing key)와 pk (public verification key)라고 하는 두 개의 키를 생성함

      sk: 서명에 사용되는 개인 키

      pk: 검증에 사용되는 공개키

      아무나 서명을 검증하는데 공개키를 이용할 수 있음

    - sign
        - 서명을 위한 함수
        - 개인 키 sk와 원본 메시지를 인자로 넣으면 sig라고 하는 서명을 반환함
    - verify
        - 검증을 위한 함수
        - 유효한 서명인지 아닌지 확인하는 sig와 원본 메시지, 그리고 개인 키로 서명된 원본 데이터를 풀기 위한 공개 키인 pk가 인자에 포함

▼ Blocks

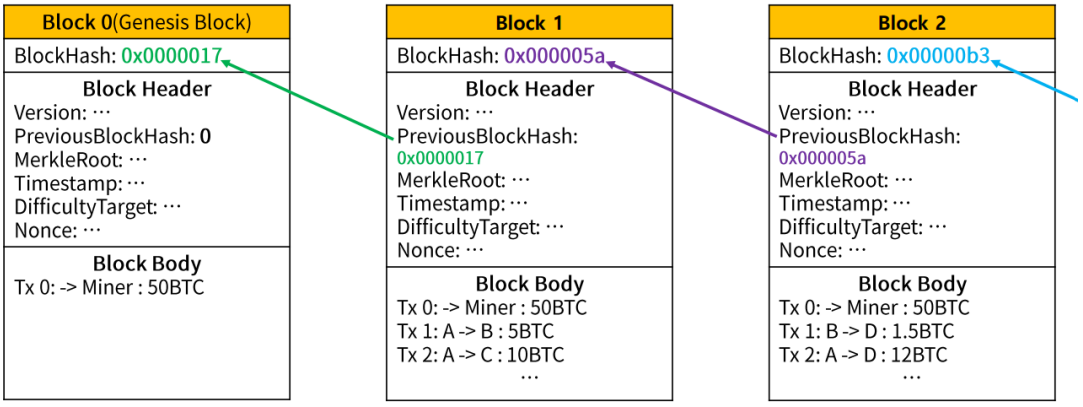
- Table of Contents
    - Origin of Bitcoin
    - Blockchain in Bitcoin
    - Blocks
    - Mercle trees

- **비트코인의 기원**
    - 2008년, 사토시 나카모토에 의해 개발된 최초의 암호화폐 기반의 디지털 지불 시스템
    - 마이닝을 통해 발행됨으로써 기존 통화의 인플레이션 및 디플레이션 문제를 해결하는 2,100만 개 비트코인 한도의 시스템
    - **비트코인에서 처음 사용된 블록체인의 특징**
        1. 중앙 집중식 관리자가 없음

           참여자들끼리 거래의 내용을 공유하고 거래를 인증하고 정보를 저장함
        2. 모든 거래 내역과 역사가 공개됨 = 하나의 거대한 분산 공개 장부

3. 기록된 데이터들은 수정될 수 없음

- **비트코인에서의 블록체인 구조**



- Transactions → Blocks → Blockchain

  - 비트코인에서 블록체인(분산 공개 장부)은 최초의 블록 0부터 제일 최근에 생성된 블록 N까지의 블록이 *Linked list*로 연결된 데이터 구조임

    → 블록이 체인처럼 연결된 데이터를 모든 참여자가 저장하고 있음

    블록은 공개 장부인 블록체인에 거래를 포함시키기 위해 하나에 합쳐 놓은 컨테이너 데이터 구조이다. 블록은 블록체인에서 데이터가 갱신되는 한 주기 또는 최소 단위라고 볼 수 있다. 트랜잭션 단위로 장부가 업데이트되지 않고 블록에 포함된 트랜잭션들이 한 번에 블록체인에 저장되기 때문이다.

    블록 헤더는 여러 개의 블록 메타데이터로 구성되어 있다. 이는 크게 3개의 그룹으로 나눌 수 있다.

    첫째, 이전 블록의 해시값(Previous BlockHash). 이는 현재의 블록이 블록체인에 있는 이전 블록과 연결되어 있음을 나타낸다.

    둘째, 머클 트리 루트(MerkleRoot). 머클 트리 루트는 블록 내의 거래 전부를 효율적으로 요약하는 데 사용되는 데이터 구조이다.

    셋째, 타임 스탬프(Timestamp), 난이도(Difficulty Targe

t), 그리고 난스(Nonce). 이들은 채굴 경쟁과 관련 있는 필드들로, Timestamp는 블록이 생성된 시간을 의미하고, Nonce는 poW(Proof-of-work)에서 사용되는 카운터 역할을 하고, Difficulty Target은 Nonce를 찾는 작업 난이도를 조절하는 필드이다. 이는 Nonce 값을 계산하는데 기준이 되는 특정 숫자를 나타내며, 비트코인 블록체인 전체에 걸쳐 동일하게 적용된다. 간단하게는 블록 해시의 0의 개수로 난이도를 조절할 수 있다. 0의 개수가 적어질수록, nonce값이 맞을 확률이 증가한다.

Proof-of-work(작업증명)는 새로운 블록을 블록체인에 추가하는 작업을 완료했음을 증명한다. 새로운 블록을 블록체인에 추가하려면 그 새로운 블록의 블록 해시값을 계산해야 하고, 그 블록 해시를 계산해 내려면 블록 헤더 정보 중의 하나인 nonce값을 계산해 구해야 한다.
결국 새로운 nonce 값을 구하는 것이 작업증명이라고 말할 수 있다.

- **Genesis block**
  - 비트코인의 블록체인에서 첫 번째 블록으로, 사토시 나카모토가 비트코인을 개발해 처음 블록을 생성시킨 2009년에 만들어짐
  - 블록체인 안에 있는 모든 블록들의 공통 선조가 됨

    블록체인에 연결되어 있는 어떠한 블록이라도 계속 이전 블록을 따라가다 보면 결국에는 이 genesis block에 도달하기 때문

    → 비트코인을 시작하는 어떤 노드들도 적어도 하나의 genesis block을 가지고 시작하고, 이는 변경될 수 없음

    - → You can look into the genesis block using several websites of block explorer
      : https://blockchain.info/block/
      : 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

- **비트코인에서 블록의 구조와 블록이 포함하고 있는 정보**

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields from the block header
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
variable	Transactions	The transactions recorded in this block

The structure of a block

- 블록
  - 메타데이터를 담고 있는 헤더와 그 뒤에 블록 크기를 결정하는 트랜잭션들로 이루어져 있음 → 블록을 크게 두 부분, 블록 헤더와 다수의 거래내역을 포함하는 블록 바디로 나눌 수 있음
  - 블록 헤더의 크기는 80바이트인 반면, 트랜잭션의 평균 크기는 최소 250바이트임
  - 블록에는 평균적으로 500개 이상의 거래가 담겨 있으므로, 모든 거래가 포함된 후 완성된 블록은 블록 헤더의 크기보다 약 1,000배 정도 큼

- 블록 헤더 요소

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the Merkle-Tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block(seconds from Unix Epoch)
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

The structure of a block header

- **블록체인에서 블록을 식별하는 방법**
  - **Block Hash (Block Header Hash)**

    : 블록의 고유한 값으로 이를 이용해 블록을 식별할 수 있음

    : 블록 해시 값은 다음 블록에 previous hash값으로 포함되며 블록들이 체인처럼 연결될 수 있도록 함

    → 중간에 블록이 수정되면 해시 값이 바뀌기 때문에 이후로 연결된 모든 블록의 previous hash, block hash가 달라지게 됨 이를 블록체인의 비가역

성 (Irreversibility)라고 함

: 블록 헤더만 가지고 해시 값을 만들면 블록 바디에 있는 거래가 변경되면 어떻게 식별할 수 있을까?

→ 블록 헤더의 Merkle Root로 해결할 수 있음 Merkle root는 블록 바디에 있는 모든 거래 내용의 축약본이므로, 블록 안의 거래가 수정되면 이 merkle root도 변하기 때문에 전체 블록 헤더의 해시 값도 변함

: block header에 SHA-256 해시 함수를 두 번 적용시켜서 나온 결과를 사용하며 총 32바이트 길이를 가짐

- **Block height**

  : 블록의 높이는 인덱스로 활용되며 블록체인에서 블록의 위치를 나타내는 지표로 사용됨

  : 블록의 높이는 최초 0부터 시작하며 블록이 생성될 때마다 1씩 증가함

  : 블록을 식별하는 유일값은 아님

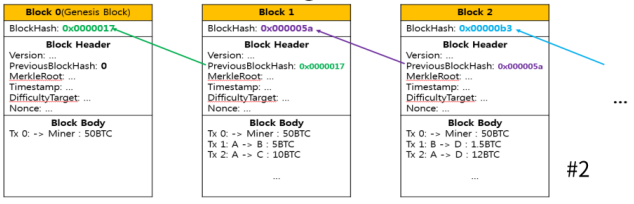
  → 블록체인의 분기(fork)가 발생했을 때, 동일한 두 개의 블록이 같은 블록의 높이를 가질 수 있으므로 블록의 높이로는 어떤 블록인지 식별X

  다만, 메인 블록체인에 연결되는 것은 두 개의 동일한 높이의 블록 중 하나만 포함됨

  → 즉, 여러 개의 블록이 블록체인에서 해당 블록의 높이를 두고 경쟁하는 구조임

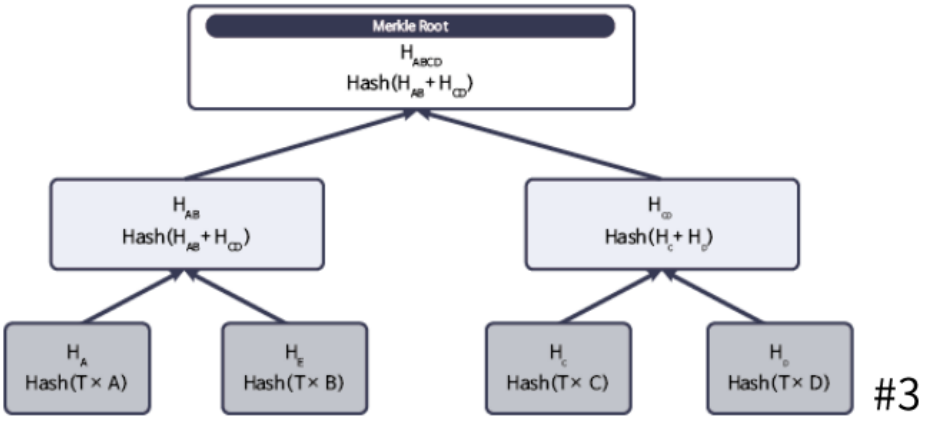
- **블록이 체인에 연결되는 과정**

  - Steps
    : A node receives an incoming block from the network
    : It validates this block
    : If the validation is passed, it links this block to the existing blockchain



현재 블록 1이 생성되어 있는 상태에서 블록 2가 만들어져 네트워크를 통해 들어왔다고 하자. 이때 노드는 블록2의 헤더를 보고 이전 블록의 해시를 찾는다. 더 이전 블록 해시는 이전 블록인 블록 1의 블록헤더 해시와 동일해야 한다. 이렇게 자신이 child(블록2)가 되고, 이것과 연결된 parent 블록을 찾으면 생성된 블록을 이용해 블록체인을 연장시킨다.

- **Merkle Tree**



  - 머클 트리는 이진 해시 트리라고도 하는데, 규모가 큰 데이터 집합의 완전성을 효율적으로 요약하고 검증하는데 사용되는 데이터 구조임
  - 비트코인에서 머클 트리를 구성할 때는 꼭 리프가 짝수여야 함

    → 만약 거래가 홀수로 있을 때는 마지막 거래를 복사해서 옆 리프에 두어야 함

    ex) H-C를 복사해서 H-D에 위치시켜 두어 짝수 개의 리프를 만들어야 함

    → 이를 균형 트리(balanced tree)라고 함

  - 머클 루트라고 불리는 해시 하나가 남을 때까지 노드 쌍을 반복해서 해싱해서 머클 트리를 만듦

    H_A와 H_B를 더해서 해싱한 것이 H_AB가 됨
    그리고 H_C와 H_D를 연결해서 (32+32=64) 해싱한 것은 H_CD가 됨

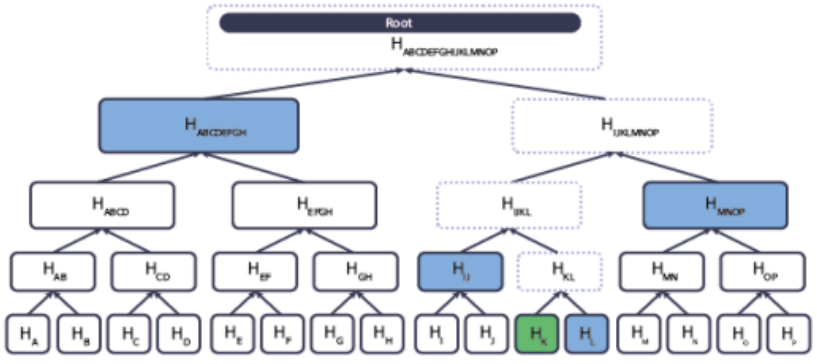
다시 이 두 개, (H_AB와 H_CD)를 연결해서 해싱하면 머클 루트(H_ABCD)가 탄생하고, 그림처럼 머클 트리가 만들어짐

암호 해시 알고리즘으로는 SHA-256을 사용함
H_A = SHA 256(SHA 256(Transaction A))
이 수식은 한 개의 거래에 대해서 두 번 SHA-256을 적용하여 H_A를 만든다는 것을 뜻함

나머지도 마찬가지임 이 과정을 계속하다 보면 결국 하나의 해시값만 남게 되고 (H_ABCD) 이를 머클 루트라고 함

- **Merkle Tree 특징**
  - 블록 내에 있는 모든 거래를 요약하기 위해 비트코인에서 사용되며, 거래의 전체 집합에 대한 모든 디지털 지문을 만들어 냄
  - 거래들이 블록 내부에 포함되는지 여부를 검증하는데 효율적인 프로세를 제공함

    → 특정 거래가 블록 내에 포함되어 있음을 입증하기 위해서는 노드가 log 2(N) 개의 32바이트 해시를 생성해서 특정 거래와 트리의 루트를 연결하는 머클 경로를 구성하면 됨



머클 경로를 이용해서 H_K가 블록 내에 포함되어 있는지 확인하는 메커니즘

차례대로 H_L, H_IJ, H_MNOP, H_ABCDEFGH는 특정 거래와 머클 루트를 연결하는 머클 경로를 나타내고 이를 이용하면 H_K가 블록 내에 포함되어 있음을 알 수 있음

- **머클 트리의 효율성**

  거래의 건수가 증가할 때 log 2(N)의 값이 노드의 증가보다 훨씬 완만하게 증가함

Number of Transactions	Approx. Size of Block	Path Size (Hashes)	Path Size (Bytes)
16 transactions	4 KB	4 hashes	128 bytes
512 transactions	128 KB	9 hashes	288 bytes
2048 transactions	512 KB	11 hashes	352 bytes
65,535 transactions	16 MB	16 hashes	512 bytes

Merkle Tree's Efficiency

> 블록 안에 포함되는 거래의 개수는 16건 4kb 크기에서 65,535건 16mb 크기로 급속하게 증가했지만, 머클 경로는 128바이트에서 512바이트로 아주 완만하게 증가했음을 확인할 수 있음

▼ Transactions part1

- **Transaction in Bitcoin**
  - 거래는 비트코인 시스템 내에 있는 참가자들간 입력 값이라고 불리는 자금원에서부터 출력 값으로 불리는 목적지까지 가치의 전송을 인코딩하는 데이터 구조임
  - 각 거래는 전 세계적인 장부에 들어있는 공개된 항목임

- **Transaction Lifestyle**

1. 거래는 비트코인 시스템에 참여하는 모든 노드가 생성할 수 있다.

> 비트코인 거래의 경우, 이전에 발생한 거래의 출력을 자금원으로 참조한다.

거래가 생성되었으면 자금원의 소유주가 자금원을 해당 거래의 입력 값으로 사용하기 위해 해당 거래에 서명해야 한다.

2. 서명이 완료된 유효한 거래는 블록체인에 포함되기 위해 채굴자들에게 도달할 때까지 비트코인 네트워크 상에 전파되어야 한다.

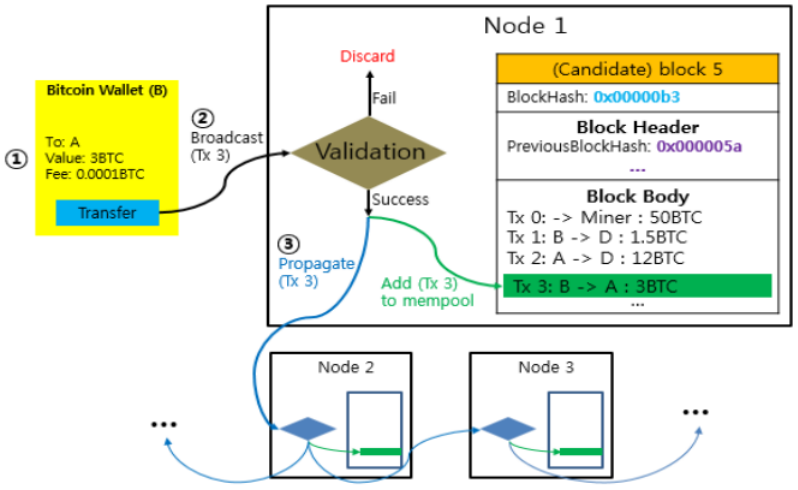
   우선 비트코인 네트워크에서 생성한 거래를 자신과 연결된, 거래를 전파하게 될 첫 번째 노드에게 전달한다.
   거래는 서명을 받았고 개인 정보를 포함하고 있지 않기 때문에 모든 네트워크를 통해 공개적으로 전송될 수 있다.

3. 비트코인 거래가 비트코인 네트워크에 연결된 한 노드로 전송되면 해당 노드에 의해 거래가 유효화된다.

   거래가 유효화되고 나면 해당 노드와 연결된 다른 노드로 이 거래를 전파하고 동시에 성공 메시지가 생성자에게 전달된다.
   만약 거래가 유효화되지 못했다면 해당 노드는 승인을 거부하고 동시에 거절 메시지가 생성자에게 전달된다.

4. 연결된 노드 전체가 해당 거래를 전송받게 된다.

   네트워크 상에 있는 노드에 새롭게 추가된 유효화 거래는 주변에 있는 3 ~ 4개의 노드로 전송되고, 전송된 각각의 거래는 또다시 3~4개의 주변 노드로 전송되며 이 과정이 반복된다.
   이런 방식을 이용해 유효 거래는 몇 초 만에 기하급수적으로 전파의 물결을 일으켜 결국 연결된 노드 전체가 해당 거래를 전송받는다.



거래 생명 주기 메커니즘을 그림으로 나타낸 예

우선 B라는 사용자가 Wallet 앱을 이용해 A라는 사용자에게 세 개의 비트코인(3BTC)을 전송한다는 거래를 생성하여 연결된 노드로 전달한다.
연결되어 있는 노드1이 이 거래를 받게 되고, 수신한 거래가 유효한 거래인지를 확인하는 유효성 검증을 수행한다.
거래가 검증을 성공적으로 통과했다면, 노드1은 검증된 거래를 메모리 풀(Memory pool)에 추가한다. 메모리 풀은 마이닝 하기 전에 블록에 포함될 수 있는 거래를 모아 두는 곳이다.
이렇게 검증된 거래를 추가함과 동시에, b 자신과 연결된 노드들에게 해당 거래를 전파한다.
거래를 수신한 노드2와 노드3은 노드1에서와 동일한 프로세스를 진행하게 되고, 결국 이 거래는 블록체인 네트워크 전체에 전파된다.

- **거래 구조**
  - 비트코인에서 거래의 입력 값과 출력 값은 계좌(계정)와는 관계가 없음
  - 입력 값과 출력 값은 소유주나 비밀키를 알고 있는 사람만이 풀 수 있는 비밀키로 잠겨있는 비트코인 덩어리로 간주해야 함

Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1-9 bytes (VarInt)	Input Counter	How many inputs are included
Variable	Inputs	One or more Transactions Inputs
1-9 bytes (VarInt)	Output Counter	How many outputs are included
Variable	Outputs	One or more Transactions Outputs
4 bytes	Locktime	A unix timestamp or block number

The structure of a transaction

- 거래는 여러 개의 필드를 포함하고, 하나의 거래에는 다수의 입력 값과 출력 값이 존재할 수 있음

  → 거래 유형이 여러 개 생김

- Input Counter

  : 얼마나 많은 입력 값이 거래에 포함되었는지 알려줌

- Inputs

  : counter에 해당하는 만큼의 입력값에 대한 정보를 다루는 필드

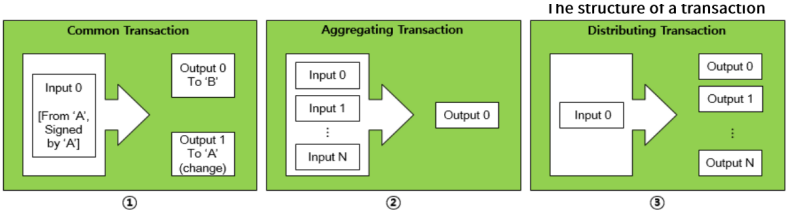
  → Input Counter과 Inputs 필드는 크기가 가변적임

- Locktime

  : 거래한 것이 블록체인에 추가되는 가장 빠른 시간을 의미하며, 바로 실행이 가능한 상태임을 알려 주기 위해 Locktime은 대부분 0으로 설정됨

  → 만약 해당 필드가 0이 아니거나 5억 미만이라면 블록의 높이로 해석

- **거래 유형**



- (1) Common Transactions

  ```
  가장 일반적인 거래

  하나의 A로부터 입력 값 1이 출력 값 1로 B에게, 그리고 출력 값 2로 다시 자신(A)에게 전송됨

  블록체인은 Input과 Output으로만 이루어졌고, Input-Output으로 나머지 코인이 발생하면 자신에게 자동으로 돌아가지 않도록 설계됨

  따라서 코인을 보내고 싶은 사람에게 일정 금액을 보내고 잔돈이 생겼다면, 꼭 나머지 금액을 출력값 2로 하여 자신의 주소로 보내야 함
  ```

- (2) Aggregating Transaction

  : 입력 값이 여러 개가 올 수 있음을 보여주는 거래

  : 블록체인에서는 여러 개의 입력 값을 합쳐서 코인들을 하나의 출력 값 1로 보낼 수 있음

- (3) Distributing transaction

  : 출력 값이 여러 개가 올 수 있음을 보여주는 거래

  : 블록체인에서 하나의 입력 값 1을 쪼개서 여러 개의 출력 값으로 분배할 수 있는 거래를 말함

- **비트코인의 잔액_UTXO**
  - 비트코인에는 계정이나 주소에 저장된 잔액이 없음

    오직 블록체인에 흩어져서 산재해있는 소비되지 않은 거래 출력(UTXO, unspent transaction output)만 있고, 이것이 특정한 소유자들에게 암호로 잠겨 있을 뿐임

사용자의 비트코인 잔액의 개념은 지갑 애플리케이션에서 파생된 것이므로, 비트코인에서는 사용자의 계정별로 잔액이 저장되지 않음
이 지갑 소프트웨어는 블록체인을 처음부터 끝까지 스캐닝하고, 해당하는 사용자에 속한 모든 이용 가능한 UTXO를 다 더해서 사용자의 잔액을 계산함

-> 사용자의 비트코인은 결국 자신에게 잠겨있는 UTXO들임

- **UTXO**

  : 블록체인상에 저장됨으로써 전체 비트코인 네트워크에 의해 통화 단위로 인정받은 불가분의 비트코인 덩어리

  → 쪼개져서 사용할 수 없으므로, UTXO가 거래의 입력으로 사용될 때는 부분만이 아니라 전부를 사용해야 함

  → 보낼 금액이 입력 값의 UTXO보다 적다면 자신의 주소로 잔액을 보내야 함

  → 잔액이 빈번하게 발생함!

  > 실제 비트코인 지갑에서는 보낼 금액에 맞도록 UTXO를 선택하는 몇 가지 알고리즘을 사용하고 있음

거래 입력 값 / 출력 값과 UTXO의 관계

- **거래 입력 값 (Transaction input)**

  : 하나의 거래에 의해서 소비되는 UTXO를 의미함

  : 거래에서 UTXO가 소비될 때, 현재 소유자의 서명을 가지고 잠금을 해제해야 소비할 수 있음

- **거래 출력 값**

  : 하나의 거래에 의해서 출력 값으로 새롭게 생성되는 UTXO를 의미함

  > Data Output을 제외하고, 거의 모든 출력값들은 소비가 가능한 UTXO를 생성함

  : 이 때 해당 소유자만 소비할 수 있도록 새로운 소유자의 비트코인 주소로 UTXO를 잠가 둠

→ UTXO를 생성하고 소비하는 거래는 체인에서 비트코인의 덩어리들이 이전 소유자에서 다음 소유자로 넘겨지는 형태로 이루어짐

→ 일정량의 비트코인을 보낸다는 것은 받는 사람의 주소로 UTXO를 등록하여 소비 가능하도록 한다는 것임

- **거래 출력 값의 구조**

  : 거래 출력 값은 두 가지 부분으로 구성되어 있음

Size	Field	Description
8 bytes	Amount	Bitcoin Value in Satoshi (10^{-8} bitcoin)
1-9 bytes (VarInt)	Locking-Script Size	Locking-Script length in bytes, to follow
variable	Locking Script	A script defining the conditions needed to spend the output

The structure of a transaction output

  - 비트코인의 양
  - 잠금 스크립트 (Locking script)

    : 출력 값을 소비할 때 충족시켜야 하는 조건을 특정하여 비트코인을 잠금

- **거래 입력 값의 구조**

Size	Field	Description
32 bytes	Transaction hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent, first one is 0
1-9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking-script
4 bytes	Sequence Number	Currently-disabled Tx-replacement feature, set to 0XFFFFFFFF

The structure of a transaction input

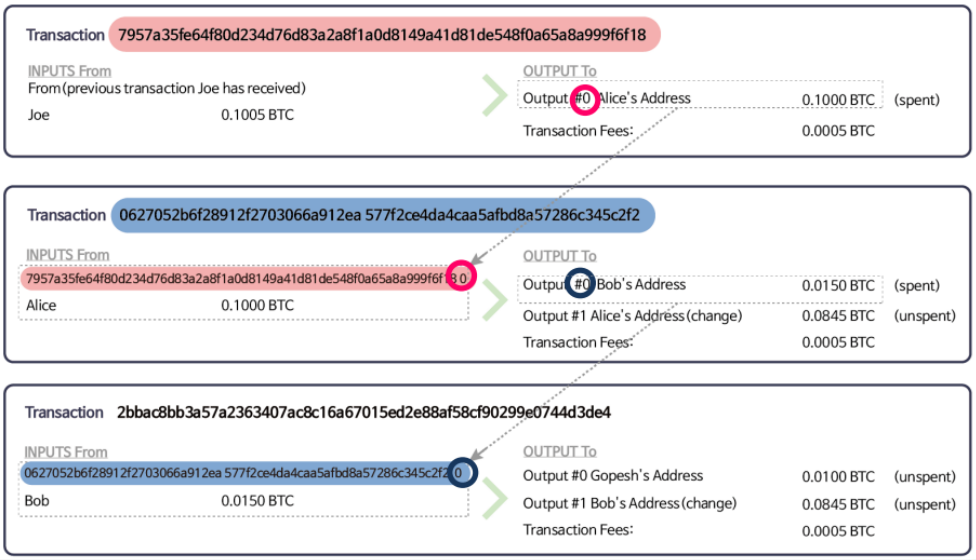
: **거래 입력 값**은 블록체인에 저장되어 있는 하나의 특정한 UTXO를 가리키는데, 이때 거래 해시 값과 시퀀스 넘버가 참조됨

: 거래 해시 값은 하나의 특정한 거래를 식별하고, 그 거래의 출력 값으로 생긴 UTXO들 중에서 시퀀스 넘버를 인덱스로 하여 특정한 UTXO를 참조할 수 있음

: 누군가 거래를 생성할 때 입력 값으로 UTXO를 소비하려고 한다면, 입력 값에 (해당 UTXO에 설정된 소비 조건을 만족하는) 잠금 해제 스크립트를 포함해야 함

- 잠금 해제 스크립트 : 잠금 스크립트에 있는 비트코인 주소의 소유권을 증명하는 전자서명

- 거래 사슬 구조
  - 거래 출력 값은 다음 거래에서 입력 값으로 쓰임



각 거래는 하나 이상의 입력 값(비트코인 계좌에서 빠져나가는 쪽)과 출력 값(비트코인 계좌로 들어오는 쪽)으로 구성된다. 거래를 통해 입력 값에서 거래 출력 값으로 가치가 이동한다. 이때, 거래 입력 값은 비트코인의 가치가 발생하는 지점으로, 주로 이전 거래의 출력 값이고, 거래 출력 값은 키를 이용해서 새로운 소유주에게 비트코인의 가치를 넘겨주는 역할을 한다. 한 거래의 출력 값은 다음 거래의 입력 값이기 때문에, 거래들도 체인처럼 연결되어 있다고 볼 수 있다.

위 그림을 보면, 처음 거래의 거래 해시와 출력 값의 번호(INDEX)가 다음 거래의 입력 값으로 나오는 것을 확인할 수 있다. 한 거래에서 나온 출력 값은 새로운 거래에서 입력 값으로 사용될 수 있기 때문에, 이와 같이 소유권 사슬을 생성함으로써 가치가 비트코인 주소들 간에 이동하게 된다.

비트코인 용어로 '소비(SPENT)'라 함은, 이전 거래에서 송금되었던 돈이 비트코인 주소로 확인된 새로운 소유주에게로 전송되는 거래에 서명을 함으로써 이루어지는 작업을 말한다. 소비되지 않은 비트코인은 거래에서 새로 생긴 출력 값이고, 이를 비트코인 안에서는 UTXO(Unspent Transaction Output)이라고 부른다.

```
  거래에서 입력 값과 출력 값의 한계는 꼭 동일하지 않아도 된다.
 보통 출력 값의 총합이 입력 값의 총합보다 약간 작은데, 그 차이는 거래수수료(transaction fee) 때문이다.
```

▼ Transactions part2

- 수수료
  - Serve as an incentive

    → 해당 거래가 포함된 블록을 채굴한 마이너가 가져감

    → 거래 채굴 장려금으로 사용됨
  - Serve as a disincentive against spam
    - 블록체인 네트워크에 악영향을 미치는 노드들의 의욕을 꺾음

      ← 스팸이나 DDOS를 하기 위해서는 거래를 생성해야 하는데 이때 수수료가 발생하면 금점적인 문제 때문에 공격을 못하기 때문
    - Fee =