새로운 블록체인 채굴 기법
마이크로컨트롤러의 부채널 정보 해쉬 사용

https://youtu.be/m2rq_vjq9qI

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

자격 증명

- 작업 증명(PoW: Proof of Work)

  1993년 서비스 거부 공격을 방지하고 네트워크 상에서 스팸과 같은 어뷰징을 막기 위해 시작
  이를 위해 서비스 사용자들에게 일정량의 작업을 요구
  ex)컴퓨터의 프로세싱 타임

- 비트코인의 PoW

  2009년, 비트코인은 PoW를 합의 알고리즘으로 사용
  트랜잭션을 검증하고, 블록체인에 새로운 블록들을 전송하는 혁신적으로 방법을 도입
  이후 많은 암호 화폐에서 널리 채택되는 합의 알고리즘

  ex) 비트코인, 이더리움, 라이트코인, 비트코인캐시, 비트코인골드, 모네로

합의 알고리즘

- 블록체인의 합의 알고리즘
  블록체인 네트워크에 합의를 달성하는 매커니즘
  퍼블릭(탈중앙화된) 블록체인은 분산화된 시스템으로 구성
  분산화된 노드는 트랜잭션의 유효성에 합의가 필요

➢정직하게 행동해야 하며 이를 보장할 방법이 필요(비잔티움 장군 문제)

비잔티움 장군 문제

- 비잔티움 장군 문제는 배신자의 존재에도 불구하고
  옳은 지휘관들이 동일한 공격 계획을 세우기 위해서 얼마만큼 존재해야 하며,
  어떤 규칙을 따라 교신해야 하는지에 대한 문제

- 상황

  적군의 도시를 공격하려는 비잔티움 제국군의 **여러 부대**가 **지리적으로 떨어진 상태**
  각 부대의 지휘관들이 (중간에 잡힐지도 모르는) **전령을 통해 교신,**
  공격 계획을 함께 세우는 상황을 가정
  이 부대의 지휘관 중 일부에는 **배신자가 섞여 있을 수 있음**

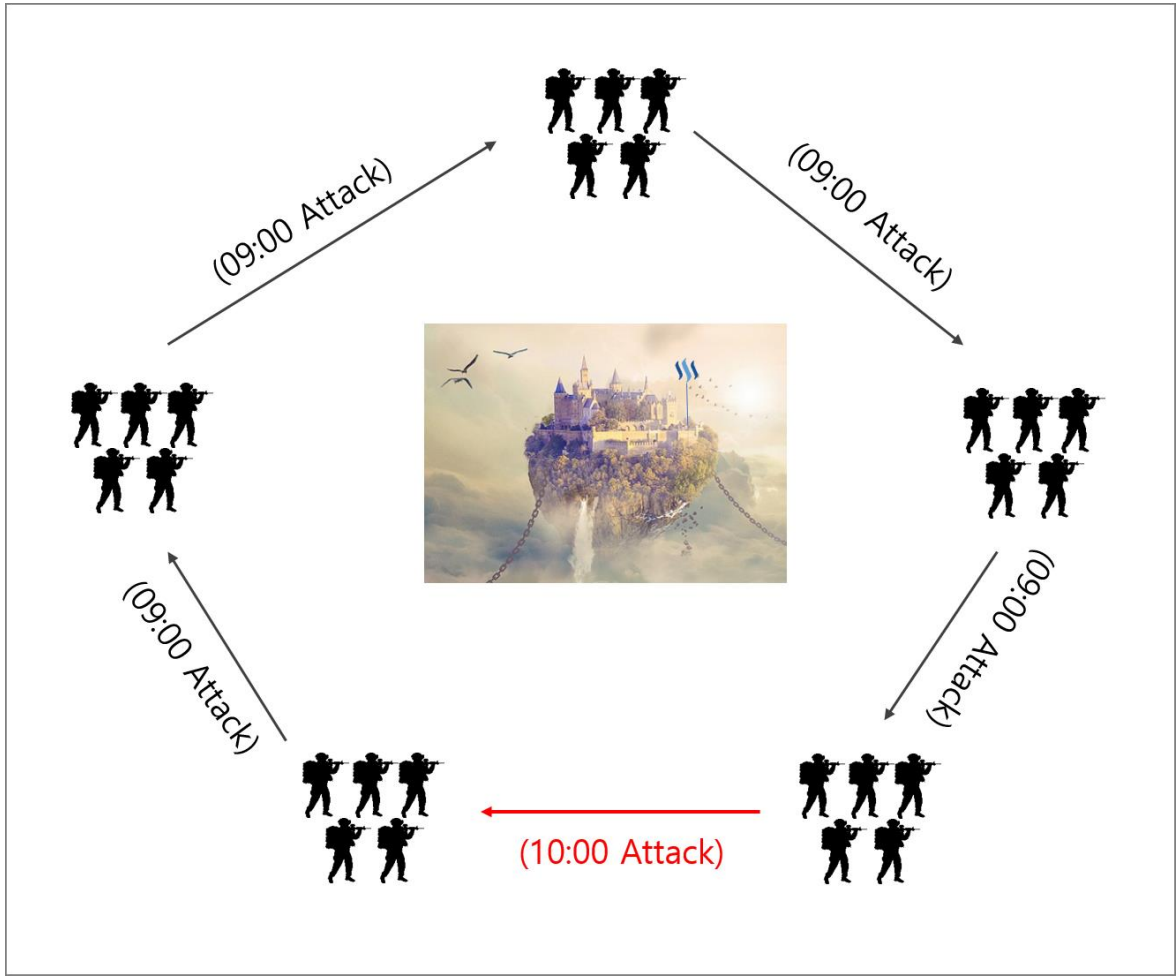
비잔티움 장군 문제

- 지휘관은 총 5명
- 각 지휘관이 보유한 병력은 1,000명
- 성에는 3,000명의 병사가 성을 지킴
- 함락을 위해서는 최소 4명의 지휘관이 동시에 공격 명령

➢배신자를 찾기

➢배신자가 없기를 기도

- 불가능을 증명한 논문



비잔티움 장군 문제

- 비트코인에서는 **타임 스탬프(Timestamp)**와
  서명(Sign, 블록체인에서는 Key)을 통해 이 문제를 해결

- ‘장군은 메시지를 보내기 위해 10분의 시간을 가지며,
  메시지는 **모든 장군의 메시지**와 메시지를 보내기 위해
  10분을 들였다는 증거를 포함한다.’ 라는 규칙이 추가

- 일정 시간동안 작업을 하여, 다음 사람에게 넘기고 다음 사람도 일정 시간 동안 작업을 하므로 조작 불가능

- 중간의 비잔틴 장군이 존재하더라도 다른 장군들은 이 장군이 거짓임을 밝혀낼 수 있다.

비트코인 합의 알고리즘

1. 새로운 **블록**을 만들 때 새로 만들 블록의 고유번호를 찾는
   일종의 **퍼즐 게임**(정답을 검증하는 것은 간단)을 풀도록 함
2. 퍼즐을 푸는 과정이 곧 작업증명
3. 수학 퍼즐이 풀리면 블록이 생성
   = 채굴(암호화폐의 거래내역을 기록한 블록을 생성하고 그 대가로 암호화폐를 얻는 행위)
4. 작업증명을 수행해 퍼즐이 풀렸다면 결국 블록은 부정이 없도록 각종 규정을 지킨 정상 거래를 모아 정상적인 방법으로 만든 원장이라는 의미
5. 작업증명을 거친 블록을 받아들이는 방식의 합의 구조
6. 작업증명을 거친 블록을 실시간 공유함으로써 모두가 항상 같은 원장 상태에 도달

블록

- “A가 B에게 100원을 송금한다.”와 같은 것이 하나의 거래이며,
  하나의 블록에는 여러 개의 거래가 포함

- 비트코인의 블록 하나에는 평균 약 1,800개의 거래 정보가 포함될 수 있으며,
  블록 하나의 물리적인 크기는 평균 0.98Mbyte

- 블록은 블록 헤더와 거래 정보, 기타 정보로 구성

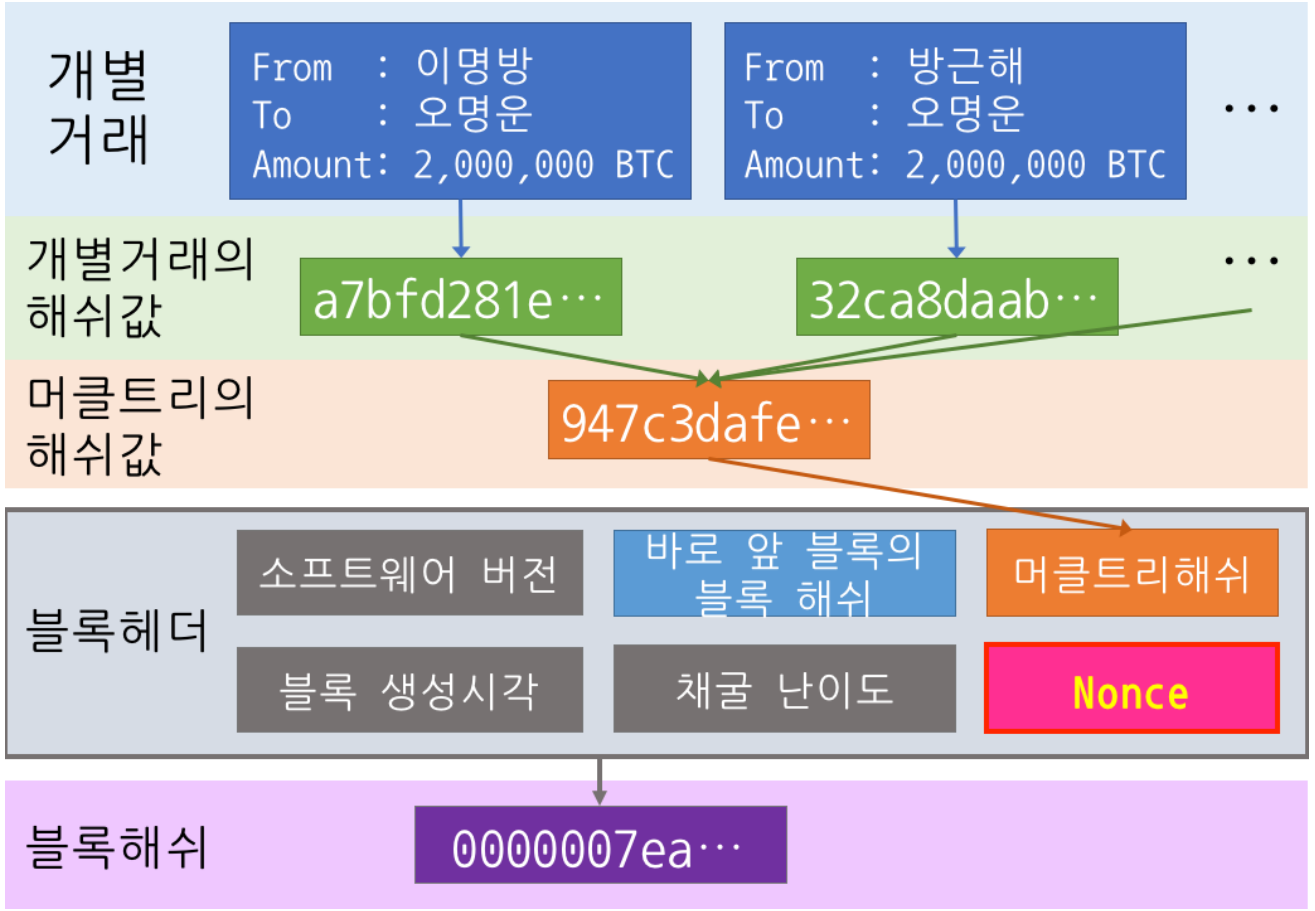
블록 헤더

블록 헤더는 다음의 6가지 정보로 구성

- version : 소프트웨어/프로토콜 버전
- previousblockhash : 블록 체인에서 바로 앞에 위치하는 블록의 블록 해쉬
- merklehash : 개별 거래 정보의 거래 해쉬를 2진 트리 형태로 구성할 때, 트리 루트에 위치하는 해쉬값
- time : 블록이 생성된 시간
- bits : 난이도 조절용 수치
- nonce : 최초 0에서 시작하여 조건을 만족하는 해쉬값을 찾아낼때까지의 1씩 증가하는 계산 회수

블록 해쉬

- 블록 헤더의 6가지 정보를 입력값으로 사용
- SHA256 해쉬 함수를 적용해서 계산되는 값



채굴

- 개별 거래 정보는 머클 트리의 해쉬값인 merklehash 값으로 집약

- 블록 헤더의 6가지 정보 중
  version, previousblockhash, merklehash, time, bits
  5가지는 블록 해쉬를 만드는 시점에서 이미 확정되어 변하지 않는 값

- nonce는 확정되어 있지 않고 새로 구해야 하는 값
  이 nonce 값을 구해서 최종적으로 블록 해쉬 값을 구하고,
  이 블록 해쉬값을 식별자로 가지는 유효한 블록을 만들어내는 것이
  작업 증명(Proof of Work), 채굴이다.

채굴

1. Nonce
   nonce을 입력값 중의 하나로 해서 계산되는 블록 해쉬값이 **특정 숫자보다 작아지게** 하는 값

2. 해쉬 함수의 특성상, 어떤 해쉬값(A)을 결과로 나오게 하는 입력값을 찾으려면,
   A에서 역산을 하는 방식은 불가능

3. 어떤 블록 해쉬값이 어떤 특정 숫자보다 작아지게 하려면,
   블록 해쉬의 입력값을 계속 바꿔가면서 해쉬값이 특정 숫자보다 작은지 비교하는 작업을 반복

4. nonce값을 계속 바꿔가면서 계산한 해쉬값이 어떤 특정 숫자보다 작다면,
   그 때의 nonce값이 새로 만들어지는 블록의 nonce값으로 확정

5. 특정 숫자 보다 작게 나온 그 해쉬값이 새로 생성되는 블록의 블록 해쉬값으로 확정

Contents

CryptoCraft LAB

Asic 등장

- Asic이란 일반적인 목적이 아니라 특정한 목적으로 사용하기 위해
  필요한 스펙, 기능을 최적화시키고, 커스터마이징 하여 생산한 주문형 반도체 칩

- Asic을 암호화폐와 관련하여 이용한다면,
  특정 계열의 해시 함수의 연산에 특화되어
  같거나 더 적은 전력을 소모
  훨씬 더 많은 해시파워를 낼 수 있음
  - 빠른 연산이 가능하여 블록을 생성할 확률이 높아져 채산성이 높아져 이익이 더 많이 생기게 됨

Asic 등장

- 개인 채굴업자들의 경우
  가동할 수 있고 관리할 수 있는 Asic 채굴기의 수에 어느 정도 한계
  전기 요금에 따라 채산성이 달라짐

➢ASIC 채굴기를 이용하여 채굴한다고 하더라도 전문 채굴업자들보다 불리

➢채굴이 일부 제한된 소수나 회사에 의해 독점 가능

Asic 등장

- 블록 속도에 영향을 주지 않음
  기기 소유자에게 더 많은 토큰을 가져다 줄 뿐, 실질적으로 네트워크를 돕는 역할은 하지 않음

➢ASIC 채굴기는 진입 장벽을 높여, 블록체인 기술의 보급 및 발전에 도움되지 않음

MONERO ASIC 저항

- MONERO에서는 CryptoNight는 범용 CPU에 최적화 된 작업 증명 (PoW) 알고리즘 사용
  - ➢ CryptoNight ASIC 개발되어 실패
  - ➢ 6개월 마다 한 번씩 하드포크

- 2019년도 11월부터 MONERO 블록체인에서 사용중인
  RandomX는 범용 CPU에 최적화 된 작업 증명 (PoW) 알고리즘

MONERO RandumX

- 특수한 하드웨어의 효율성 이점을 최소화
  여러 메모리 하드 기술
  임의 코드 실행 사용

- 입력으로 키 블록 사용
  키가 2048 블록마다 (~ 2.8 일) 변경, 키 블록과 키 변경 사이에 64 블록 (~ 2 시간)이 지연
  하드포크가 없이도 키를 주기적으로 변경함으로써 ASIC 채굴에 대한 대응

- 알고리즘
  1. 임의의 프로그램을 생성
  2. CPU의 기본 기계 코드로 변환
  3. 프로그램을 실행
  4. 프로그램의 출력을 암호로 안전한 값으로 변환

MONERO RandumX

- 임의 코드 VM에서 실행 함으로 CPU에 최적화

➢ 상용 컴퓨터에 최적화하여 진입 장벽을 낮추어, 블록체인 기술의 보급 및 발전에 도움

- 에너지 소모 문제?

➢ 저전력의 마이크로 컨트롤러 사용을 유도

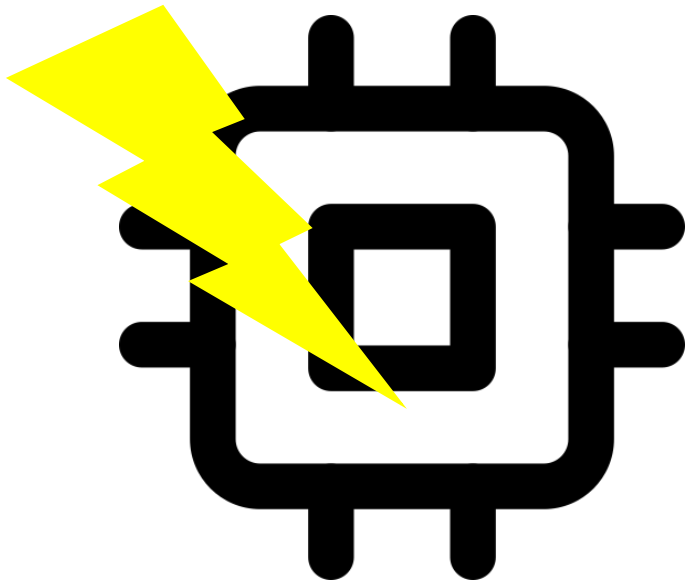
Contents

CryptoCraft LAB

부채널 분석

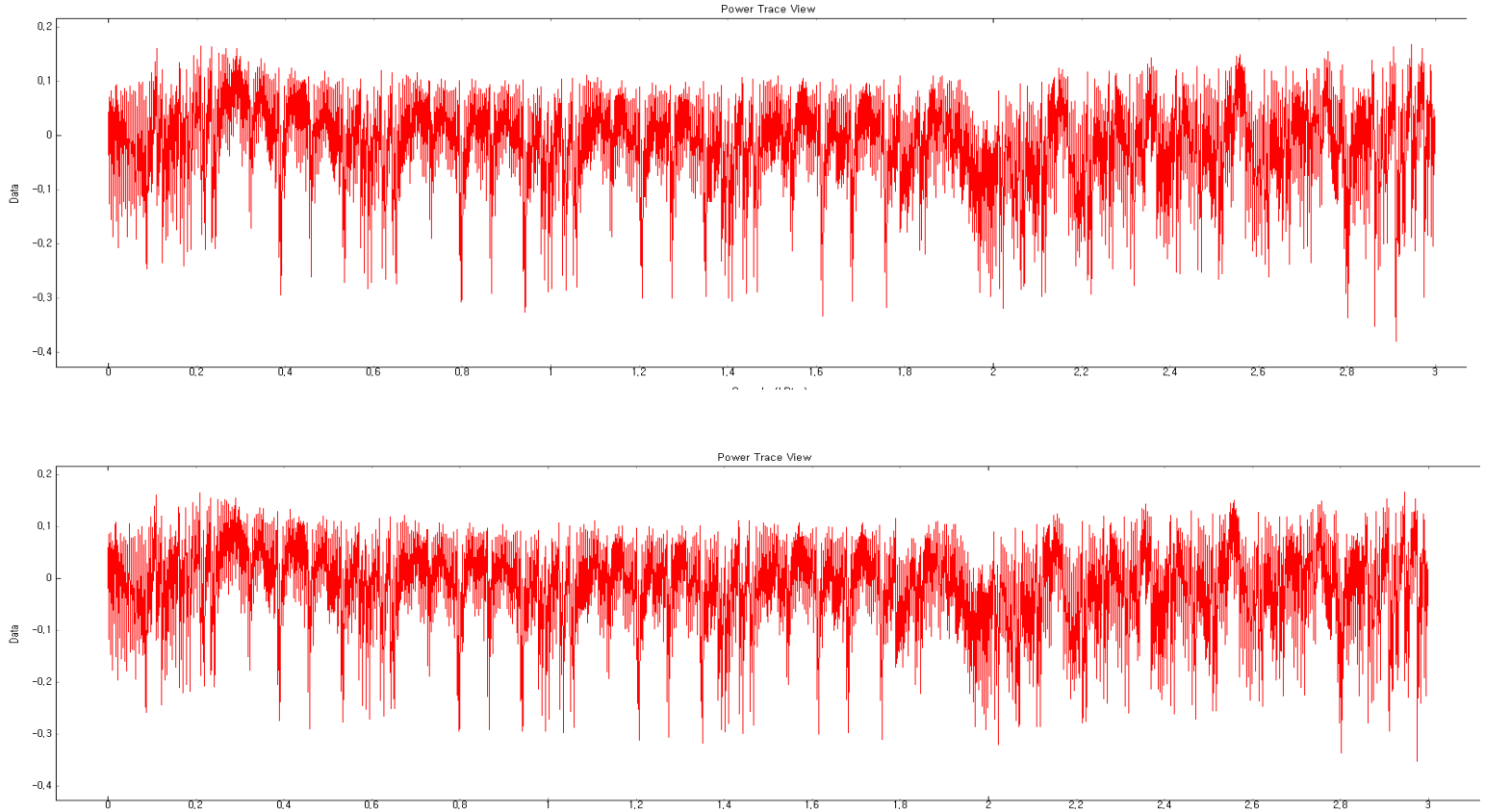
- 부채널 분석(side channel analysis)

  암호 모듈이 탑재된 전자장치에서 암호 알고리즘이 수행되는 순간에 발생되는 전력소모 및 전자기파 등의 누수 정보를 획득 및 가공, 분석하여 암호키 등의 비밀 정보를 획득하는 분석법



부채널 정보를 사용한 해쉬

- 다른 코드는 파형의 모양이 상이함
- 같은 코드의 입력 값이 다르더라도 파형의 모양은 유사
- 입력 값 변경으로 결과의 차이가 존재



부채널 정보를 사용한 해쉬

실험

- Chipwhisperer Lite 장치로 8비트 마이크로 컨트롤러에서 AES 암호 작동

같은 입력값 으로 동작	다른 입력값으로 동작
1.0	0.9938067885007734
0.9987633033242702	0.993804961687550 2
0.9989104574275279	0.9941031847744338
0.9984020806441388	0.993734424492534 5
0.998766839106 11	0.9947864437000536
0.998754747073959	0.9933423864919888
0.9979932322198061	0.995157992249117 7
0.9983948070547921	0.9943279750885147
0.9979049175405524	0.9936681418940517
0.9989637637204385	0.9940308528226844

유의미한 차이를 보임

부채널 정보를 사용한 해쉬

- 마이크로 컨트롤러에서 암호 모듈 작동 시
  같은 입력값을 사용하더라도 노이즈로 같은 파형은 나오지 않음

➢파형으로는 입력값을 알아 낼 수 없음

➢입력값을 알고 있다면 파형의
  상관계수분석으로 같은 파형임을 확인 가능

➢**해시와 유사**

전력파형 해시와 블록체인 자격증명

1. 비트코인 자격증명
   Nonce 값을 변경하여 목표값보다 작은 해시를 찾아 블록 해시로 사용
2. Asic에서 이러한 해시 함수의 연산에 특화 가능한 문제
3. MONERO같은 블록체인에서 ASIC 저항 알고리즘 사용하여 범용 CPU사용 유도

➢ 전력파형 해쉬 사용으로 작업증명에 저전력의 마이크로 컨트롤러를 사용하도록 유도

- 키값과 nonce 값을 마이크로컨트롤러에서 암호 모듈(AES)을 작동시켜 발생한 전력 파형을 블록 헤더와 함께 해쉬
  ➢ 마이크로 컨트롤러의 실행시간 조절하여 자격증명의 시간을 조절가능

- 검증시에 같은 키 값과 nonce 값을 동작 시켜 전력파형의 비교
  ➢ 실제로 작동하여 전력파형을 수집했다는 것을 확인 가능

➢ 저전력의 마이크로 컨트롤러를 사용 블록체인의 에너지소모 문제에 효과적

작업증명 알고리즘

- 입력값으로 키값과 블록 헤더(nonce 대신 수집한 파형 사용) 사용

1. 키값과 nonce값을 사용하여 마이크로 컨트롤러에서 암호작동
2. 작동시의 파형 수집하여 블록헤더에 저장
3. 블록헤더를 해쉬
4. 해쉬의 결과가 목표값보다 작은지 확인
5. 작지 않다면 nonce값을 증가시켜 1~3과정 수행
6. 목표값보다 작으면 블록생성

검증 알고리즘

1. 블록헤더를 해시하여 블록 해시값과 같은지 확인
2. 키 값과 nonce 값을 8비트 작동시켜 파형수집
3. 수집한 파형과 블록 헤더의 파형 값과 상관계수를 비교하여 목표값 이상의 값이 나오는지 확인
4. 목표값 보다 크면 검증 성공
5. 목표값 보다 작다면 검증 실패

검증 알고리즘

1. 블록헤더를 해시하여 블록 해시값과 같은지 확인
2. 키 값과 nonce 값을 8비트 작동시켜 파형수집
3. 수집한 파형과 블록 헤더의 파형 값과 상관계수를 비교하여 목표값 이상의 값이 나오는지 확인
4. 목표값 보다 크면 검증 성공
5. 목표값 보다 작다면 검증 실패

고려사항

1. 수집한 파형의 용량 문제
   적절한 다운 샘플링과 구간을 정하여 크지않도록

2. 마이크로 컨트롤러에서 작동하는 코드의 실행시간 조절

Contents

CryptoCraft LAB

전력 클래스

```
class Power_capture:
    scope = cw.scope()
    scope.default_setup()
    target = cw.target(scope)
    ktp = cw.ktp.Basic()
    def capture_trace(self, key, pt):
        _pt = ''
        for i in range(15, -1, -1):
            _pt = _pt + ('%02x ' % (pt >> (i * 8) & 0xff))
        self.ktp.setPlainType(True)
        self.ktp.setInitialKey(key)
        self.ktp.setInitialText(_pt)
        key, pt = self.ktp.next()

        return cw.capture_trace(self.scope, self.target, pt, key).wave

    def validate_trace(self, key, pt, validate_trace):
        trace = self.capture_trace(key, pt)
        col=abs(np.corrcoef(trace[:], validate_trace[:])[0][1])
        print(col)
        if(col<0.997):
            print("The current trace of the block does not equal the generated trace of the block.")
            return True
        return False
```

블록 클래스

```
class Block:
    def __init__(self, transactions, previous_hash, key, nonce = 0):
        self.timestamp = datetime.now()
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.nonce = nonce
        self.key = key
        self.trace = Power_capture.capture_trace(Power_capture, self.key, self.nonce)
        self.hash = self.generate_hash()

    def print_block(self):
        # prints block contents
        print("timestamp:", self.timestamp)
        print("transactions:", self.transactions)
        print("current hash:", self.get_hash())
        print("nonce:", self.nonce)
```

블록 클래스

```
def nonce_increase(self):
    self.nonce =self.nonce+1;
    self.trace = Power_capture.capture_trace(Power_capture, self.key, self.nonce)

def generate_hash(self):
    # hash the blocks contents
    block_contents = str(self.timestamp) + str(self.transactions) + str(self.previous_hash) + str(self.key)  + str(self.trace)
    block_hash = hashlib.sha256(block_contents.encode()).hexdigest()
    self.hash = block_hash
    return block_hash

def get_hash(self):
    block_contents = str(self.timestamp) + str(self.transactions) + str(self.previous_hash) + str(self.key) + str(
        self.trace)
    return hashlib.sha256(block_contents.encode()).hexdigest()
```

블록

```
class Blockchain:
    def __init__(self):
        self.chain = []
        self.all_transactions = []
        self.key = '0x4B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C'
        self.genesis_block()

    def genesis_block(self):
        transactions = {}
        block = Block(transactions, 0, self.key)

        self.chain.append(block)
        return self.chain

    def add_block(self, transactions):
        previous_block_hash = self.chain[len(self.chain) - 1].hash
        new_block = Block(transactions, previous_block_hash, self.key)
        proof = self.proof_of_work(new_block)
        self.chain.append(new_block)
        return proof, new_block
```

블록체인 클래스

```
def validate_chain(self):
    for i in range(1, len(self.chain)):
        current = self.chain[i]
        previous = self.chain[i - 1]
        if current.hash != current.get_hash():
            print("The current hash of the block does not equal the generated hash of the block.")
            return False
        if previous.hash != previous.get_hash():
            print("The previous block's hash does not equal the previous hash value stored in the current block.")
            return False
        if (Power.validate_trace(current.key, current.nonce, current.trace)):
            print("The current trace of the block does not equal the generated trace of the block.")
            return False
        if (Power.validate_trace(previous.key, previous.nonce, previous.trace)):
            print("The current trace of the block does not equal the generated trace of the block.")
            return False
        if current.previous_hash != previous.hash:
            return False
    return True
```

블록체인 클래스

```
def proof_of_work(self, block, difficulty=1):
    block.generate_hash()
    count=0
    while block.hash[:difficulty] != '0' * difficulty:
        block.nonce_increase()
        block.generate_hash()
        count +=1
    print(count, block.hash)
    return block.hash
```

Q & A