How to Interchain

between diff blockchains

매그니스 인턴 | 아주대학교 소프트웨어학과 김용현

Index

Last Update
2020.02.10

Summary

- 인터체인(Interchain)

  : 서로 다른 체인 간에 트랜잭션을 교환하는 기술

  좁은 의미로 Sidechain으로도 명명

- 대표적인 인터체인 방법

  1) Two Way Pegging · · · · · · · · · · NOT EXIST

  2) Atomic Swap · · · · · · · · · · · · NOT EXIST

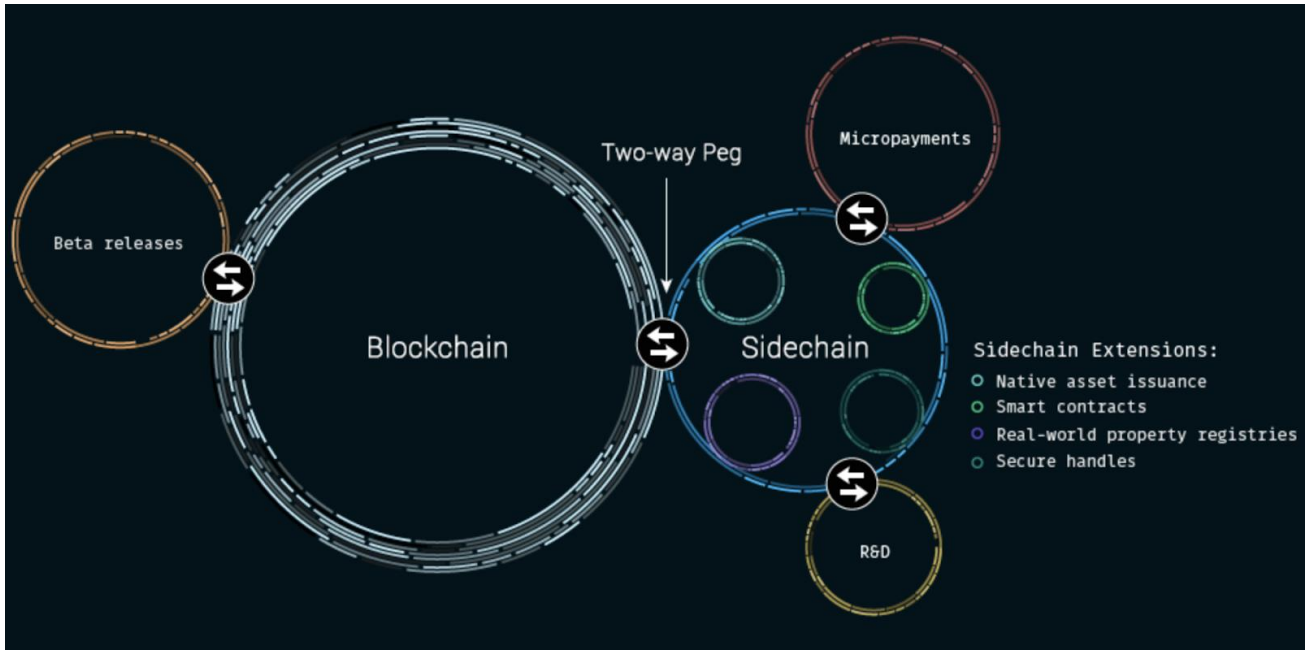
  3) Relayer · · · · · · · · · · · · · · · · EXISTENCE

  4) Exchange · · · · · · · · · · · · · · · EXISTENCE

- 블록체인간 인터체인 목록

  1) Bitcoin Blockchain ⇔ Ethereum Blockchain

  2) Ethereum Blockchain ⇔ EOS Blockchain



Requirement

- Reorg(Reorganization)

  : 체인 내 어떠한 이유로 인해 fork가 일어나 race condition이 되었을 경우,

  다음 생성되는 블록이 한 분기를 선택해 PoW(작업증명)를 하는 과정

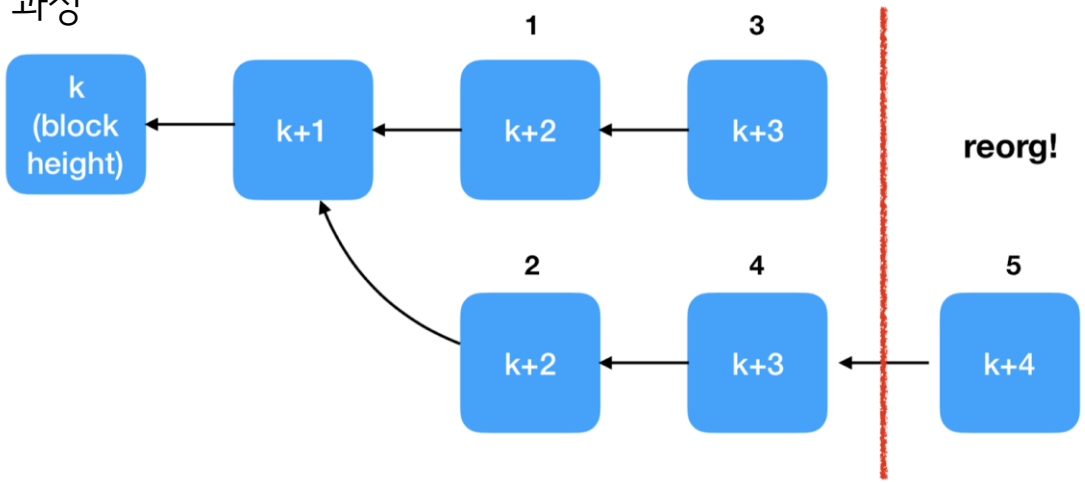
- 보통 6개의 체인(6 confirmation)이 형성됐을 경우,

  reorg가 잘 일어나지 않아 해당 정보가 확실하다고 판단

- 발생 이유

  1) Orphan

  2) 51% Attack



[그림 1] Concept of Reorganization

[1] 리오그(reorganization)을 알아보자: https://steemit.com/kr/@morning/reorganization

Requirement

- SPV(Simplified Payment Verification)

  : 체인의 헤더만 저장해 트랜잭션을 검증할 수 있는 노드 (cf. Full node)

- Bitcoin의 트랜잭션은 Merkle Tree를 이용해 저장,

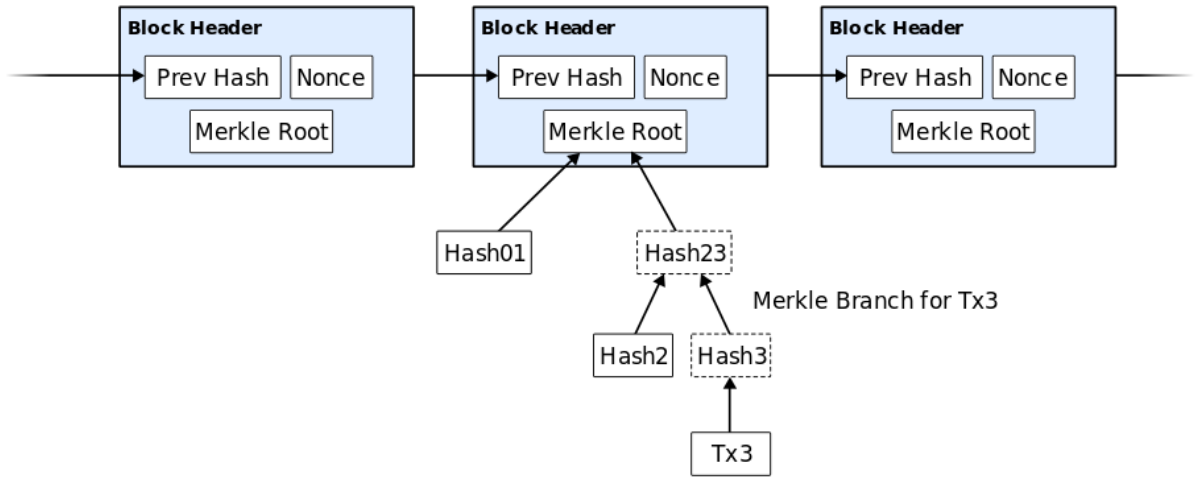
  즉, 헤더 내 해시 값을 통해 PoW를 확인(SPV Proof)

- 헤더 내 필드

  1) Version

  2) hashPrevBlock

  3) hashMerkleRoot



[그림 2] Concept of Reorganization

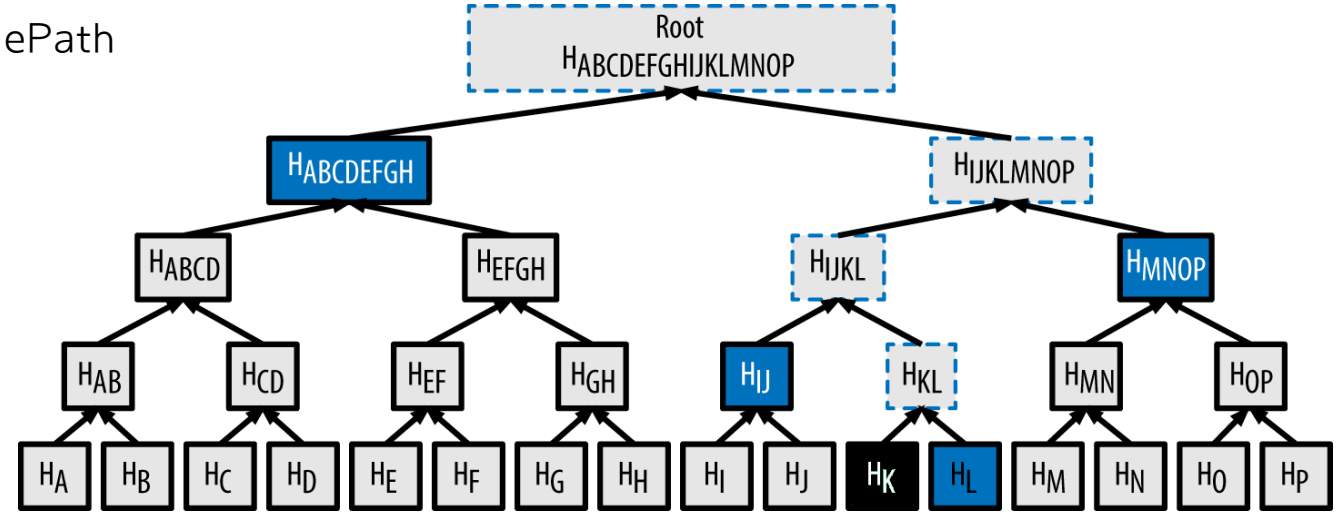
Requirement

- SPV Proof

  : Full node peer로부터 Merkle path(머클 경로)를 통해 hash root를 검증하는 것

  - SPV_node.hashMerkleRoot == Full_node.merklePath
  - SPV 취약점
    1) Full node의 데이터 누락(false negative)
    2) Full node의 SPV node 정보 수집을 통한 DoS 공격
  - SPV Proof로 트랜잭션 포함 여부 검증 이후,

    일정 수의 블록이 쌓이는 지 확인 필요(6 confirmation)



[그림 3] Merkle path about *Hk* transaction

[2] Non-Interactive Proofs of PoW – SPV와 SkipList: https://medium.com/decipher-media/%EB%B8%94%EB%A1%9D%EC%B2%B4%EC%9D%B8-%ED%99%95%EC%9E%A5%EC%84%B1-%EC%86%94%EB%A3%A8%EC%85%98-%EC%8B%9C%EB%A6%AC%EC%A6%88-6-1-nipopow-non-interactive-proofs-of-proof-of-work-spv%EC%99%80-skiplist-b1c7e1213b50

Concepts

- Two Way Pegging – Not Exist

  (Parent chain)
  : 한쪽 체인에서 토큰 동결(Pegging) 증명 이후,
  (Sidechain)
  다른 쪽 체인이 확인해 동일 가치 토큰을 발행

〈진행 과정〉

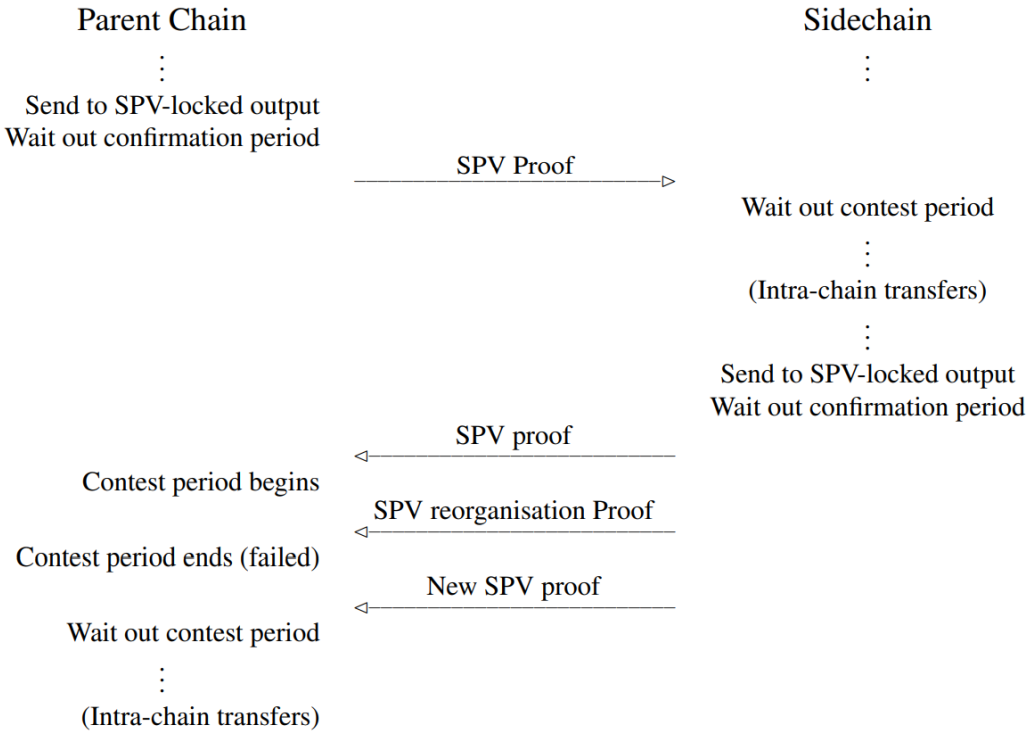
1) confirmation period* 동안 lock 걸린 SPV 트랜잭션 생성

2) 트랜잭션 lock 해제된 후, 상대 체인은 SPV Proof 진행

   (단, Reorg가 일어날 수 있으므로 contest period* 이후에 검증.)

3) Reorg가 일어났을 경우, 상대 체인은 reorganization proof* 진행

   이후 다시 SPV Proof를 진행해 토큰 교환을 진행



[그림 4] Symmetric Two Way Peg Protocol

- confirmation period: 상위 체인에 대해 확인하는 기간. 토큰이 상대 체인으로 전달되기 전에 진행된다.
- contest period: 새롭게 전송된 토큰이 상대 체인에서 쓰지 않는 기간. 해당 논문에서는 1~2일 정도 기간을 잡는 것을 권유한다.
- reorganization proof: contest period 중 체인 내에서 SPV-locked 트랜잭션에 포함되지 않은 블록이 있는지 검증하는 작업. 만약 발견했을 경우, 토큰 교환을 무효 처리한다.

Concepts

- Two Way Pegging – Not Exist

– 필수 조건

1) 상대 체인의 트랜잭션을 SPV로 검증 가능 (때문에 같은 타입의 블록체인 프로토콜을 사용함)

2) 상대 체인의 Reorg 관찰 및 reorganization proof를 통한 토큰 전송 되돌리기 가능

– 주의 사항

사이드체인은 많은 체인의 토큰과 인터체인 할 수 있지만 서로 다른 블록체인 토큰은 교환할 수 없음

1) 서로 다른 블록체인 플랫폼간 보안을 보장할 수 없음

2) 서로 다른 블록체인의 트랜잭션을 검증할 수 있는 SPV 프로토콜이 있어야 함. (이는 기술적, 정치적 이유 등으로 인해 어렵다고 알려짐)

---

[3] Adam Back, 2014, "3.2 Symmetric two-way peg", *Enabling Blockchain Innovation with Pegged Sidechains,*

[4] Interchain Overview: https://medium.com/decipher-media/%EB%B8%94%EB%A1%9D%EC%B2%B4%EC%9D%B8-%ED%99%95%EC%9E%A5%EC%84%B1-%EC%86%94%EB%A3%A8%EC%85%98-%EC%8B%9C%EB%A6%AC%EC%A6%88-3-1-interchain-overview-8ed188d5b7d9

Concepts

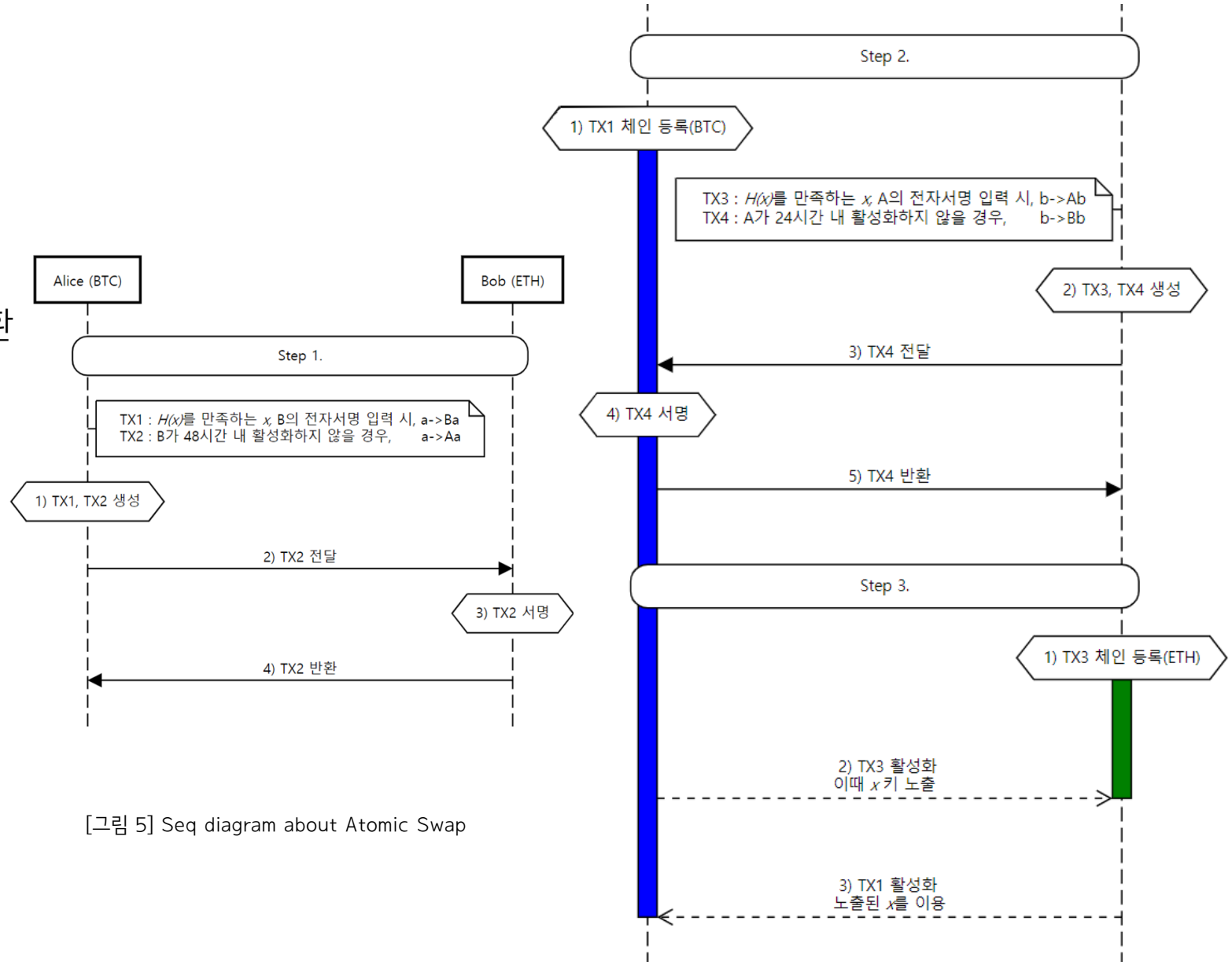
- Atomic Swap – Not Exist

  : 서로 다른 체인 간에 서로 다른 자산을 교환

  (Ex. A의 a(BTC) ⇔ B의 b(ETH))

〈특이사항〉

1) 각 체인마다 트랜잭션을 2개 사용
2) 계약 실패 시, 자금 회수에 있어 오랜 시간 걸림
3) PoW 기반의 같은 해시 알고리즘을 사용해야함



[그림 5] Seq diagram about Atomic Swap

[5] Atomic Swap: 탈중앙화된 화폐 교환: https://www.blocko.io/blockchain/atomic-swap/

Concepts

- Relayer – Existence BTC RELAY

  : 블록체인을 정보(Ex. 블록 헤더)를

  다른 블록체인에게 알려주는 개체

〈특이사항〉

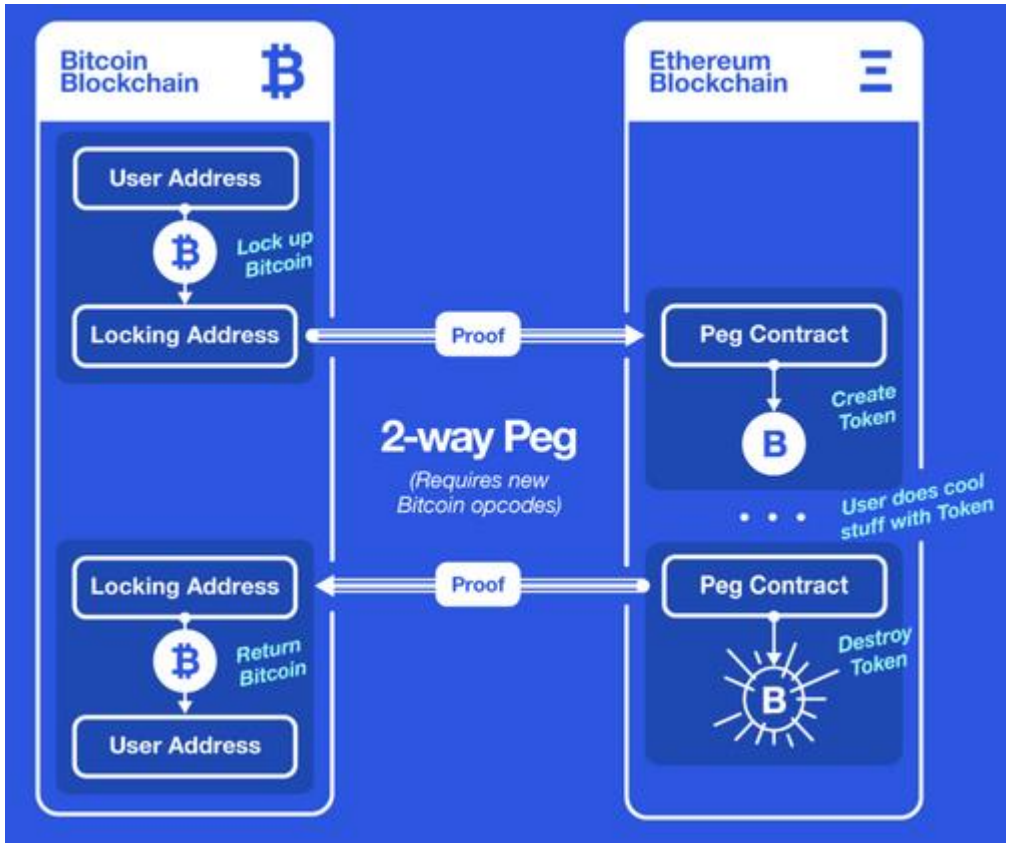
1) 블록체인 플랫폼이 서로 다르더라도 가능

2) 체인 외부에 있는 주체로, 탈중앙화적이지 않음

〈종류〉

1) BTC Relay (Bitcoin → Ethereum)

2) RADAR Relay (Ethereum ↔ ERC20 token)



[그림 6] BTC Relay blueprint

Concepts

- Exchanges – Existence Peatio

: 거래소 내 상장된 코인들을 거래해주는 중개소

〈구성〉

1) Makers
(거래 생성자)
2) Takers
(거래 선택자)
3) Exchange
(중개자)

〈특징〉

1) Centralized exchange
2) 거래소마다 HTS(Home Trading System)와 매매체결 시스템 보유
3) 각 코인마다 Hot Wallet 지급. 거래소가 관리
4) 거래소 모델 → "환전소, 브로커, 펀드"
5) Multisig 지갑을 통해 보안 강화

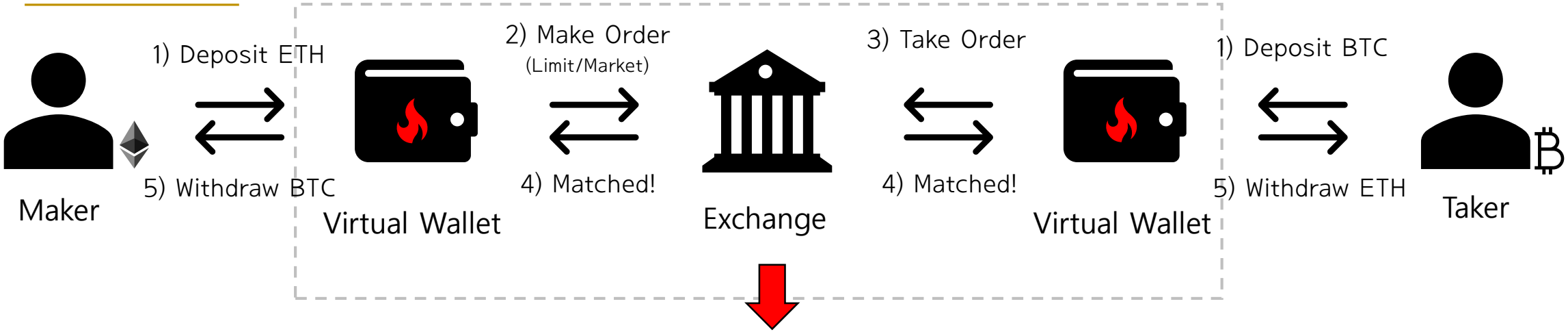
[그림 7] Top 10 Korean Cryptocurrency Exchanges, 2018

[8] Bitcoin Exchange: https://www.investopedia.com/terms/b/bitcoin-exchange.asp

Case: Sell ETH by BTC

Overall

EXCHANGE'S TERITORY



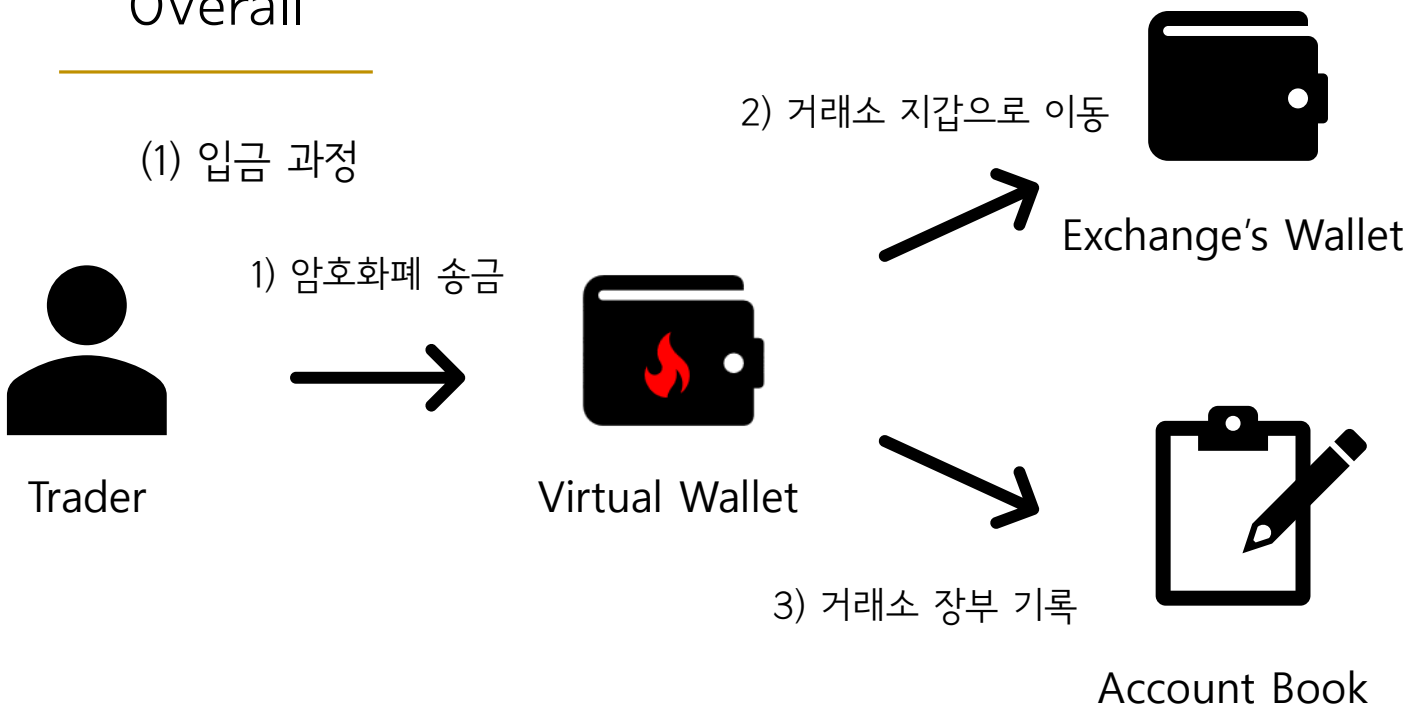
Order Book
(BTC to ETH)

Limit Order
(제한 거래)

Market Order
(시장가 거래)

Overall

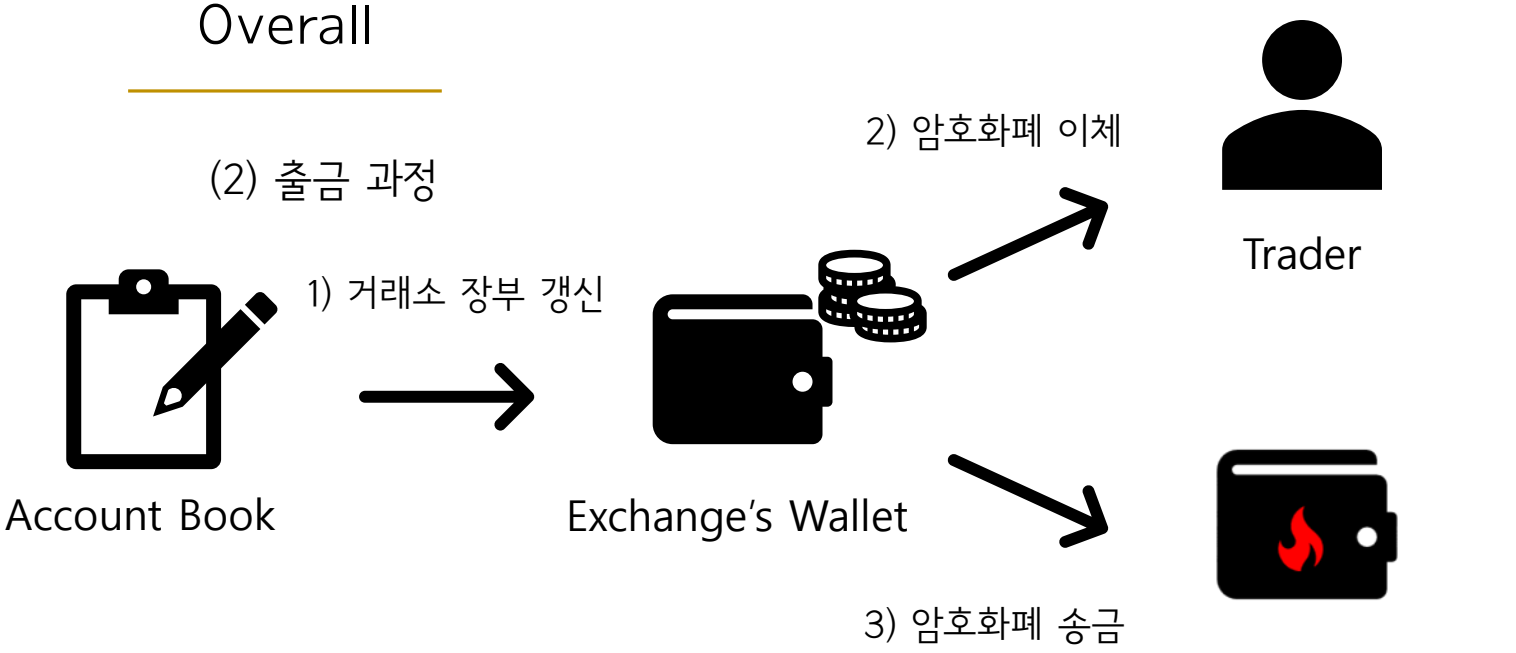
(1) 입금 과정

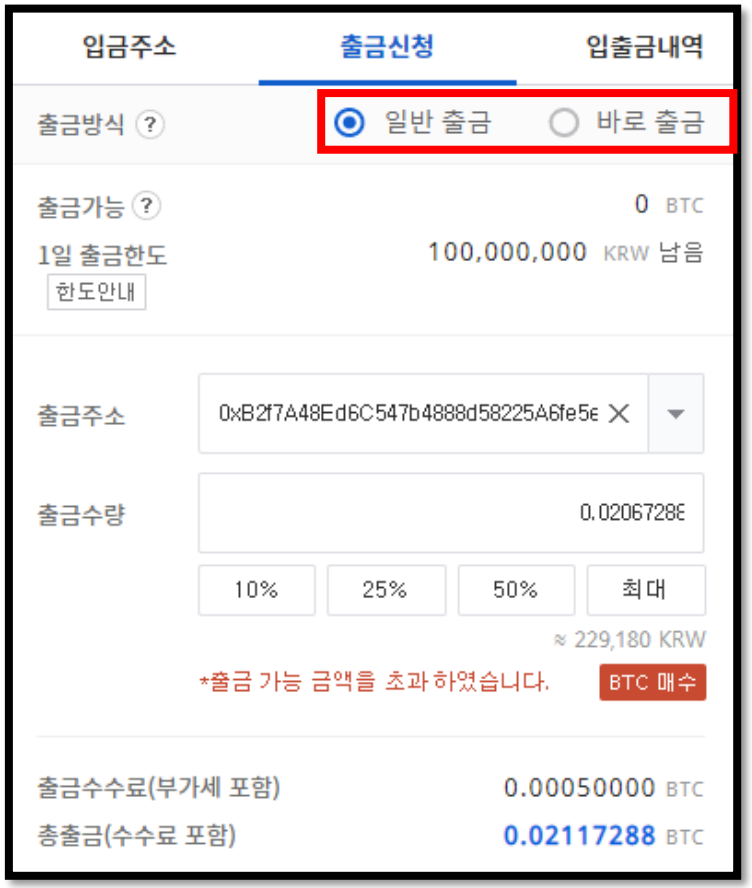


1) 암호화폐를 거래소에서 지정해준 입금주소로 송금
2) 입금 프로세스를 거쳐 거래소 지갑으로 송금
3) 입금된 암호화폐를 거래소 장부에 기록. 이후 거래소 계정 잔고에 반영.

Overall

(2) 출금 과정





1) 출금 수량만큼 거래소 장부 갱신 후 이체 요청
2) 출금 수수료를 제외한 금액만큼 암호화폐를 이체
3) 만약 거래소 내 다른 회원에게 이체할 경우, 그 회원의 가상 지갑으로 보내 수수료를 지불하지 않을 수 있음.
(단, 체인 네트워크에 기록되지 않고 오직 거래소 장부에만 기록됨.)

Vulnerability

- 탈중앙화 되지 않아 해커들로부터 공격 대상
- KYC(Know Your Customer)로 인한 고객 정보
- Multisig 지갑 3개의 private key
  1) User Key · · · · · · · · User
  2) Exchange Key · · · · Exchange
  3) Backup Key · · · · · · User

  → 실거래 시, 2개의 private key 사용
- Rounding, Payment
  소수점 처리 시 결제가 같이 일어날 경우 손해를 볼 수 있음.
- Pump and Dump

Method

- BTC Relay

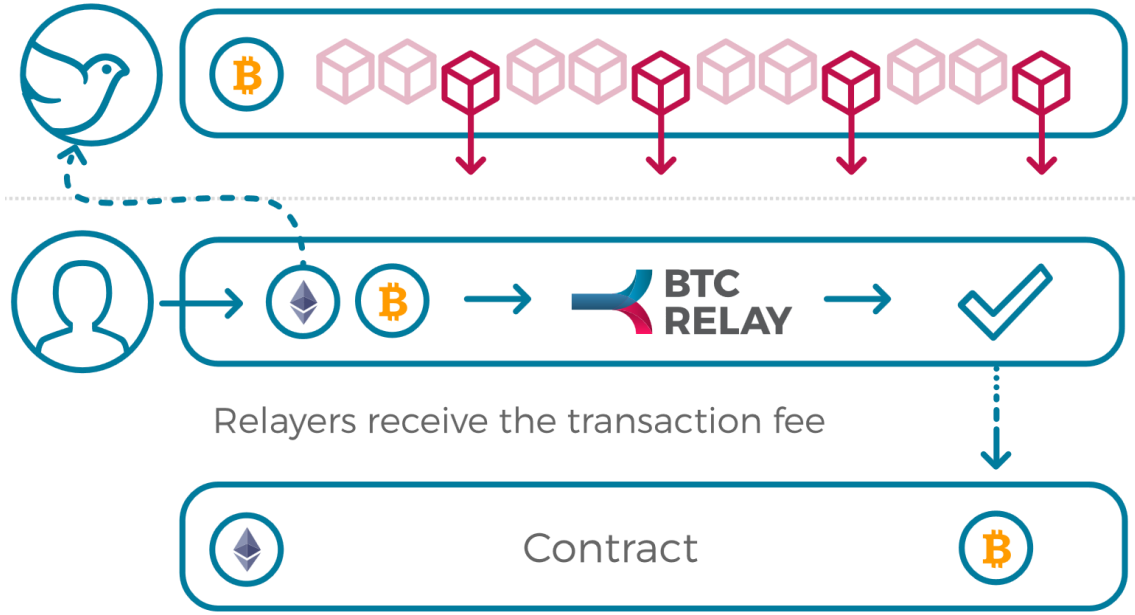
: **비트코인 블록 헤더**를 저장하는 **이더리움 스마트 컨트랙트.**

〈구성〉

1) Relayer
(중개자)
2) BTC Relay Smart Contract (*BTCRelay*)
(SPV Client 역할)
3) Ethereum Application
(Merkle 증명 시 사용)

〈특징〉

1) 단방향 통신만 가능 (Bitcoin → Ethereum)
2) Reorg 처리 복잡함
3) Relayer들에게 인센티브를 통한 수수료 경쟁을 통해 탈중앙화 가능
4) 오직 1개의 Fullnode(blockchain.info)만 사용
5) Merkle 증명 시(SPV Proof) 스마트 컨트랙트 혹은 블록체인 밖에서 수행할 수 있음

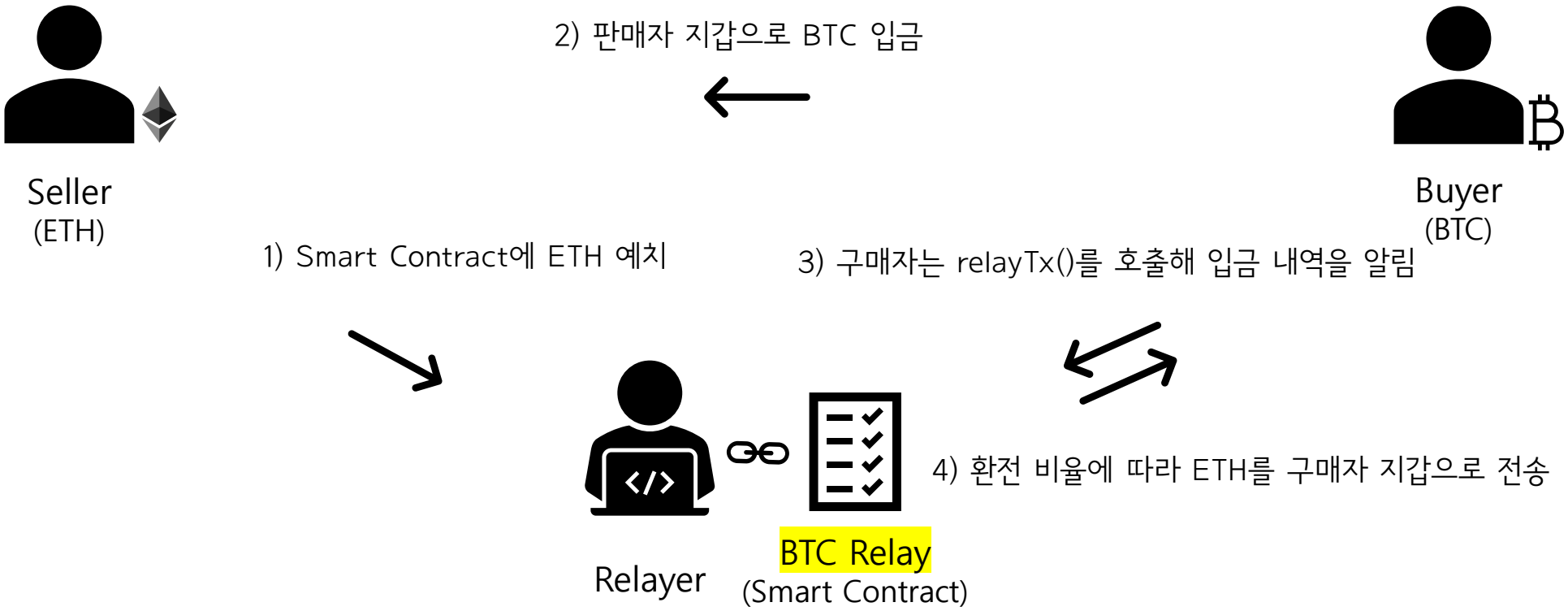


[그림 6] BTC Relay process

단, 이 경우 스마트 컨트랙트에서 실행 시 완전히 신뢰 가능하지만 비용(gas)이 많이 들며, 블록체인 밖에서 실행 시 비용이 적게 들지만 중앙화 이슈가 존재한다.

[6] 블록체인 확장성 솔루션 시리즈 3-2:: Relayer: https://medium.com/decipher-media/%EB%B8%94%EB%A1%9D%EC%B2%B4%EC%9D%B8-%ED%99%95%EC%9E%A5%EC%84%B1-%EC%86%94%EB%A3%A8%EC%85%98-%EC%8B%9C%EB%A6%AC%EC%A6%88-3-2-relayer-1efdef52b16b
[7] BTC Relay Documentation Release 1.0: https://buildmedia.readthedocs.org/media/pdf/btc-relay/latest/btc-relay.pdf

Overall



※주기적으로 BTC 블록 헤더를 BTC Relay에 저장 이 과정에서 Relayer는 BTC Relay에 해당 블록 수수료(fee)를 기록한다.
이를 통해 BTC Relay에서 구매자의 입금 내역을 확인 해당 블록이 선택됐을 경우, 수수료에 해당하는 ETH를 받게 된다.

Detail

(1) fetchd.py – main()

```python
def main():
    # logging.basicConfig(level=logging.DEBUG)
    logger.info("fetchd using PyEPM %s" % __version__)

    parser = ArgumentParser()
    parser.add_argument('-s', '--sender', required=True, help='sender of tra
    parser.add_argument('-r', '--relay', required=True, help='relay contract

    parser.add_argument('--rpcHost', default='127.0.0.1', help='RPC hostname
    parser.add_argument('--rpcPort', default='8545', type=int, help='RPC por
    parser.add_argument('--startBlock', default=0, type=int, help='block num
    parser.add_argument('-w', '--waitFor', default=0, type=int, help='number
    parser.add_argument('--gasPrice', default=int(10e12), type=int, help='ga
    parser.add_argument('--fetch', action='store_true', help='fetch blockhea
    parser.add_argument('-n', '--network', default=BITCOIN_TESTNET, choices=
    parser.add_argument('-d', '--daemon', default=False, action='store_true'
    parser.add_argument('--feeVTX', default=0, type=int, help='fee to charge
    parser.add_argument('--feeRecipient', help='address of fee recipient')

    args = parser.parse_args()
```

1) argument 설정

```python
# this will not handle exceptions or do retries.  need to use -d switch if desired
if not args.daemon:
    run(feeVerifyTx, feeRecipient, doFetch=args.fetch, network=args.network, startBlock=args.startBlock)
    return

while True:
    for i in range(4):
        try:
            run(feeVerifyTx, feeRecipient, doFetch=args.fetch, network=args.network, startBlock=args.startBlock)
            sleep(SLEEP_TIME)
        except Exception as e:
            logger.info(e)
            logger.info('Retry in 1min')
            sleep(60)
            continue
        except:  # catch *all* exceptions
            e = sys.exc_info()[0]
            logger.info(e)
            logger.info('Rare exception')
            raise
        break
```

2), 3) run() 실행 구문

1) 전역변수 및 argument 설정
2) args에 데몬이 있을 경우, SLEEP_TIME(5분)을 주기로 run() 반복 실행
3) args에 데몬이 없을 경우, run() 1번 실행

SLEEP_TIME: 5 * 60 (sec)
GAS_FOR_STORE_HEADERS: 1,200,000 (wei)
CHUNK_SIZE: 5
nTime: 5

Detail

(2) fetchd.py – run()

```
def run(feeVerifyTx, feeRecipient, doFetch=False, network=BITCOIN_TES
    chainHead = getBlockchainHead()
    if not chainHead:
        raise ValueError("Empty BlockchainHead returned.")
    chainHead = blockHashHex(chainHead)
    logger.info('BTC BlockchainHead: %s' % chainHead)
```

1) chainHead 설정

```
# refetch if needed in case contract's HEAD was orphaned
if startBlock:
    contractHeight = startBlock
else:
    contractHeight = getLastBlockHeight()
realHead = get_hash_by_height(contractHeight, network=network)
heightToRefetch = contractHeight
```

2) realHead 설정

```
while chainHead != realHead:
    logger.info('@@@ chainHead: {0}  realHead: {1}'.format(chainHead, realHead))
    fetchHeaders(heightToRefetch, 1, 1, feeVerifyTx, feeRecipient, network=network)

    # wait for some blocks because Geth has a delay (at least in RPC), of
    # returning the correct data.  the non-orphaned header may already
    # be in the Ethereum blockchain, so we should give it a chance before
    # adjusting realHead to the previous parent
    #
    # realHead is adjusted to previous parent in the off-chance that
    # there is more than 1 orphan block
    # for j in range(4):
    instance.wait_for_next_block(from_block=instance.last_block(), verbose=True)

    chainHead = blockHashHex(getBlockchainHead())
    realHead = get_hash_by_height(heightToRefetch, network=network)

    heightToRefetch -= 1
```

3) chainHead != realHead

1) getBlockchainHead()를 통해 BTC Relay에서 가장 마지막 블록의 헤더 값 가져옴
   이때, 헤더가 존재하지 않을 경우 프로그램이 종료되므로, BTC Relay 배포(deploy)시 반드시 첫번째 블록 헤더 설정해야 함
2) startBlock, getLastBlockHeight()를 통해 BTC Relay에서 가장 마지막 블록의 헤더 값 가져옴
3) chainHead와 realHead이 서로 다를 경우, 최대 5번까지 reorg 처리 루프 실행

Detail

(2) fetchd.py – run()

```
while chainHead != realHead:
    logger.info('@@@ chainHead: {0}  realHead: {1}'.format(chainHead, realHead))
    fetchHeaders(heightToRefetch, 1, 1, feeVerifyTx, feeRecipient, network=network)
```

```
instance.wait_for_next_block(from_block=instance.last_block(), verbose=True)

chainHead = blockHashHex(getBlockchainHead())
realHead = get_hash_by_height(heightToRefetch, network=network)

heightToRefetch -= 1

if heightToRefetch < contractHeight - 10:
    if i == nTime - 1:
        # this really shouldn't happen since 2 orphans are already
        # rare, let alone 10
        logger.info('@@@@ TERMINATING big reorg? {0}'.format(heightToRefetch))
        sys.exit()
    else:
        logger.info('@@@@ handle orphan did not succeed iteration {0}'.format(i))
        break  # start the refetch again, this time ++i
```

4) 헤더 페치 후 reorg 싱크 처리 과정

```
actualHeight = last_block_height(network)  # pybitcointools 1.1.33

if startBlock:
    instance.heightToStartFetch = startBlock
else:
    instance.heightToStartFetch = getLastBlockHeight() + 1

logger.info('@@@ startFetch: {0} actualHeight: {1}'.format(instance.heightToStart

chunkSize = CHUNK_SIZE
fetchNum = actualHeight - instance.heightToStartFetch + 1
numChunk = fetchNum / chunkSize
leftoverToFetch = fetchNum % chunkSize
```

```
if doFetch:
    fetchHeaders(instance.heightToStartFetch, chunkSize, numChunk, feeVerifyTx, fe
    fetchHeaders(actualHeight - leftoverToFetch + 1, 1, leftoverToFetch, feeVerify
```

5) actualHeight부터 heightToStartFetch까지 헤더 가져오기

4) chunkSize, numChunk 모두 1로 설정해 가장 최근 블록부터 고아블록까지 reorg 되기 직전의 블록을 찾아야 함
   단, 10개의 블록이 넘어가도록 서로 일치하는 블록을 발견하지 못한다면 현재 이더리움 체인이 고아가 되었다고 가정
   헤더가 같을 때까지 이더리움 블록체인을 받아와 4)번 과정을 반복 → 5번 이상 시 big reorg로 간주 → sys.exit()
5) 실제 높이(actualHeight)부터 시작 높이(heightToStartFetch)만큼 비트코인 체인의 블록 헤더를 가져옴

BTC Relay 배포 이후,
마지막 블록의 헤더값을 설정하는 함수

Detail

(1) btcrelay.se – setInitialParent()

Serpent(se): Python을 기반으로 만든 언어.
chainwork: 전체 체인의 work 총량. chainwork 값이 클수록 강력하고 올바른 체인임을 알 수 있다.

```
def setInitialParent(blockHash, height, chainWork):
    # reuse highScore as the flag for whether setInitialPa
    if self.highScore != 0:
        return(0)
    else:
        self.highScore = 1  # matches the score that is se

    self.heaviestBlock = blockHash
```

1) 2) 초기 블록 설정

```
    # _height cannot be set to -1 because inMainChain() assumes
    # a block with height0 does NOT exist (thus we cannot allow
    # real genesis block to be at height0)
    m_setHeight(blockHash, height)

    # do NOT pass chainWork of 0, since score0 means
    # block does NOT exist. see check in storeBlockHeader()
    m_setScore(blockHash, chainWork)

    # other fields do not need to be set, for example:
    # _ancestor can remain zeros because self.internalBlock[0]

    return(1)
```

3) blockHash를 key로 가지는 블록의 height와 chainwork 저장

1) 플래그 highScore를 통해 초기화 함수가 오직 한번만 실행되도록 설정
2) 처음 가져온 블록(blockHash)을 heaviestBlock으로 설정
3) 해당 블록에 대한 height, chainwork 저장
4) BTC Relay 배포 및 초기화 이후, 다른 함수들은 모두 event-driven 방식을 통해 Relayer와 Ethereum App에서 호출됨

[Relayer] – 블록 헤더 값을 BTC Relay에 기록할 때 호출하는 함수
만약 Relayer와 수수료 받을 노드가 같다면, storeBlockWithFee() 호출

Detail

(2) incentive.se – storeBlockchainWithFeeAndRecipient()

```
# first 16 bytes are the last gas price; last 16 bytes is the changeRecipientFe
data gasPriceAndChangeRecipientFee
# sets _feeInfo for the block and updates gasPriceAndChangeRecipientFee
def storeBlockWithFee(blockHeaderBytes:str, feeWei):
    return(self.storeBlockWithFeeAndRecipient(blockHeaderBytes, feeWei, msg.ser

# this indirection is needed by test_fee.py, but a configurable recipient may t
def storeBlockWithFeeAndRecipient(blockHeaderBytes:str, feeWei, feeRecipient):
    beginGas = msg.gas
    res = self.storeBlockHeader(blockHeaderBytes)
```

1) 초기 가스 및 헤더값 저장

```
if res:
    blockHash = m_dblShaFlip(blockHeaderBytes)
    m_setFeeInfo(blockHash, feeWei, feeRecipient)
    remainingGas = msg.gas
```

2) 수수료 설정 및 남은 가스 계산

```
    gLastGasPrice = m_getLastGasPrice()
    minGP = 1023*gLastGasPrice/1024
    maxGP = 1025*gLastGasPrice/1024
    if tx.gasprice < minGP:
        currGP = minGP
    elif tx.gasprice > maxGP:
        currGP = maxGP
    else:
        currGP = tx.gasprice

    gChangeRecipientFee = 2 * currGP * (beginGas - remainingGas) #

    m_setGasPriceAndChangeRecipientFee(currGP, gChangeRecipientFee)

    return(res)
```

3) gasPriceAndChangeRecipientFee에 큰 영향을 주지 않기 위해 클램핑

1) 초기 가스와 헤더값 저장
2) 수수료 정보 및 남은 가스 계산
   수수료를 저장하는데 필요한 가스비를 동적으로 계산해 다른 Relayer가 블록 헤더값을 다시 기록할 때 참고한다.
3) 이전 가스량의 1/1024 스케일 클램핑을 통해 수수료를 설정하고, 교체 비용은 현재 수수료의 2배로 설정

Detail

(3) incentive.se – feePaid(), changeFeeRecipient()

[Relayer] – recipient에게 기록된 수수료를 지불하는 함수

[Relayer] – 기존 recipient를 교체할 때 사용하는 함수

```
# if sufficient fee for 'txBlockHash' is provided, pay the fee
# and return 1.  otherwise return 0.
# It is the recipient's responsibility to accept the fee.
# This does NOT return any funds to incorrect callers
def feePaid(txBlockHash, amountWei):
    if msg.value >= amountWei:
        if msg.value > 0:
            feeRecipient = m_getFeeRecipient(txBlockHash)

            if self.depthCheck(0):
                send(feeRecipient, msg.value)  # recipient's r
                log(type=EthPayment, feeRecipient, msg.value)
            else:
                ~invalid()
        return(1)
    return(0)
```

1) feePaid

```
# callers must sufficiently send the block's current fee, AND feeWei mus
# than the block's current fee
# This does NOT return any funds to incorrect callers
def changeFeeRecipient(blockHash, feeWei, feeRecipient):
    if !self.feePaid(blockHash, m_getChangeRecipientFee(), value=msg.val
        return(0)

    # feeWei is only allowed to decrease
    if feeWei < m_getFeeAmount(blockHash):
        m_setFeeInfo(blockHash, feeWei, feeRecipient)
        return(1)

    return(0)
```

2) changeFeeRecipient

1) 기록된 수수료보다 높은 금액 송금 확인
2) txBlockHash에 저장된 recipient의 주소 가져옴
3) recipient에게 송금된 금액을 전달
   성공 시 1, 실패 시 0을 반환

1) 블록 헤더를 기록한 recipient에게 수수료 지불
2) 바꾸는 수수료가 이전 수수료보다 더 가격이 낮은지 확인
3) 수수료와 recipient를 교체
   성공 시 1, 실패 시 0을 반환

Detail

[Buyer] – BTC 송금을 검증하는 함수.

블록 해시값을 이용해
SPV proof를 진행하는 함수

(1) btcrelay.se – verifyTx(), helperVerifyHash__()

```
def verifyTx(txBytes:str, txIndex, sibling:arr, txBlockHash):
    txHash = m_dblShaFlip(txBytes)
    if len(txBytes) == 64:  # todo: is check 32 also needed?
        log(type=VerifyTransaction, txHash, ERR_TX_64BYTE)
        return(0:uint256)
    res = self.helperVerifyHash__(txHash, txIndex, sibling, txBlock
    if res == 1:
        return(txHash:uint256)
    else:
        # log is done via helperVerifyHash__
        return(0:uint256)
```

1) verifyTx()

```
def helperVerifyHash__(txHash:uint256, txIndex, sibling:arr, txBlockHash):
    if !self.feePaid(txBlockHash, m_getFeeAmount(txBlockHash), value=msg.value):
        log(type=VerifyTransaction, txHash, ERR_BAD_FEE)
        return(ERR_BAD_FEE)

    if self.within6Confirms(txBlockHash):
        log(type=VerifyTransaction, txHash, ERR_CONFIRMATIONS)
        return(ERR_CONFIRMATIONS)

    if !self.priv_inMainChain__(txBlockHash):
        log(type=VerifyTransaction, txHash, ERR_CHAIN)
        return(ERR_CHAIN)

    merkle = self.computeMerkle(txHash, txIndex, sibling)
    realMerkleRoot = getMerkleRoot(txBlockHash)

    if merkle == realMerkleRoot:
        log(type=VerifyTransaction, txHash, 1)
        return(1)

    log(type=VerifyTransaction, txHash, ERR_MERKLE_ROOT)
    return(ERR_MERKLE_ROOT)
```

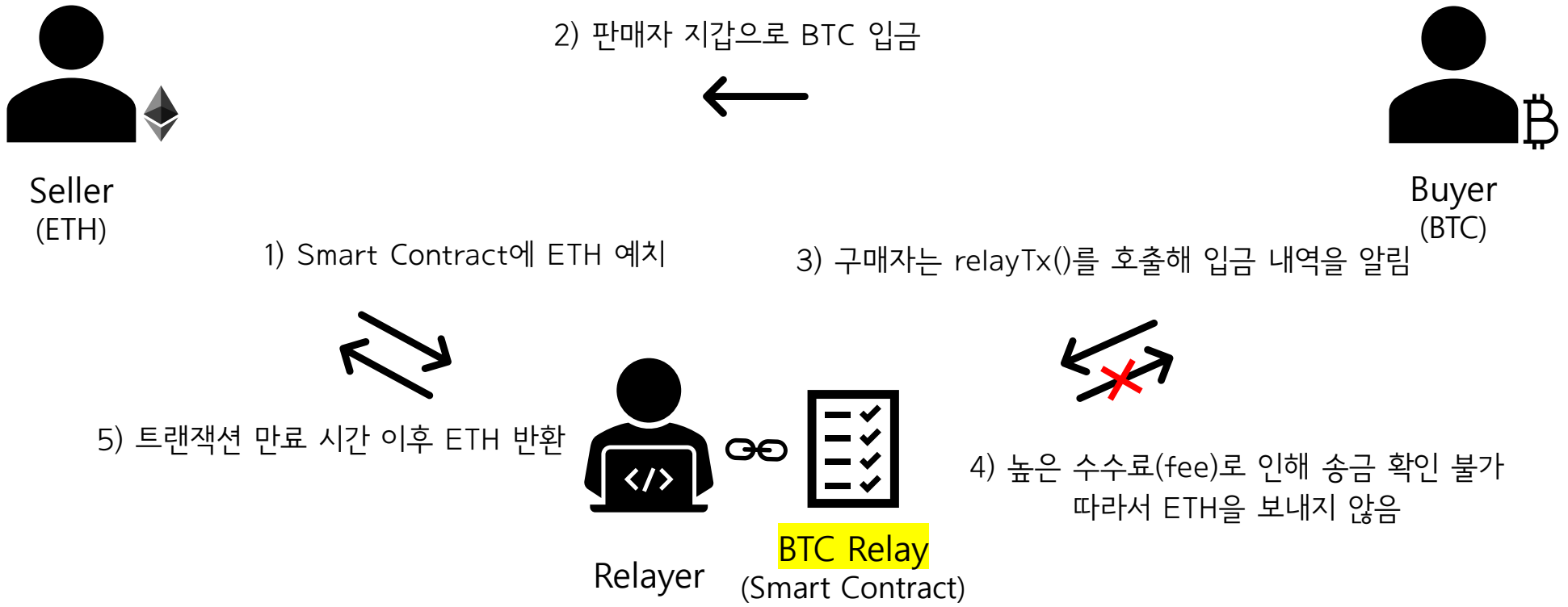
2) helperVerifyHash__()

1) 트랜잭션(txBytes), 트랜잭션 인덱스(txIndex)
   트랜잭션 정보(sibiling), 해시값(txBlockHash) 구함
2) 위 4개를 매개변수로 하여 helperVerifyHash() 호출
   해당 함수를 통해 BTC 송금을 확인

1) 검사할 블록에 대해 수수료 지불
2) 해당 블록의 6 Confirmation 확인
3) 해당 블록이 메인 체인에 있는지 확인
4) 머클 루트 확인 후(SPV proof) 결과 반환

Vulnerability

relayTx() -> helperVerifyHash__() -> feePaid()
getBlockHeader() -> feePaid()
changeFeeRecipient() -> feePaid()



※의도적으로 블록 수수료(fee)를 높은 가격으로 설정

Method

- Peatio

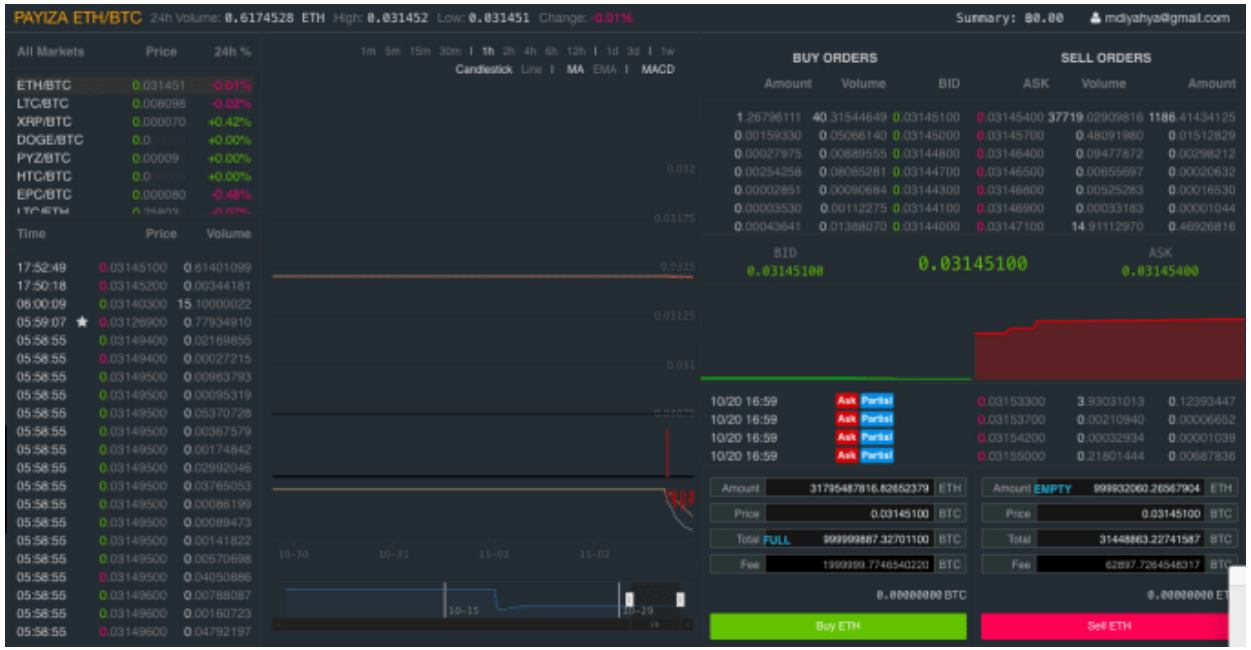
  : Open-Source Assets Exchange

〈특징〉

1) Ruby로 개발된 프로젝트
2) Peatio-Core 사용
   → DBMS 및 RabitMQ, 인증과 관련한 공통 프레임워크

〈구성〉

1) OS: Ubuntu 14.04 LTS
2) Back-end: Ruby on Rails
3) DB: MySQL, Redis
4) Etc: RabbitMQ, PhantomJS, ImageMagick



[그림 7] Peatio customization cryptocurrency exchange

Redis(Remote Dictionary Server): 키-값 구조의 비정형 데이터를 저장하고 관리하기 위한 오픈 소스 기반의 비관계형 데이터베이스 관리 시스템
RabbitMQ: 오픈 소스 메시지 브로커 소프트웨어. AMQP(Advanced Message Queuing Protocol)를 구현
PhantomJS: 웹 페이지 상호작용을 자동화하기 위해 사용되는 헤드리스 브라우저
ImageMagick: 그래픽 이미지를 새로 만들거나, 고치는 데 사용되는 오픈 소스 소프트웨어

[8] Peatio, an open-source assets exchange: https://github.com/peatio/peatio

Method

- 현재 Bitcoin(btc) ⇔ 위안(cny)간에 거래 가능
- config/currencies.yml에서 화폐 종류를 추가할 수 있음.
- 사용하는 데몬 종류
    1) AMQP
    2) global_state
    3) hot_wallets
    4) payment_transaction
    5) stats
    6) websocket_api
    7) withdraw_audit

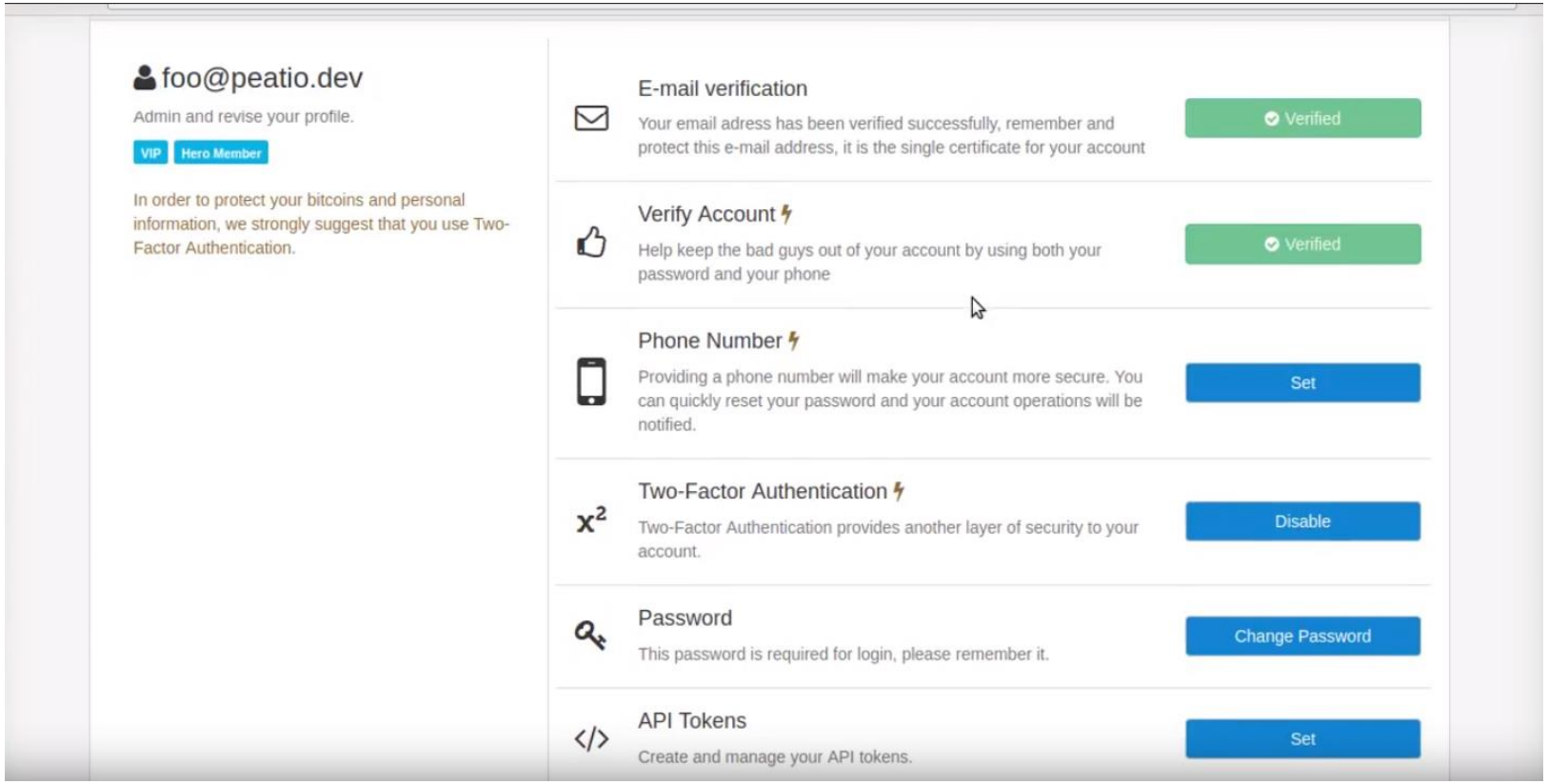
config > currencies.yml.example

```yaml
- id: 1
  key: renminbi
  code: cny
  symbol: "¥"
  coin: false
  precision: 2
  assets:
    accounts:
      -
        bank: '招商银行'
        address: '6225 0885 6022 3501'
        password: '111111'
        tel: '95555'
- id: 2
  key: satoshi
  code: btc
  symbol: "฿"
  coin: true
  quick_withdraw_max: 0
  rpc: http://username:password@127.0.0.1:18332
  blockchain: https://blockchain.info/tx/#{txid}
  address_url: https://blockchain.info/address/#{address}
  assets:
    accounts:
      -
        address: 1HjfnJpQmANtuW7yr1ggeDfyfe1kDK7rm3

```

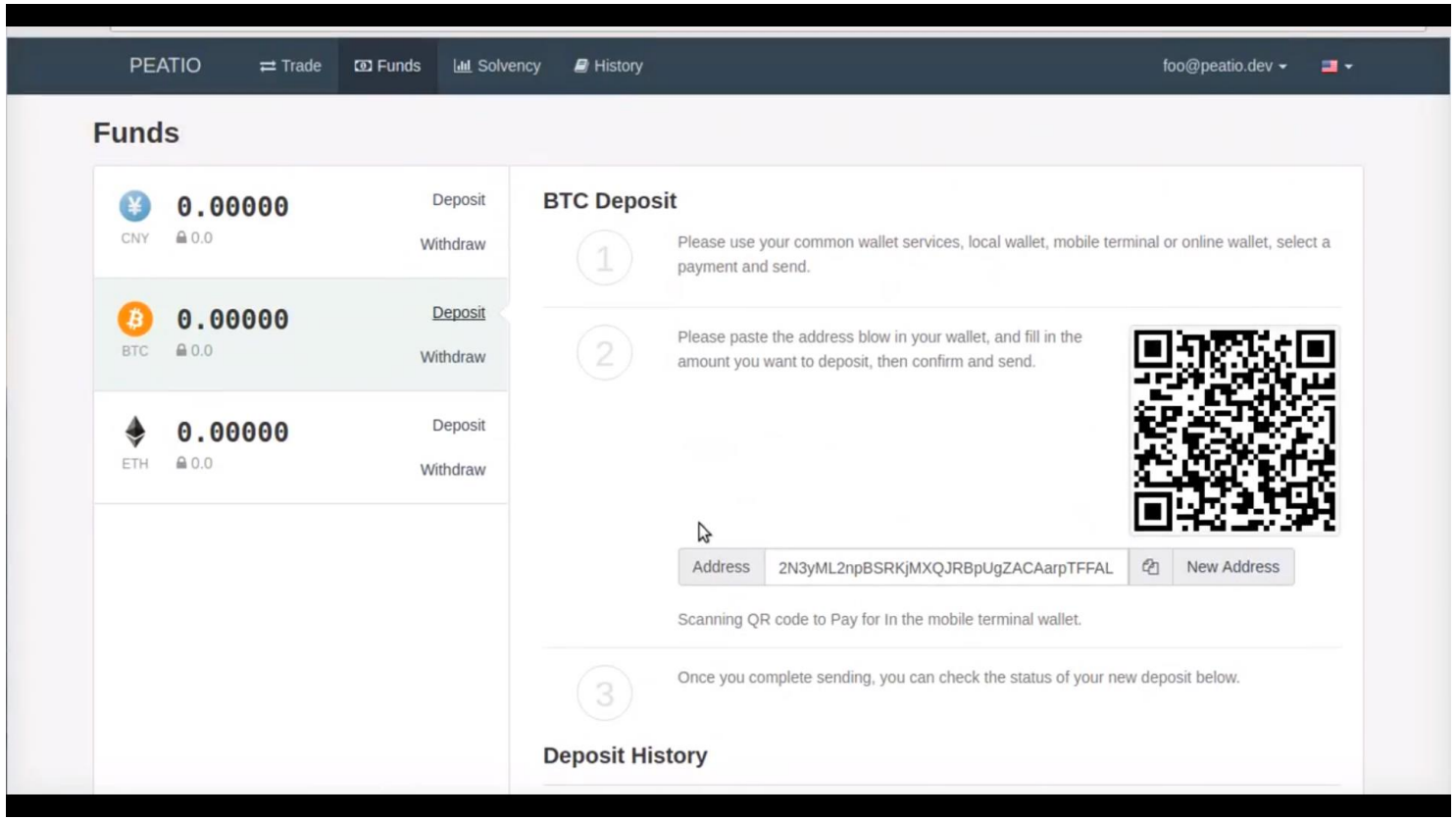
Method

- 계정 생성 후, 이메일 인증과 2단계 인증(휴대폰, 혹은 google Authenticator)를 거치면 블록체인 입금 주소 발급



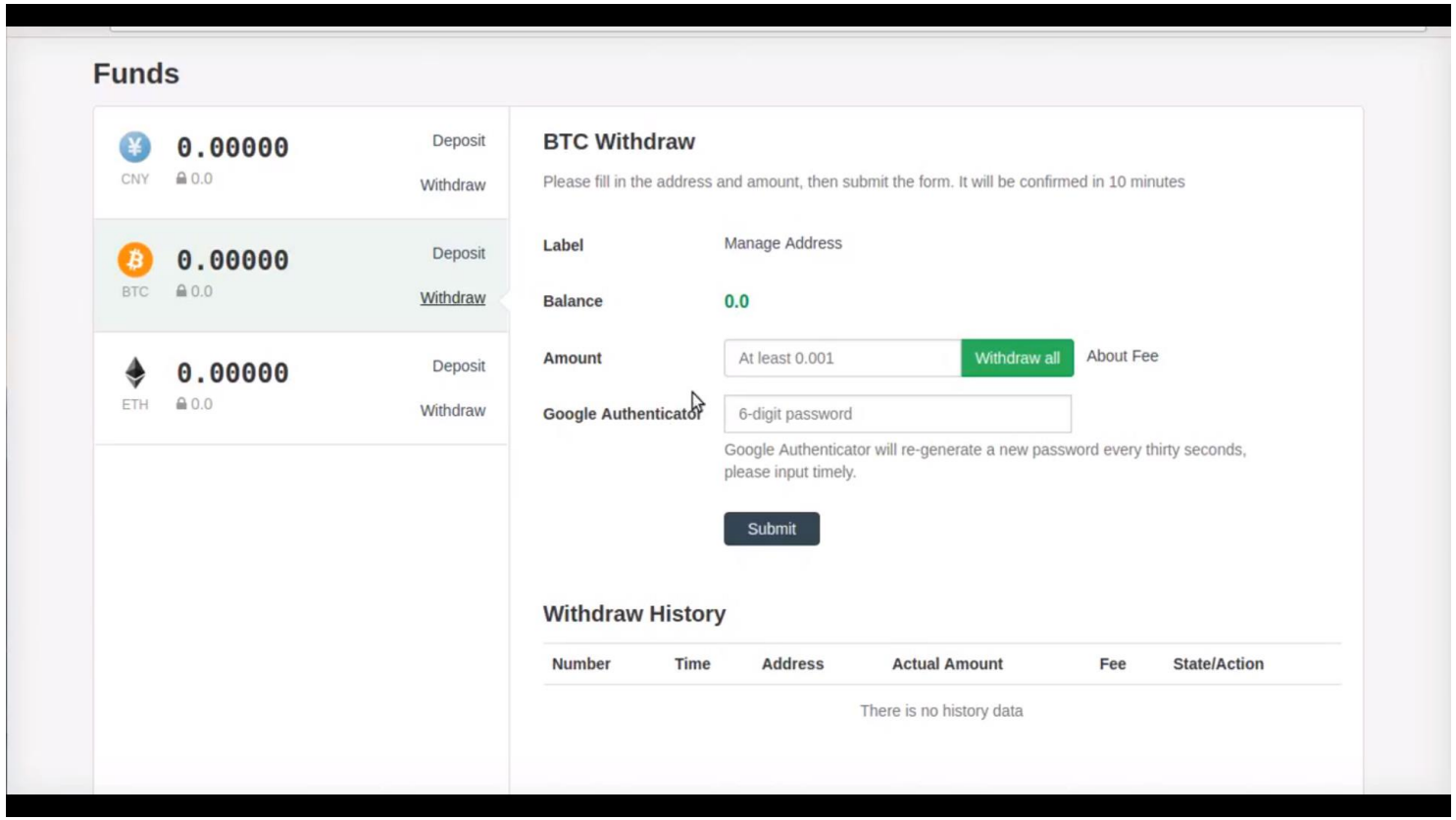
Method

- Deposit



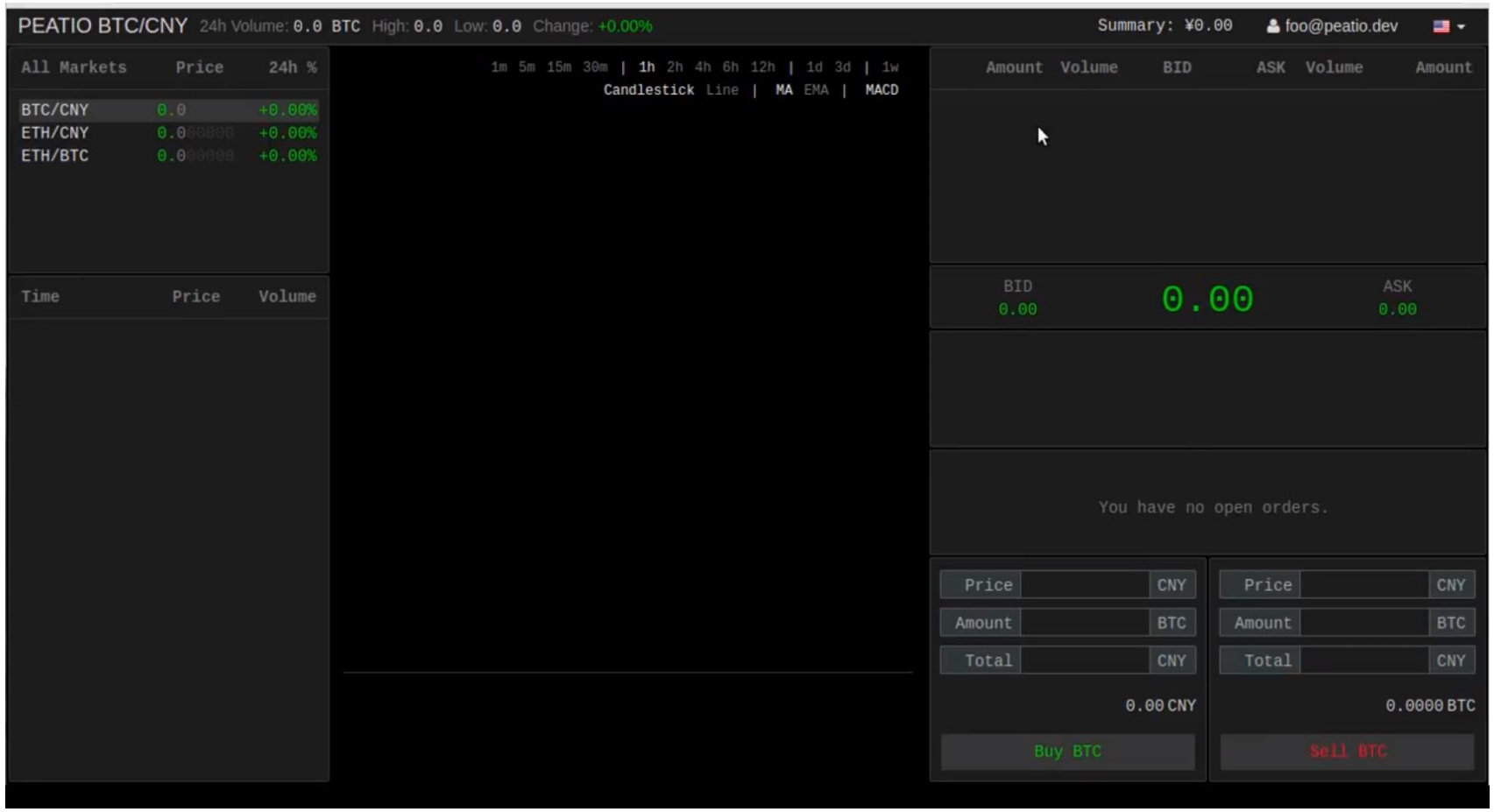
Method

- Withdraw



Method

- 마켓 종류에서 코인 종류 선택 후 거래 가능



References

- Blockstream, *Enabling Blockchain Innovations with Pegged Sidechains*, 2014
  https://blockstream.com/sidechains.pdf
- Ethereum, *BTC Relay Documentation Release 1.0*, 2016
  https://buildmedia.readthedocs.org/media/pdf/btc-relay/latest/btc-relay.pdf
- Martin Holst Swende, *Hacking on BTC Relay*, 2016
  https://swende.se/blog/BTCRelay-Auditing.html#
- Ethereum, *BTC Relay GitHub*, 2018
  https://github.com/ethereum/btcrelay
- Decipher Media, *블록체인 확장성 솔루션 시리즈:: Interchain solution*, 2018-2019
  https://medium.com/decipher-media/archive
- Jake Frankenfield, *Bitcoin Exchange*, 2019-2020
  https://www.investopedia.com/terms/b/bitcoin-exchange.asp