퍼블릭 블록체인과 프라이버시

정순형
Onther Inc.

2019. 10. 15.

온더(Onther Inc.) 소개 - 이더리움 확장성 해결하는 R&D 스타트업

...오늘 이더리움 재단은 한국 커뮤니티에 감사의 마음을 전하고자 소정의 지원(Grants)을 하고자 합니다. **첫째로, 플라즈마EVM을 만들어낸 온더(Onther Inc.)에게 감사의 말을 전합니다. 온더는 플라즈마EVM을 통해 이더리움의 확장성 솔루션에 대하여 매우 흥미롭고 새로운 아이디어를 제시했고, 내용은 매우 인상적이었습니다. 온더가 제시한 디자인은 재단의 플라즈마 연구자들도 생각해내지 못한 것이었습니다. 이러한 결과가 한국 커뮤니티가 만들어낸 위와 같은 사례중 한가지라고 생각합니다...**

-비탈릭 부테린, 이드콘 코리아 축사 中

Tokamak Network

영지식 증명을 활용한 프라이버시 토큰(zk-ERC20) 구현

주의 : 코드가 많음

철학자(정순형)

Jun 7 · 23 min read

Special Thanks to Carl Park(4000d)

우리가 사토시 나카모토를 찾지 못하는 이유?

Tokamak Network

우리가 사토시 나카모토를 찾지 못하는 이유?

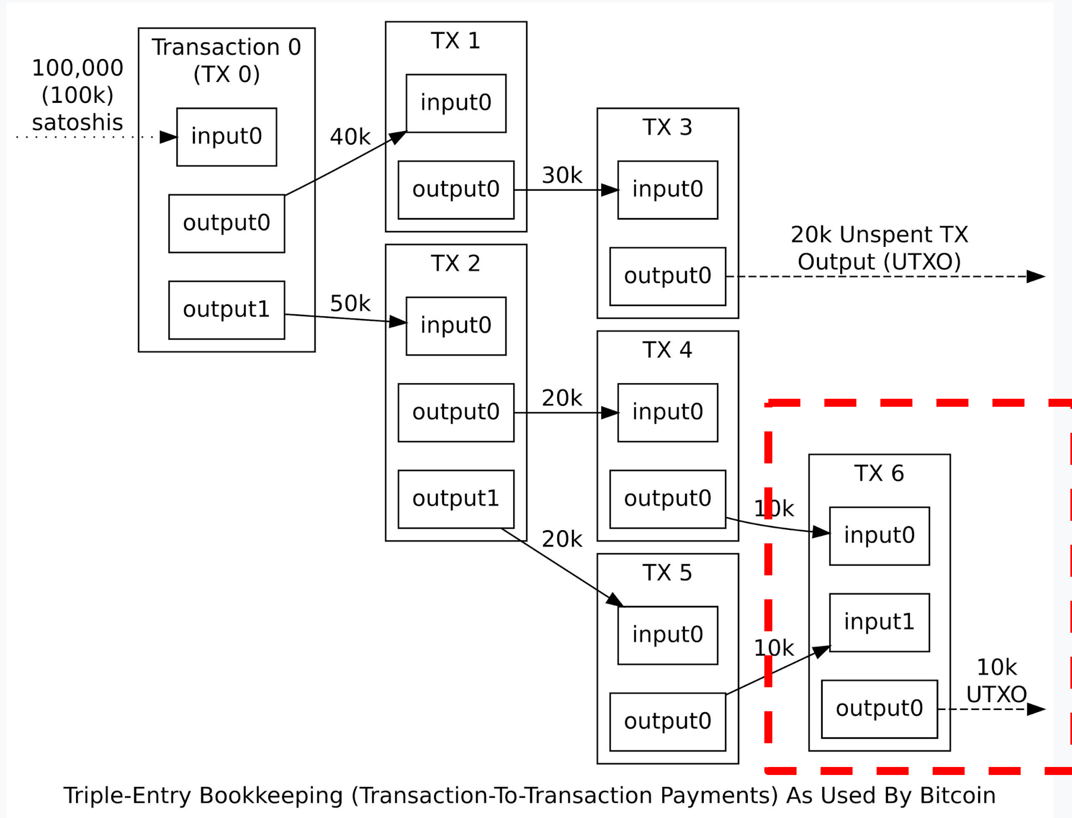
"사토시 나카모토는 **(KYC를 하는)거래소**를 이용하지 않았기 때문"

<#1신분증>

<#2본인이 신분증 들고 있는 사진>

Tokamak Network

그가 거래소를 이용한다면 누군지 쉽게 찾습니다



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

거래소를 이용하는 순간, 이와 관련된 모든 거래 내역 및 활동을 추적할 수 있다.

Tokamak Network

GDPR과 정보 주권 그리고 퍼블릭 블록체인

- EU거주자의 개인정보 보호를 강화하고 표준화하기 위해 제정된 법 (2018.5.24.발표)
- 규제대상
  - 개인정보의 수집, 저장, 전송, 사용 등 개인정보 저장 및 처리에 관한 규제
  - 목적 : <u>데이터 주체에게 더 많은 권한, 통제권</u>
- 개인정보란?
  - 식별되었거나, 식별가능한 자연인(정보주체)과 관련된 모든 정보
  - <u>비식별화</u> : 개인정보와 정보주체의 연결을 없애는 경우
    - 만약 거래소 혹은 서비스가 KYC를 할 경우?
    - 거래소는 해당 <u>KYC정보를 바탕으로 정보주체의 과거 및 향후의 모든 거래기록을 추적/식별</u>할 수 있음 → **비식별화 조치 위배**

Tokamak Network

GDPR과 정보 주권 그리고 퍼블릭 블록체인

- EU거주자의 개인정보 보호를 강화하고 표준화하기 위해 제정된 법 (2018.5.24.발표)

We Need <u>Private Transaction</u> in Public Blockchain!

  - 식별되었거나, 식별가능한 자연인(정보주체)과 관련된 모든 정보
  - <u>비식별화</u> : 개인정보와 정보주체의 연결을 없애는 경우
    - 만약 거래소 혹은 서비스가 KYC를 할 경우?
    - 거래소는 해당 <u>KYC정보를 바탕으로 정보주체의 과거 및 향후의 모든 거래기록을 추적/식별</u>할 수 있음 → **비식별화 조치 위배**

Tokamak Network

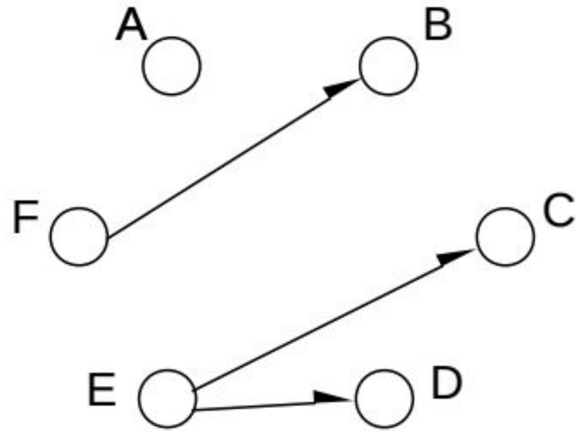
ZKDai Background

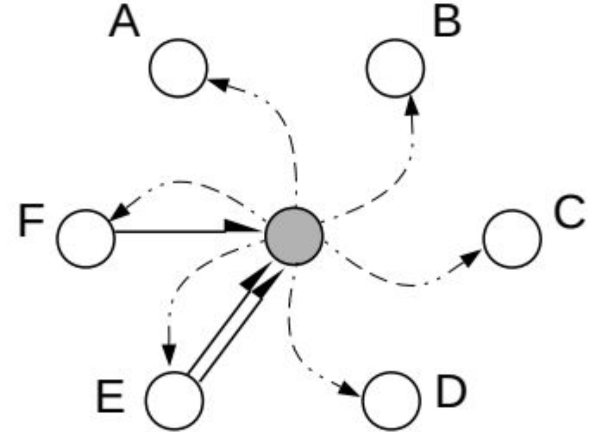
- 5W 1H
  - Why : ERC20 Private Transaction
  - What : Secret recipients and hidden value transaction using zkSnarks
  - Who : Arpit Agarwal, now work at Matic.network(Plasma Team)
  - When : Dec 7 - 9 , 2018
  - Where : EthSingapore Hackaton
  - How : (본문)

- 사전 지식(zk-snarks + MakerDAO + ERC20)
  - https://media.consensys.net/introduction-to-zksnarks-with-examples-3283b554fc3b
  - https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649
  + https://docs.google.com/presentation/d/1EAolw_oxb41oaMpM1OZTNCetN1E834aMWnZt9hS5Ah4/edit?usp=sharing

Tokamak Network

IDEA : Re-constructing Transaction Topology





(시크릿) 노트

- **Note** = (**pk**, v);

  pk := publicKey of the note owner

  v := value of note in dai

- A Note is on the chain as [Hash(**Note**), Encrypt(**Note**)]

  encrypted with the **pk** of the note owner → 본인 노트를 찾기 위해서 필요함

  Let's call Hash(Note) as the **secret note**

- To spend a note *zk-prove* that you own

  1. secretKey (sk) that corresponds to pk
  2. value of the note

- Generate 2 new notes

  N1 = (receiverPk, v') and N2 = (**pk**, v - v')

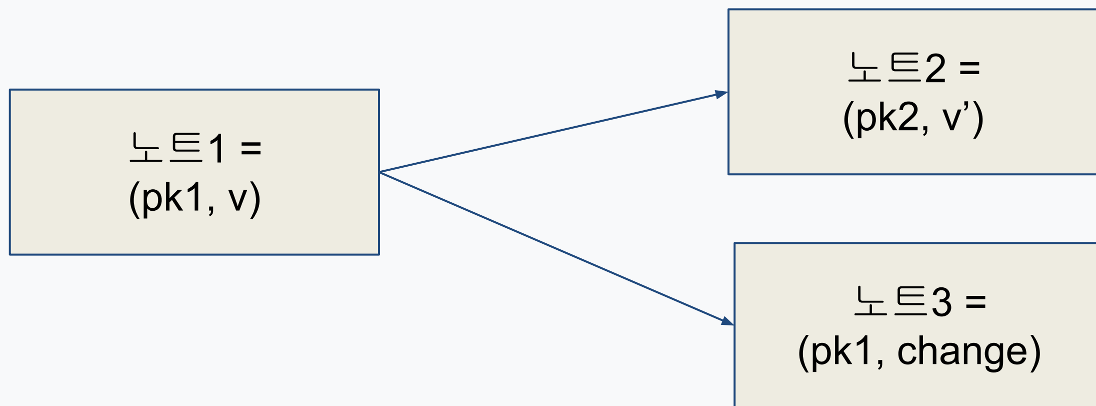
  send (zkProof, encrypt(receiverPk, n1), encrypt(pk, n2)) as transferNote transaction payload

노트 =
(소유자, 액면가)

Tokamak Network

ZKP는 어떤 연산을 증명하는가? (1)

1. 나는(pk1) 시크릿노트1에 대한 소유권자가 확실해!
2. v = v’ + change도 확실해!
3. 시크릿노트2는 Pk2의 공개키를 통해 만들어졌어!
4. 시크릿노트3은 나(pk1)을 통해 만들어졌어!



Tokamak Network

ZKP는 어떤 연산을 증명하는가? (2)

C(oldNote, newNote1, newNote2, sk, v, receiverPk, v’, change)

OldNote : 예전에 소유하던 노트(해시)

newNote1 : 소유권이 옮겨진(전송된) 노트(해시)

newNote2 : 소유권을 옮기고 난(전송된) 후 남은 노트(해시)

sk : 예전 노트의 개인키

v : 예전 노트의 잔액

receivePk : 전송된 노트 소유자의 공개키

v’ : 전송된 노트의 잔액

change : 예전노트 - 전송된노트 = 전송 후 예전노트의 잔액

Tokamak Network

ZKP는 어떤 연산을 증명하는가? (3)

```
C(oldNote, newNote1, newNote2, sk, v, receiverPk, v’, change) {
   pk = computePublicKeyfromSecret(sk)
   oldNote == sha256(pk, v)
   v == v’ + change
   newNote1 == sha256(receiverPk, v’)
   newNote2 == sha256(pk, change)
   return 1
}
```

oldNote, newNote1, newNote2 는 퍼블릭 파라미터(알려져도 소유자 및 잔액이 들통나지 않음)

→ why? **Note = hash(pk, v)** 이기 때문

Tokamak Network

ZK Proof의 마법 : 5개의 숫자

[

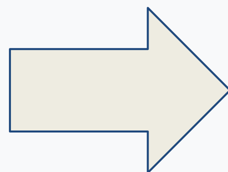
179426333,
181327921,
198492781,
533001793,
334216237

]

Tokamak Network

ZK Proof의 마법

[
179426333,
181327921,
198492781,
533001793,
334216237
]



나는(pk1) 시크릿노트1에 대한 소유권자가 확실하고

5(원래 노트 액면가)= 3(노트1의 액면가)+ 2(노트2의 액면가)이고

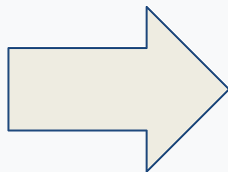
시크릿노트2는 Pk2의 공개키를 통해 만들어졌고

시크릿노트3은 나(pk1)을 통해 만들어졌어!

Tokamak Network

ZK Proof의 마법

[
179426333,
181327921,
198492781,
533001793,
334216237
]



노트1 =
(철수, 5)

노트2 =
(영희, 3)

노트3 =
(철수, 2)

zk-S(**Succinct**)nark

Tokamak Network

스마트 컨트랙트는 어떻게 노트를 기록하는가?

enum State {Invalid, Created, Spent} //노드의 상태

mapping(bytes32 => State) public notes; //노트 해시와 노트 상태를 mapping

string[ ] public allNotes; //만들어진 모든 노트의 encrypt한 값(잔액 확인용)

bytes32[ ] public allHashedNotes; //만들어진 모든 노트의 해시값

시크릿노트(노드해시)와 그 사용여부(State)
encrypt(시크릿노트)

노트1 Created

Tokamak Network

스마트 컨트랙트는 어떻게 노트를 기록하는가?

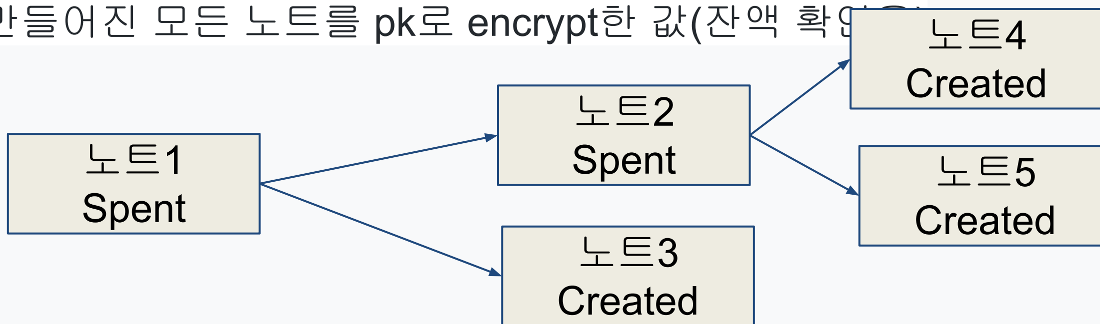
```
enum State {Invalid, Created, Spent} //노드의 상태

mapping(bytes32 => State) public notes; //mapping of hash of the note to state

string[] public allNotes; //만들어진 모든 노트의 해시

bytes32[] public allHashedNotes; //만들어진 모든 노트를 pk로 encrypt한 값(잔액 확
```

시크릿노트(노드해시)와 그 사용여부(State)
encrypt(시크릿노트)



Tokamak Network

스마트 컨트랙트는 노트의 소유권을 어떻게 바꾸나?

1. 연산(노트2개 만들어버리는 연산)에 대한 증거(Proof)를 체크하고
2. 노트1을 찾아서 “사용됨"으로 바꾸고
3. 노트2,3을 만든후 “미사용됨”으로 초기화한다

→ 컨트랙트에 기록된 이 내용만 가지고는 누가 누구에게 보냈는지 알수 없게됨

Tokamak Network

스마트 컨트랙트는 노트의 소유권을 어떻게 바꾸나?

```
75    function transferNote(
76      uint[2] a,
77      uint[2] a_p,
78      uint[2][2] b,
79      uint[2] b_p,
80      uint[2] c,
81      uint[2] c_p,
82      uint[2] h,
83      uint[2] k,
84      uint[7] input,
85      string encryptedNote1,
86      string encryptedNote2
87    ) {
88      require(
89        verifyTx(a, a_p, b, b_p, c, c_p, h, k, input),
90        'Invalid zk proof'
91      );
92
93      bytes32 spendingNote = calcNoteHash(input[0], input[1]);
94      // emit debug(spendingNote, bytes32(0));
95      require(
96        notes[spendingNote] == State.Created,
97        'spendingNote doesnt exist'
98      );
99
100     notes[spendingNote] = State.Spent;
101     bytes32 newNote1 = calcNoteHash(input[2], input[3]);
102     createNote(newNote1, encryptedNote1);
103     bytes32 newNote2 = calcNoteHash(input[4], input[5]);
104     createNote(newNote2, encryptedNote2);
105   }
```

Tokamak Network

스마트 컨트랙트는 노트의 소유권을 어떻게 바꾸나?

```
75    function transferNote(
76      uint[2] a,
77      uint[2] a_p,
78      uint[2][2] b,
79      uint[2] b_p,
80      uint[2] c,
81      uint[2] c_p,
82      uint[2] h,
83      uint[2] k,
84      uint[7] input,
85      string encryptedNote1,
86      string encryptedNote2
87    ) {
88      require(
89        verifyTx(a, a_p, b, b_p, c, c_p, h, k, input),
90        'Invalid zk proof'
91      );
92
93      bytes32 spendingNote = calcNoteHash(input[0], input[1]);
94      // emit debug(spendingNote, bytes32(0));
95      require(
96        notes[spendingNote] == State.Created,
97        'spendingNote doesnt exist'
98      );
99
100     notes[spendingNote] = State.Spent;
101     bytes32 newNote1 = calcNoteHash(input[2], input[3]);
102     createNote(newNote1, encryptedNote1);
103     bytes32 newNote2 = calcNoteHash(input[4], input[5]);
104     createNote(newNote2, encryptedNote2);
105   }
```

왜 a, a_p로 인풋값을 2개로 나눠서 적지..?

→ zokrates랭귀지 함수 파라미터가 254비트밖에 지원 못하는 바람에 인풋값을 “128비트 2개씩” 사용

Tokamak Network

Problem - Verify Too Costly

```
75    function transferNote(
76      uint[2] a,
77      uint[2] a_p,
78      uint[2][2] b,
79      uint[2] b_p,
80      uint[2] c,
81      uint[2] c_p,
82      uint[2] h,
83      uint[2] k,
84      uint[7] input,
85      string encryptedNote1,
86      string encryptedNote2
87    ) {
88      require(
89        verifyTx(a, a_p, b, b_p, c, c_p, h, k, input),
90        'Invalid zk proof'
91      );
92
93      bytes32 spendingNote = calcNoteHash(input[0], input[1]);
94      // emit debug(spendingNote, bytes32(0));
95      require(
96        notes[spendingNote] == State.Created,
97        'spendingNote doesnt exist'
98      );
99
100     notes[spendingNote] = State.Spent;
101     bytes32 newNote1 = calcNoteHash(input[2], input[3]);
102     createNote(newNote1, encryptedNote1);
103     bytes32 newNote2 = calcNoteHash(input[4], input[5]);
104     createNote(newNote2, encryptedNote2);
105   }
```

700,000,000 GAS !!

Tokamak Network

Current Block Gas Limit - 800,000,000

Block	Age	Txn	Uncles	Miner	Gas Used	Gas Limit	Avg.Gas Price	Reward
7927260	41 secs ago	37	0	Ethermine	3,254,762 (40.68%)	8,000,000	5.98 Gwei	2.01947 Ether
7927259	43 secs ago	65	0	Nanopool	7,951,924 (99.35%)	8,003,877	2.13 Gwei	2.01692 Ether
7927258	45 secs ago	10	0	F2Pool 2	1,619,768 (20.22%)	8,011,664	5.71 Gwei	2.00925 Ether
7927257	50 secs ago	152	0	Spark Pool	7,983,441 (99.75%)	8,003,849	5.27 Gwei	2.04211 Ether
7927256	1 min ago	80	0	0x11905bd0863ba5…	7,940,730 (99.26%)	8,000,029	1.04 Gwei	2.00823 Ether
7927255	1 min ago	53	0	Spark Pool	7,634,789 (95.37%)	8,005,618	3.14 Gwei	2.02394 Ether
7927254	1 min ago	148	0	Spark Pool	7,982,670 (99.76%)	8,001,738	1.51 Gwei	2.01208 Ether
7927253	1 min ago	24	0	F2Pool 2	1,192,354 (14.89%)	8,007,811	10.35 Gwei	2.01233 Ether
7927252	1 min ago	102	0	Ethermine	7,996,266 (99.95%)	8,000,000	8.53 Gwei	2.06817 Ether
7927251	1 min ago	59	0	MiningPoolHub	7,979,201 (99.74%)	8,000,029	1.26 Gwei	2.01001 Ether
7927250	2 mins ago	144	0	Ethermine	7,904,825 (98.81%)	8,000,000	8.51 Gwei	2.06728 Ether

Tokamak Network

Scalability::
we need decentralized and turing complete Layer 2

Block # 7927260

Secret Note Transfer( )

tx1

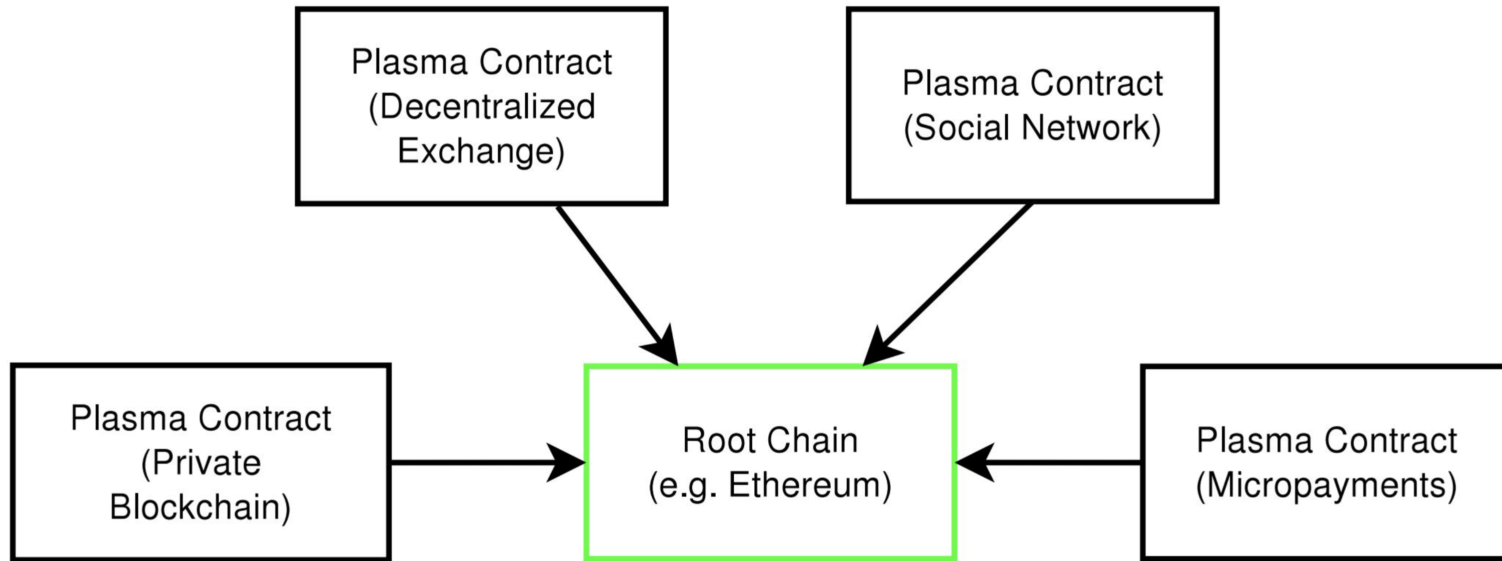
tx1

tx1

tx1

Tokamak Network

Scalability:: we need decentralized and turing complete Layer 2



토카막 플라즈마에 응용

Requestable + Secret Note + Dex

(프라이버시 거래소)

Ethereum Mainnet

Secret
Requestale
ERC20
(Token)

Enter

Exit

Plasma

Secret
Requestable
ERC20
(Note)

Trade

Secret
Note
Decentralized
Exchange

이더리움 메인체인

플라즈마 체인

Tokamak Network

문제 및 개선

1. 구현된 zkDai의 문제
   a. 검증 함수(verifyTx) 가 가스를 상당히 많이 소모(약 7백만)
      i. 기타 대안(zk-stark, bullet proof, sonic 등은 더 많은 검증비용 소모)
   b. 노트 소유자 잔액 조회 문제 encrypt(hash(pk, v))
      i. 내가 소유한 노트의 정확한 잔액을 알아야만 트랜잭션을 날릴 수 있다.
   c. 노트를 배열 처리해서 노트가 많아지면 매우 느림 → 배열 길이 무한..?
   d. 노트를 쓰려면 항상 2개로 쪼갠다.(튜링 불완전성)
      i. 모든 금액을 보내는 경우 노트 하나는 잔액이 0이됨
      ii. 여러명에게 보내고 싶거나, 멀티 시그를 하고 싶을 때는..?
   e. 충분한 노트가 쌓이지 않으면 프라이버시 보장 어렵다

Tokamak Network

문제 및 개선

2. ZKP 구조적 문제
   a. 이니셜 셋업에 toxic waste를 누가 셋업할 것인가? → MPC
   b. witness를 만드는 연산이 클라이언트에게 부담이 적지 않고, 유저는 플라즈마 외에 별도의 클라이언트 혹은 서비스가 필요

3. Dex문제
   a. zokrates랭귀지가 지원하는 라이브러리가 많지 않음.
      → eWasm필요

Tokamak Network

Test & Script

https://github.com/Onther-Tech/zk-ERC20

Demo - Setup Architecture

Zokrates Docker Container

```
/home/zokrates/zk-related
```

~ Repo-directory/zk-related/

```
npm run zokrates:compile &&
npm run zokrates:setup &&
npm run zokrates:export-verifier
```

1. 디렉토리 공유
2. 컴파일, 셋업, verifier 컨트랙트 파일 생성

out
~~proof.json : 연산에 대한 proof~~
verification.key : verifier키
~~witness~~
out.code
proving.key : prover 키
verifier.sol : 베리파이어 컨트랙트
zk-circuit.code : 원본 서킷 코드

Tokamak Network

Demo - Transfer Note Logic and Architecture

Zokrates Docker Container

/home/zokrates/zk-related

```
npm run zokrates:compile &&
npm run zokrates:setup &&
npm run zokrates:export-verifier
```

out
~~proof.json : 연산에 대한 proof~~
verification.key : verifier키
witness
out.code
proving.key : prover 키
verifier.sol : 베리파이어 컨트랙트
zk-circuit.code : 원본 서킷 코드

~ Repo-directory/zk-related/

1.CreateDummy( )

Note # 0
(Accounts[0], 5)

Tokamak Network

Demo - Transfer Note Logic and Architecture

Zokrates Docker Container

~ Repo-directory/zk-related/

`/home/zokrates/zk-related`

```
npm run zokrates:compile &&
npm run zokrates:setup &&
npm run zokrates:export-verifier
```

1.CreateDummy( )

~~Note # 0~~
~~(Accounts[0], 5)~~

Note # 1
(Accounts[1], 3)

Note # 2
(Accounts[0], 2)

out
proof.json : 연산에 대한 proof
verification.key : verifier키
witness
out.code
proving.key : prover 키
verifier.sol : 베리파이어 컨트랙트
zk-circuit.code : 원본 서킷 코드

2. Make Witness → Proof.json 파일 생성

Tokamak Network

Demo - Transfer Note Logic and Architecture

Zokrates Docker Container

```
/home/zokrates/zk-related
```

```
npm run zokrates:compile &&
npm run zokrates:setup &&
npm run zokrates:export-verifier
```

out
proof.json : 연산에 대한 proof
verification.key : verifier키
witness
out.code
proving.key : prover 키
verifier.sol : 베리파이어 컨트랙트
zk-circuit.code : 원본 서킷 코드

~ Repo-directory/zk-related/

1.CreateDummy( )

~~Note # 0 (Accounts[0], 5)~~

Note # 1 (Accounts[1], 3)

Note # 0 (Accounts[0], 2)

2. Make Witness → Proof.json 파일 생성
3. Proof.json의 proof를 이용해 transferNote() 함수를 호출

Tokamak Network

Demo - Transfer Note Logic and Architecture

Zokrates Docker Container

```
/home/zokrates/zk-related

npm run zokrates:compile &&
npm run zokrates:setup &&
npm run zokrates:export-verifier
```

~ Repo-directory/zk-related/

out
proof.json : 연산에 대한 proof
verification.key : verifier키
witness
out.code
proving.key : prover 키
verifier.sol : 베리파이어 컨트랙트
zk-circuit.code : 원본 서킷 코드

1.CreateDummy( )

~~Note # 0 (Accounts[0], 5)~~

Note # 1 (Accounts[1], 3)

Note # 0 (Accounts[0], 2)

2. Make Witness → Proof.json 파일 생성
3. Proof.json의 proof를 이용해 transferNote() 함수를 호출

Tokamak Network

More Layer 2 Infomation?

https://tokamak.network/kr

Tokamak Network

References

- ZkDAI 블로그 : https://medium.com/@atvanguard/zkdai-private-dai-transactions-on-ethereum-using-zk-snarks-9e3ef4676e22
- zkDAI Setup Video : https://www.youtube.com/watch?v=DCnaUYbk75k&feature=youtu.be
- zkdai githuib : https://github.com/atvanguard/ethsingapore-zk-dai
- Zokrates Doc : https://zokrates.github.io/
- 영지식 증명을 활용한 프라이버시 토큰(zk-ERC20) 구현 : https://medium.com/onther-tech/%EC%98%81%EC%A7%80%EC%8B%9D-%EC%A6%9D%EB%AA%85%EC%9D%84-%ED%99%9C%EC%9A%A9%ED%95%9C-%ED%94%84%EB%9D%BC%EC%9D%B4%EB%B2%84%EC%8B%9C-%ED%86%A0%ED%81%B0-zk-erc20-%EA%B5%AC%ED%98%84-14fa69b49418

Tokamak Network