

이더리움 플랫폼의 이해

목차

- 이더리움 Account
- Transaction
- Smart Contract

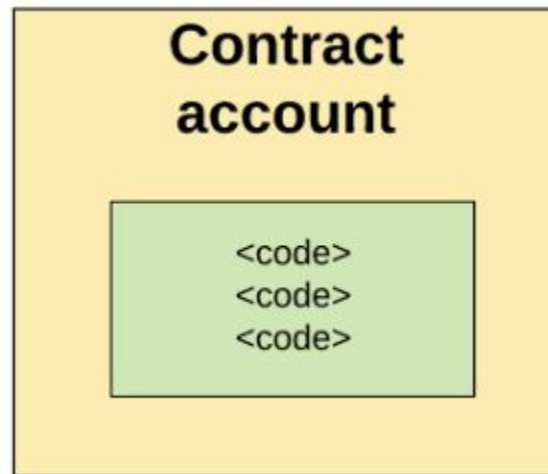
- Account

- 이더리움에서 고유의 상태 값을 관리하는 단위
- 트랜잭션을 만들어 내, 다른 계정의 상태값을 바꿀 수 있는 주체
- 트랜잭션에 의해 상태 값이 바뀔 수 있는 대상



이더리움 Account

- 이더리움 계정의 종류
 - EOA - Externally Owned Account
 - CA - Contract Account



EOA (Externally Owned Account)

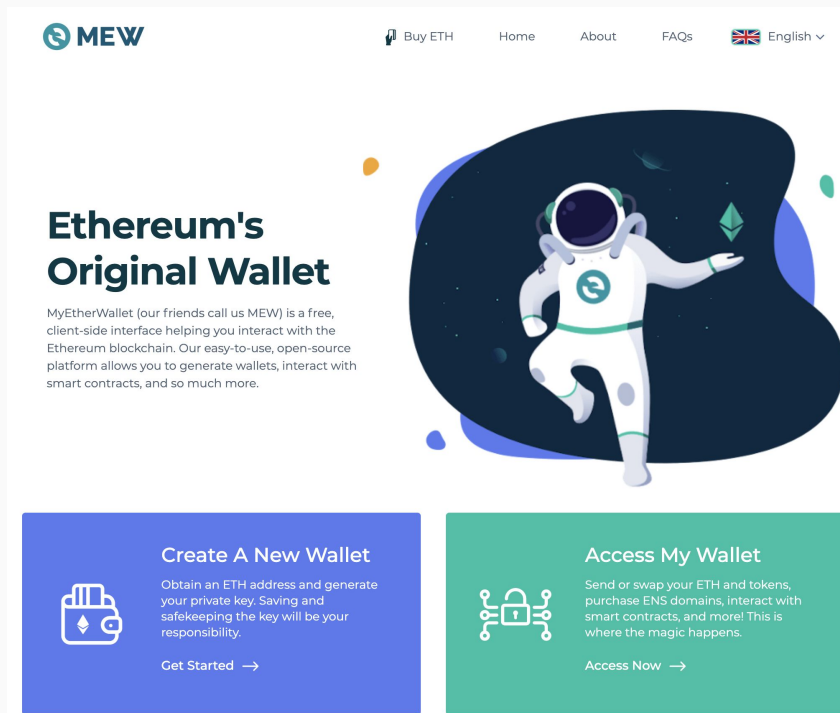
- **Externally Owned Account**

- 이더리움 밖에서 접근가능한 계정
- 이더리움 바깥에서, (지갑 프로그램, 웹 서버 등)에서 이더리움 네트워크로 트랜잭션을 전파하기 위해선 이더리움 계정이 필요
- 보내려는 트랜잭션에 개인키로 서명하여, 트랜잭션 생성자 임을 증명.



EOA (Externally Owned Account)

- EOA 만들기
 - My Ether Wallet 접속 후 Create A New Wallet 클릭
<https://www.myetherwallet.com/>



EOA (Externally Owned Account)

- EOA 만들기
 - By Keystore File 탭 선택
 - Password 설정


Get a New Wallet

Already have a wallet? [Access My Wallet](#)

MEWconnect

By Keystore File

By Mnemonic Phrase




NOT RECOMMENDED

This is not a recommended way to access your wallet. Due to the sensitivity of the information involved, these options should only be used in offline settings by experienced users.

> Read: [Using MEW Offline](#)

Your Password ⓘ

.....



Password strength: **Strong**

Next →

DO NOT FORGET to save your password. You will need this **Password + Keystore File** to unlock your wallet.

EOA (Externally Owned Account)

- EOA 만들기
 - Download Keystore File 버튼 클릭.
 - 적당한 위치에 파일 저장

Get a New Wallet

Already have a wallet? [Access My Wallet](#)

By Keystore File

Save My Keystore File



Don't Lose It

Be careful, it can not be recovered if you lose it.



Don't Share It

Your funds will be stolen if you use this file on a malicious phishing site.



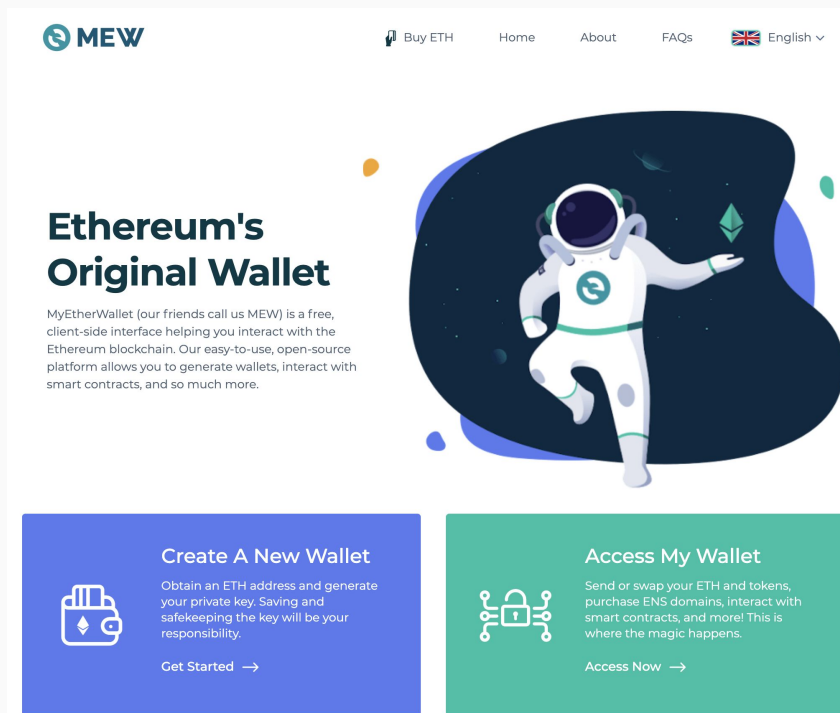
Make a Backup

Secure it like the millions of dollars it may one day be worth.

Download Keystore File

EOA (Externally Owned Account)

- EOA 접근하기
 - My Ether Wallet 접속 후 Access My Wallet 클릭
<https://www.myetherwallet.com/>



EOA (Externally Owned Account)

- EOA 접근하기
 - Software 클릭

Access My Wallet

Do not have a wallet? [Create A New Wallet](#)



MEWconnect

Use MEWconnect to
access my wallet



Hardware

Ledger wallet, FINNEY,
Trezor, BitBox, Secalot,
KeepKey



MetaMask

Use the MetaMask
extension to access my
wallet



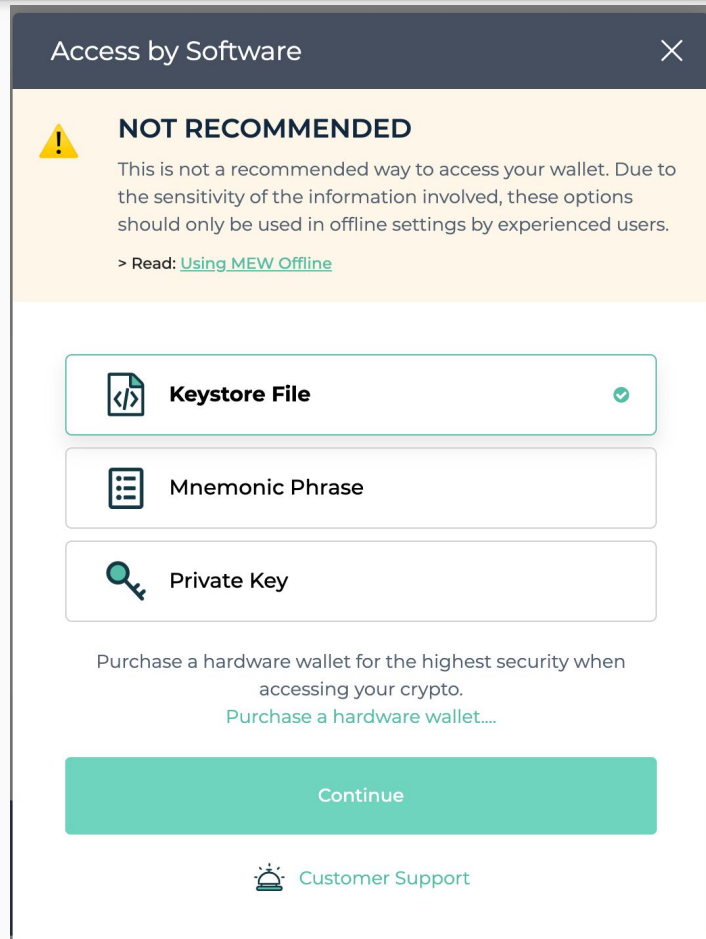
Software

Keystore file, Private key,
Mnemonic phrase

Not recommended

EOA (Externally Owned Account)

- EOA 접근하기
 - Keystore File 선택 후 Continue 클릭
 - 다운 받은 keystore file 선택 (UTC-2019--.....)



EOA (Externally Owned Account)

- EOA 접근하기
 - 설정했던 Password 입력
 - Access Wallet 클릭

Password

!

NOT RECOMMENDED


This is not a recommended way to access your wallet. Due to the sensitivity of the information involved, these options should only be used in offline settings by experienced users.



> Read: [Using MEW Offline](#)


Enter password


Access Wallet


EOA (Externally Owned Account)





[Buy ETH](#) [Transaction History](#) [FAQs](#) [English](#)  


 Dashboard

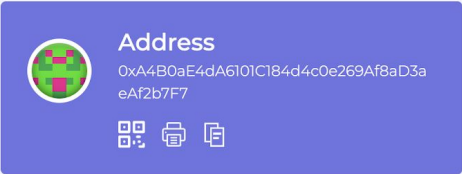
 Send


 Swap

 Dapps

 Contract




 Message

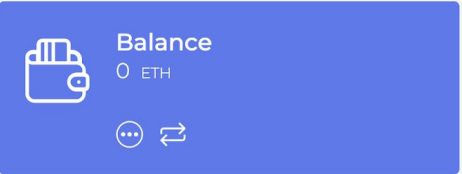





Address

0xA4B0aE4dA6101C184d4c0e269Af8aD3aeAf2b7F7



  







Balance

0 ETH



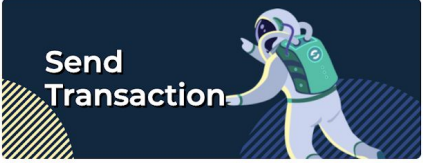


Network


myetherwallet.com(ETH)
Last block# : 8788347

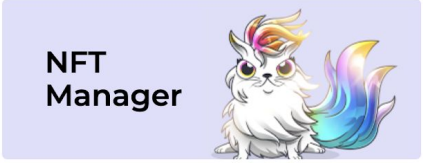
Change

Actions




Send Transaction





NFT Manager





Swap



More

MEW has partnered with Bity, Kyber Network, Changelly, and Simplex to allow users to swap fiat to crypto, ETH and BTC, ETH and ERC-20.



1 ETH / 0.0210 BTC



1 ETH / 155.1100 EUR



1 ETH / 975.5816 KNC

1 BAT / 0.0013 ETH

1 ETH / 173.0167 DAI

Tokens

+ Custom Tokens

FEN

ID

FZC

PWR

MGT

PSM

TMED

0

0

0

0

0

0

0

✖

✖

✖

✖

✖


✖

✖

Dapps

View All

Follow us on Twitter



EOA (Externally Owned Account)

- Keystore File

- 생성된 개인키는 등록한 Password를 통해 암호화 되어 저장

```
1  {
2    "version": 3,
3    "id": "ca24036e-87f9-4e6e-a45d-95355fc1a08c",
4    "address": "a24f7bc7e0f42ea82d0398474fd65d8207f69d9b",
5    "crypto": {
6      "ciphertext": "7577dd21e3a650c34cc0a16e8e4d58679aff626d74dcb9d3d1c2abb650138dc8",
7      "cipherparams": {
8        "iv": "d09c46cfd0dbd0c6056a795c3512634a"
9      },
10     "cipher": "aes-128-ctr",
11     "kdf": "scrypt",
12     "kdfparams": {
13       "dklen": 32,
14       "salt": "f6454fd174d2cb1d4087dc5aa771b5dc5c4ab08929fed88fae91b0529ffad5e3",
15       "n": 8192,
16       "r": 8,
17       "p": 1
18     },
19     "mac": "87e36d64896cb55fb517a57b53470cd630dae5dee7598decc38daa66606d4a07"
20   }
21 }
```

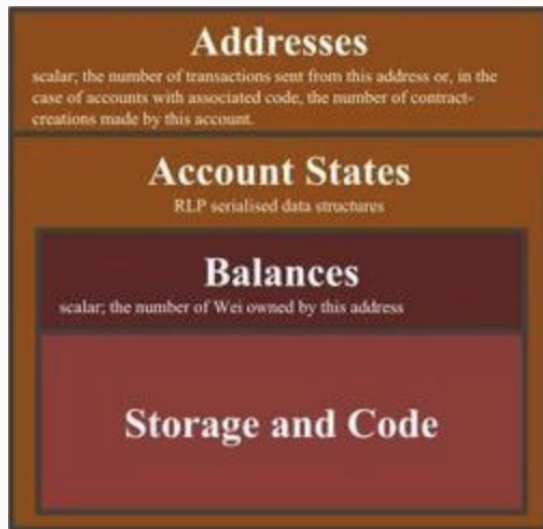
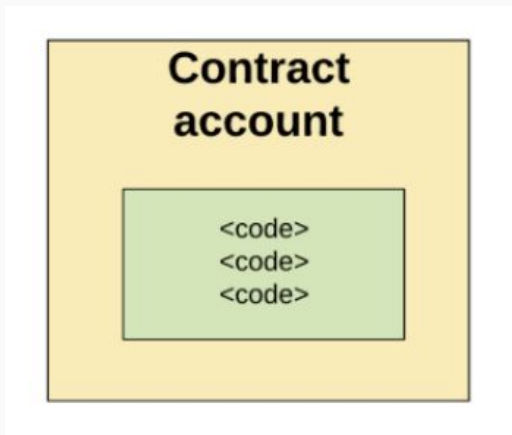
EOA (Externally Owned Account)

- EOA와 일반적인 웹 서비스 계정과의 차이
 - 일반적인 웹서비스(네이버, 카카오톡)은 계정생성과 동시에 계정 정보가 서버에 등록
 - EOA는 생성이 되었다고 해서, 이더리움네트워크에 등록되는것은 아님.
-> 상태 정보 부재
 - 이더리움 네트워크에 연결되어 있지 않더라도, 언제든지 EOA 생성이 가능
이더리움 네트워크와 연결되지 않은 Cold Wallet이 가능한 이유
- EOA 상태의 등록
 - 생성된 계정 상태에 변화가 발생할 때, 계정 정보가 네트워크에 등록됨
 - 상태변화가 생기는 경우 -> 다른 계정으로 부터 이더리움을 받는 경우
또는 EOA 에서 트랜잭션을 발생시키는 경우

CA (Contract Account)

- Contract Account

- 이더리움 안에서 생성되는 계정
- EOA와는 달리 개인키가 존재하지 않음
- 컨트랙트가 생성될 때, 계정이 생성되며 동시에 이더리움 네트워크에 상태가 등록.

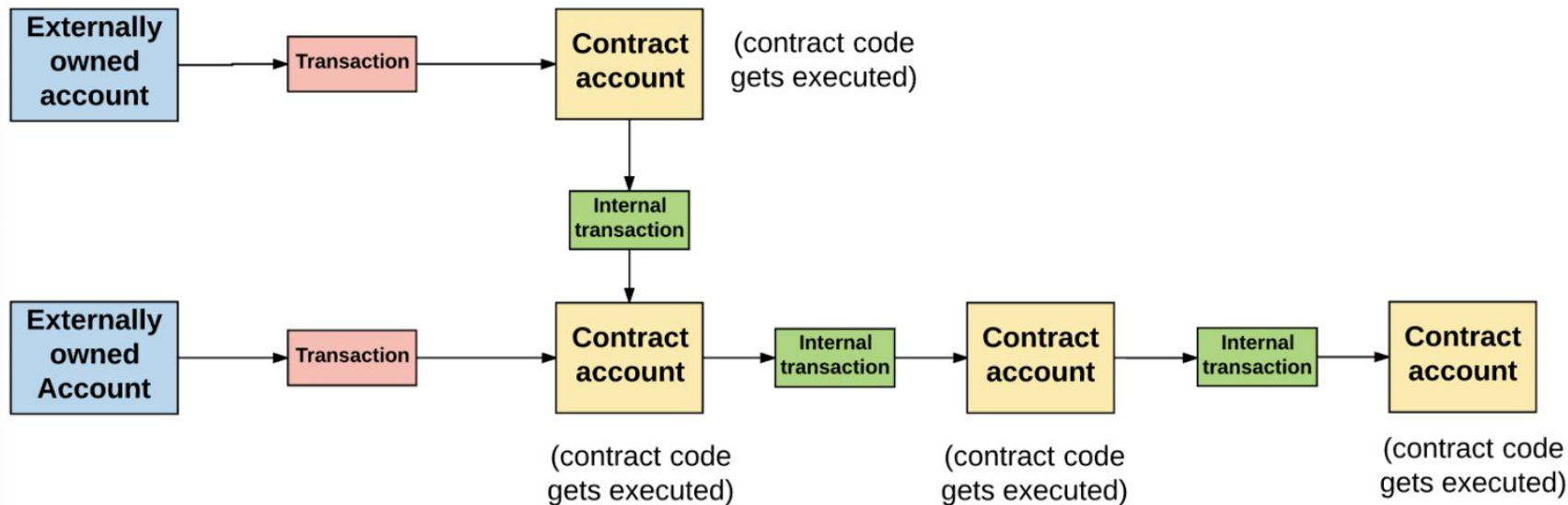


Storage/Code 영역에
컨트랙트의 코드가
올라간다.

CA (Contract Account)

- Contract Account

- EOA와는 달리 개인키가 존재하지 않기 때문에, 외부에서 트랜잭션을 만들어 낼 수 없음
- CA가 트랜잭션을 생성하기 위해선, 반드시 EOA의 트랜잭션이 필요함



Transaction

데이터베이스 트랜잭션(Database Transaction)은 [데이터베이스 관리 시스템](#) 또는 유사한 시스템에서 상호작용의 단위이다. 여기서 유사한 시스템이란 트랜잭션이 성공과 실패가 분명하고 상호 독립적이며, 일관되고 믿을 수 있는 시스템을 의미한다.

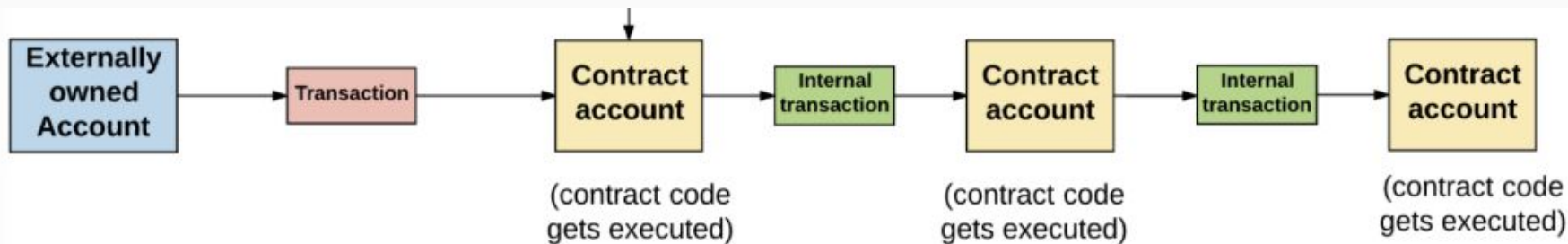
이론적으로 데이터베이스 시스템은 각각의 트랜잭션에 대해 원자성(**Atomicity**), 일관성(**Consistency**), 고립성(**Isolation**), 영구성(**Durability**)을 보장한다. 이 성질을 첫글자를 따 [ACID](#)라 부른다. 그러나, 실제로는 성능향상을 위해 이런 특성들이 종종 완화되곤 한다.

어떤 시스템들에서는 트랜잭션들은 논리적 작업 단위(**LUW, Logical Units of Work**)로 불린다

- 원자성
 - 트랜잭션과 관련된 작업들이 부분적으로 실행되다가 중단되지 않는 것을 보장
 - ex) A 가 B에게 송금을 하는 트랜잭션
 - A의 계좌 잔액조회
 - A의 계좌 잔액에서 100원을 차감
 - B의 계좌 잔액조회
 - B의 계좌 잔액에 100원을 추가
 - B 계좌 조회 단계에서 실패할 경우
 - i. 2번째 단계까지 실행후 저장?
 - ii. 전체 트랜잭션을 실패로 처리

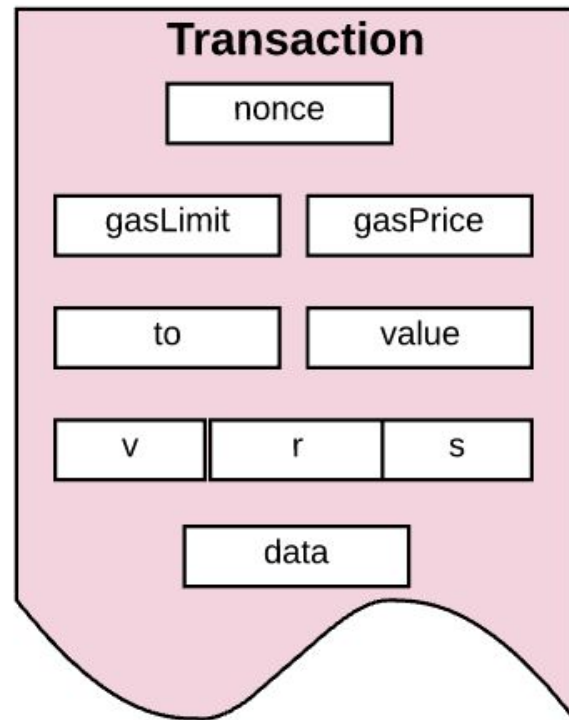
QUIZ

Internal Transaction이 모두 정상적으로 종료되고, 다시 Transaction 으로 회귀한 시점에서, 마지막 처리 과정에서 오류가 발생하면 어떻게 될 것인가?



Transaction

- 이더리움 트랜잭션 구조
 - **nonce** : transaction Sender의 전체 카운트
 - **gasLimit** : 최대 지불용의가 있는 가스의 갯수
 - **gasPrice** : 1가스당 가격
 - **to**: 트랜잭션을 받을 주소
 - **value**: 보내려는 Eth의 수량
 - **v,r,s** : 트랜잭션 사인 정보
 - **data**: 트랜잭션에 실어보낼 데이터
컨트랙트인 경우, 함수 해시 및 Parameter



- Transaction Nonce

- Transaction nonce는 트랜잭션 Sender의 트랜잭션 카운트 값.
- 트랜잭션을 보낼 때 마다 nonce값이 증가
- 동일한 nonce로 2개 이상의 트랜잭션은 존재하지 않음.

- Nonce 충돌

- 동일한 nonce로 2개이상의 트랜잭션이 있는경우, nonce 충돌이라고 얘기함.
- Block Confirm 전이라면, 의도적인 nonce 충돌 유발을 통해, 트랜잭션 덮어쓰기가 가능.
- ex) 가스 수수료 변경, Coin rail 거래소의 해킹 방어

- Gas

- 이더리움은 트랜잭션 처리의 실시간성 확보를 위해 **Gas** 의 개념을 도입
- 트랜잭션이 처리해야할 오퍼레이션을 정량적 수치로 표현한 단위

<https://ethereum.github.io/yellowpaper/paper.pdf>

실시간 **시스템 (Realtime System)**

- 미리 정해진 시간 내에 반드시 정확한 동작을 하도록 만들어진 **시스템**

- GasPrice

- 1 Gas 당 가격으로, 채굴 노드의 보상으로, 전체 오퍼레이션에 대한 비용 산정.
- 채굴 노드는 **Greedy** 한 행동패턴을 보이며, 최대 이익을 얻을 수 있는 트랜잭션을 우선으로 처리.
- nonce 충돌이 나는 경우, **GasPrice bidding**을 통해 하나의 트랜잭션을 선택

Transaction

APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero} .
G_{base}	2	Amount of gas to pay for operations of the set W_{base} .
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low} .
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid} .
G_{high}	10	Amount of gas to pay for operations of the set W_{high} .
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{ssset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the <i>Homestead</i> transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.
$G_{quaddivisor}$	20	The quadratic coefficient of the input sizes of the exponentiation-over-modulo precompiled contract.

- Transaction 처리 절차
 - Raw Transaction : 보내려는 트랜잭션 정보를 생성
 - Transaction Sign : sender의 개인키를 통해 트랜잭션 정보에 서명
 - Broadcast : 서명한 트랜잭션을 메인넷에 전파
 - Miner Accept : 채굴 노드에서 트랜잭션을 블록에 삽입
 - Block Broadcast : 채굴자가 생성한 블록이 메인넷에 전파

- Raw Transaction
 - 트랜잭션 필드의 값을 목적에 맞게 채워넣음

```
var rawTx = {  
  nonce: web3.utils.toHex(nonce),  
  from: ownerAddress,  
  gasLimit: web3.utils.toHex(200000),  
  gasPrice: web3.utils.toHex(10e9), // 10 gwei  
  to: contractAddress,  
  value: 0,  
  data: token.methods.burn(amount).encodeABI()  
};
```

- Transaction Sign
 - 작성한 트랜잭션을 개인키를 통해 서명

```
var tx = new ethereumjstx(rawTx, {'chain': 'ropsten'});  
tx.sign(privateKey_t);
```

Transaction

- Transaction Broadcast
 - 서명한 트랜잭션을 이더리움 네트워크에 전파

```
var serializedTx = tx.serialize();  
web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
```

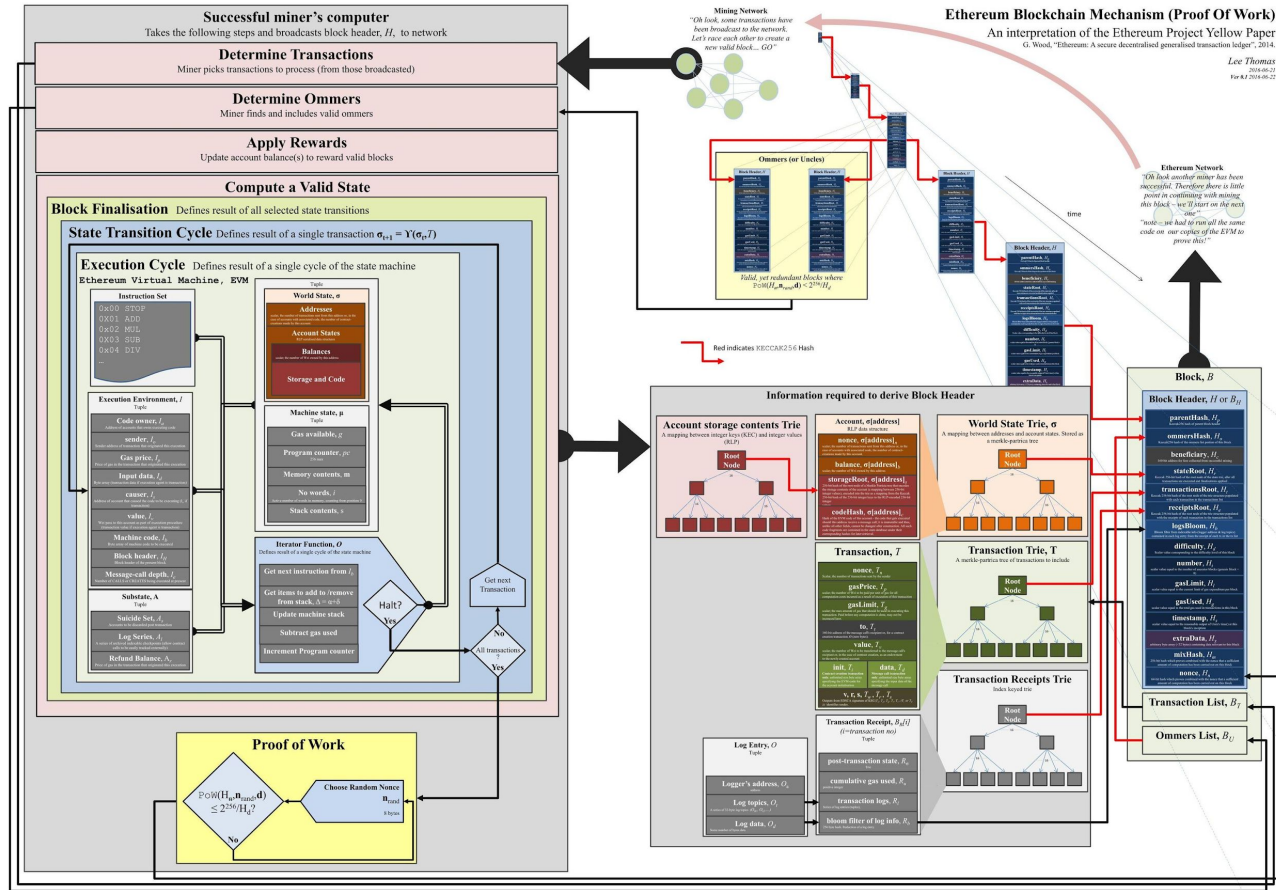
Transaction

```
var rawTx = {
  nonce: web3.utils.toHex(nonce),
  from: ownerAddress,
  gasLimit: web3.utils.toHex(200000),
  gasPrice: web3.utils.toHex(10e9), // 10 gwei
  to: contractAddress,
  value: 0,
  data: token.methods.burn(amount).encodeABI()
};

var tx = new ethereumjsTx(rawTx, {'chain': 'ropsten'});
tx.sign(privateKey_t);

var serializedTx = tx.serialize();
web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
```

Transaction



- 스마트 컨트랙트란?

- 신뢰할 수 없는 환경에서, 신뢰할 만한 데이터 처리를 해주기 위한 소프트웨어
- 블록체인 플랫폼 위에서 동작하는 소프트웨어

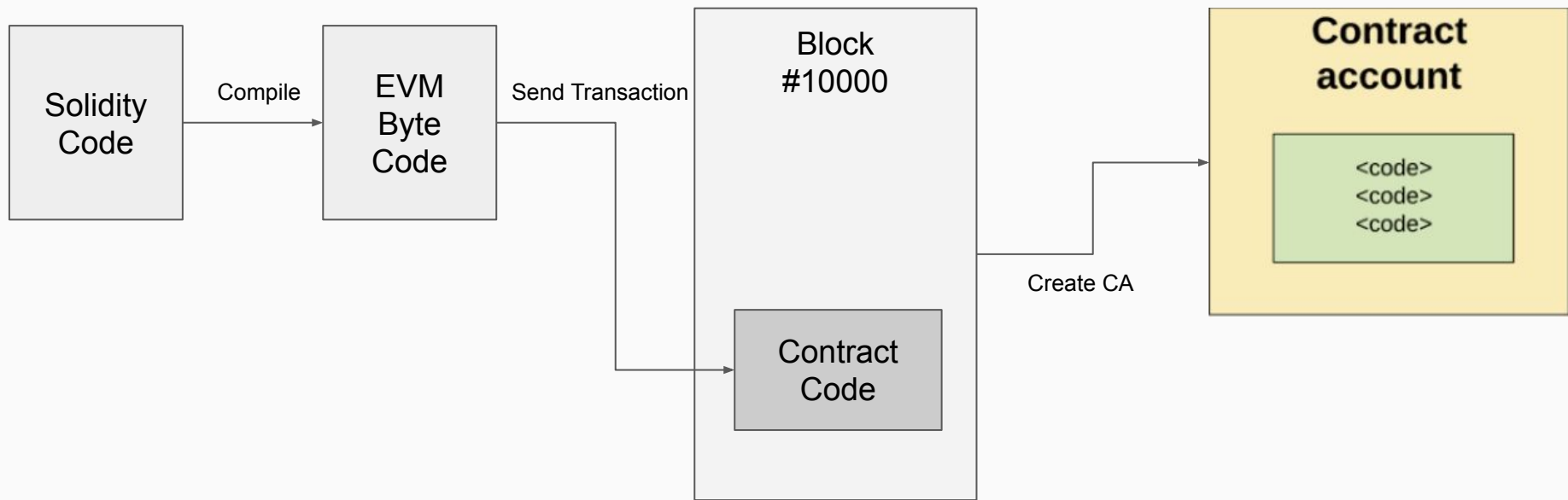
- DBMS의 SQL과의 비교

- 두 프로그래밍 언어 모두, 데이터를 조회하거나, 수정하는데 사용
- 이더리움 스마트 컨트랙트는 “튜링완전언어”로, SQL 보다 복잡한 형태의 로직을 구성하고 처리할 수 있음.

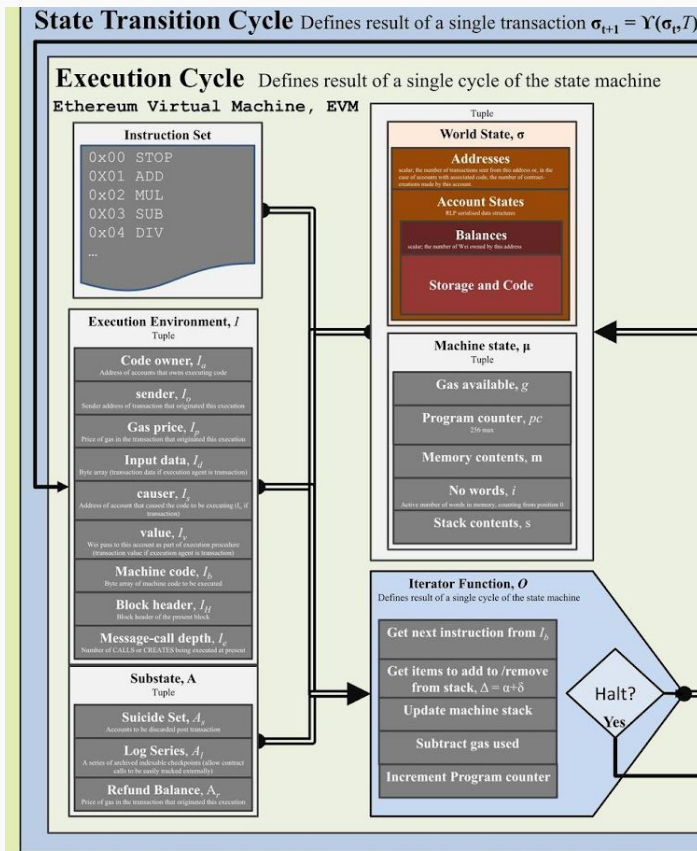
- 튜링머신
 - 튜링완전언어(모든 수학적문제를 풀 수 있는 일반적인 알고리즘을 만들어낼 수 있는 컴퓨터언어) +
 - 무한한 저장공간
 - 모든 계산 가능한 문제를 계산해내는 기계 = 튜링기계(인간의 뇌)
- 튜링 완전 언어의 조건
 - 1)프로세스를 충분히 분할할 수 있을 만큼 **작은 단위**를 사용할 수 있어야 한다
 - 2)**조건설정**과 **반복 명령어**가 있어야 한다.

Smart Contract

- 스마트 컨트랙트의 빌드/배포 프로세스



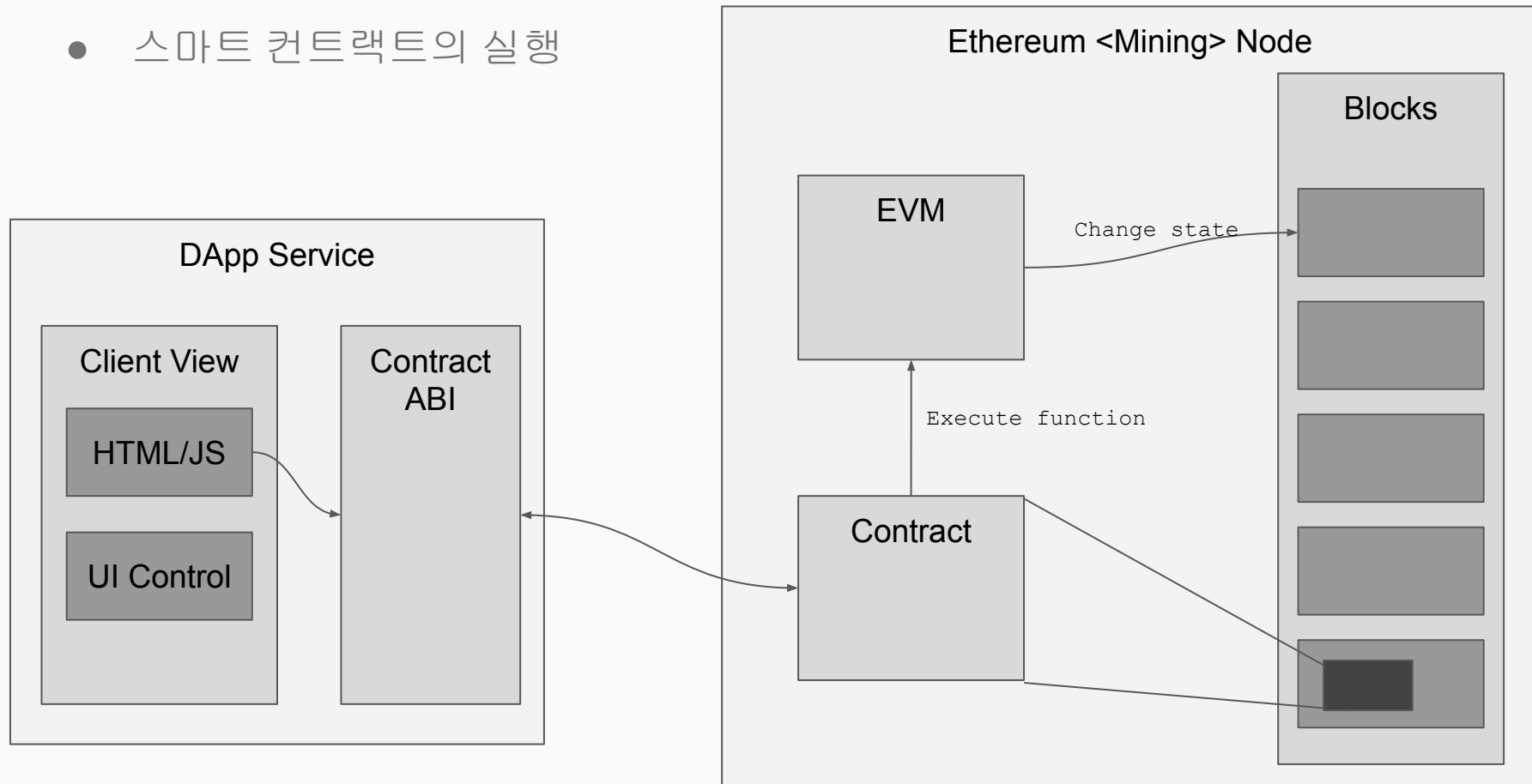
Smart Contract



- 스마트 컨트랙트 배포 및 노드 동기화
 - 컨트랙트 배포에대한 트랜잭션이 완료되면, State Trie에 바이너리 코드가 업로드
 - 모든 노드들이 동일한 컨트랙트 코드를 보유.

Smart Contract

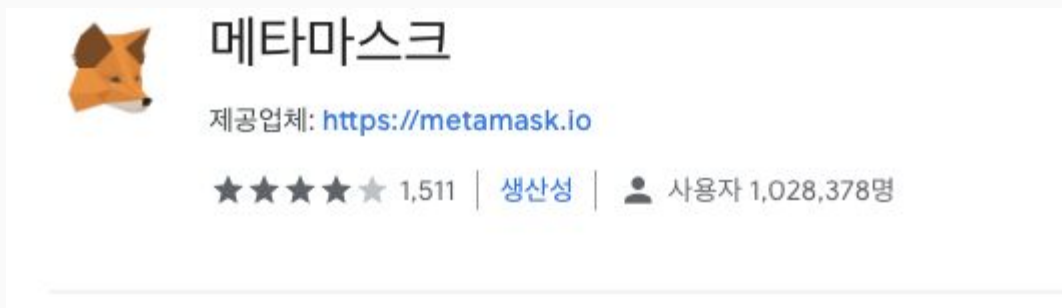
- 스마트 컨트랙트의 실행



- 스마트 컨트랙트를 실제로 배포 하고 실행해 보기
- 사전 준비
 - Metamask
트랜잭션 생성/서명을 쉽게 처리하기 위한 이더리움 지갑 어플리케이션
 - Remix IDE
이더리움 재단에서 제공하는 스마트 컨트랙트 개발 IDE
컨트랙트 개발, 배포, 테스트를 UI환경에서 지원

- Metamask 설치


- <https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn>



- Remix IDE 접속

<https://remix.ethereum.org/>

Smart Contract 실습

 Home X

remix

1 tabs

The new layout has arrived

[Learn more](#)

[Use previous version](#)

Environments

Solidity

Vyper

Workshops

Featured Plugins

Pipeline

Debugger

[See all Plugins](#)

File

New File

Open Files

Connect to Localhost

Import From:

Gist

GitHub

Swarm

Ipfs

https

Resolver-engine




Resources

Documentation

Gitter channel


Medium Posts

Tutorials

  0 ☐ listen on network  Search with transaction hash or address

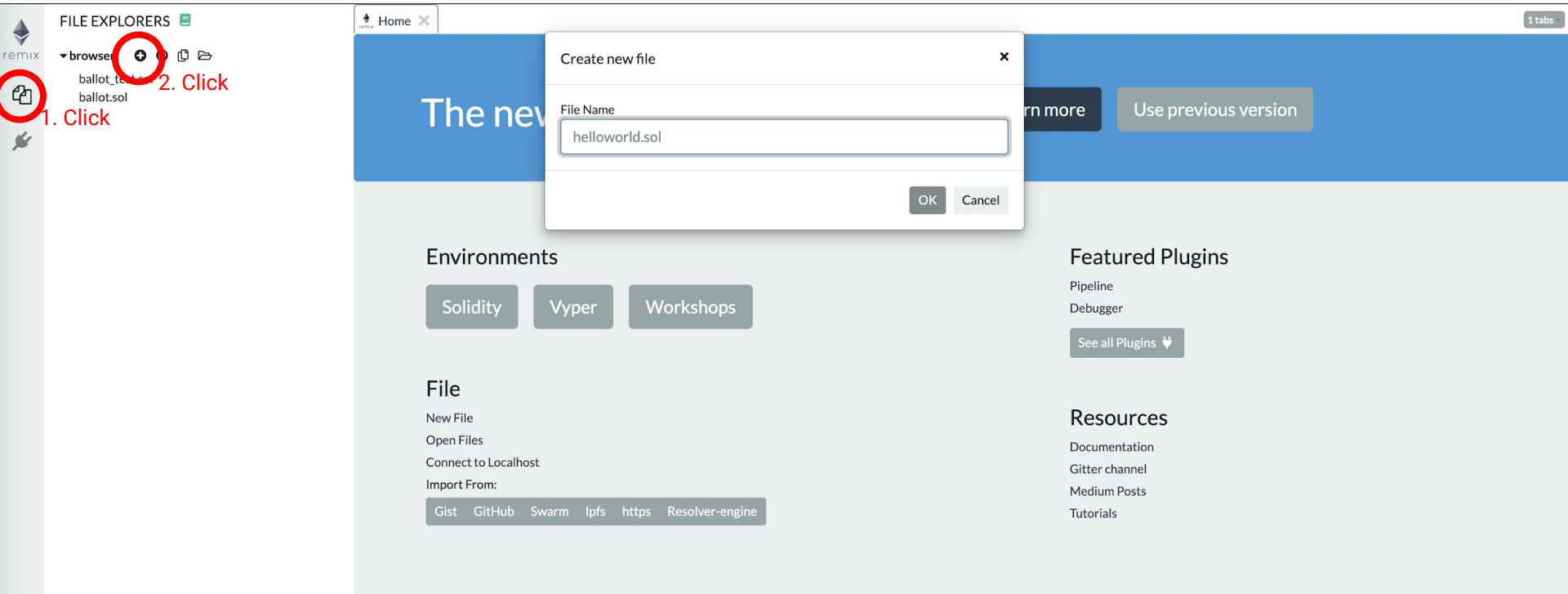
```
- Welcome to Remix v0.8.8 -

You can use this terminal for:
• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:
  o web3 version 1.0.0
  o ethers.js
  o swarmgw
  o remix (run remix.help() for more info)
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
• Use exports.register(key, obj).remove(key).clear() to register and reuse object across script executions.
```

 >

SmartContract 실습

- 컨트랙트 소스코드 파일 생성



SmartContract 실습

- 소스코드 작성

```
pragma solidity ^0.5.0;


contract HelloWorld{
    address public creator;

    event Hi(address indexed);

    constructor() public{
        creator = msg.sender;
    }

    function hi() public {
        emit Hi(msg.sender);
    }

    function changeCreator() public {
        creator = msg.sender;
    }
}
```



```
remix Home helloworld.sol x
1  pragma solidity ^0.5.0;
2
3  contract HelloWorld{
4      address public creator;
5
6      event Hi(address indexed);
7
8      constructor() public{
9          creator = msg.sender;
10     }
11
12     function hi() public {
13         emit Hi(msg.sender);
14     }
15
16     function changeCreator() public{
17         creator = msg.sender;
18     }
19 }
```


SmartContract 실습

- 확장 프로그램 추가
 - Deploy & Run Transactions
 - Solidity Compiler

Deploy & Run Transactions

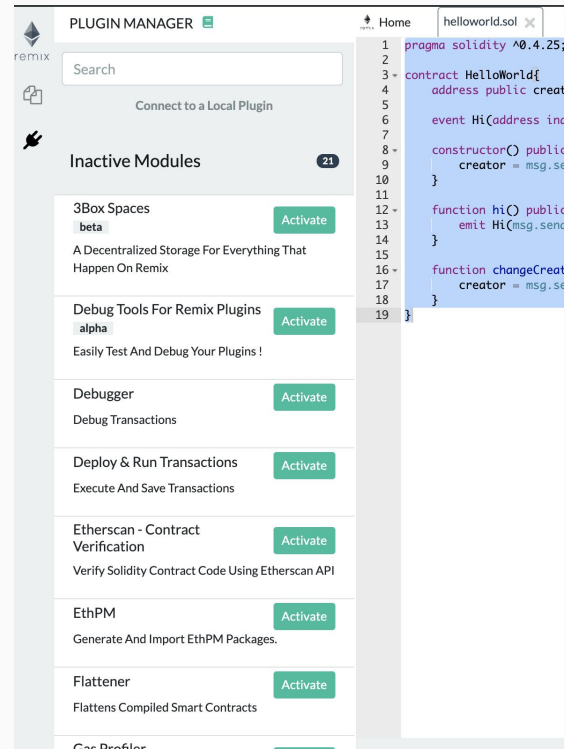
Activate

Execute And Save Transactions

Solidity Compiler

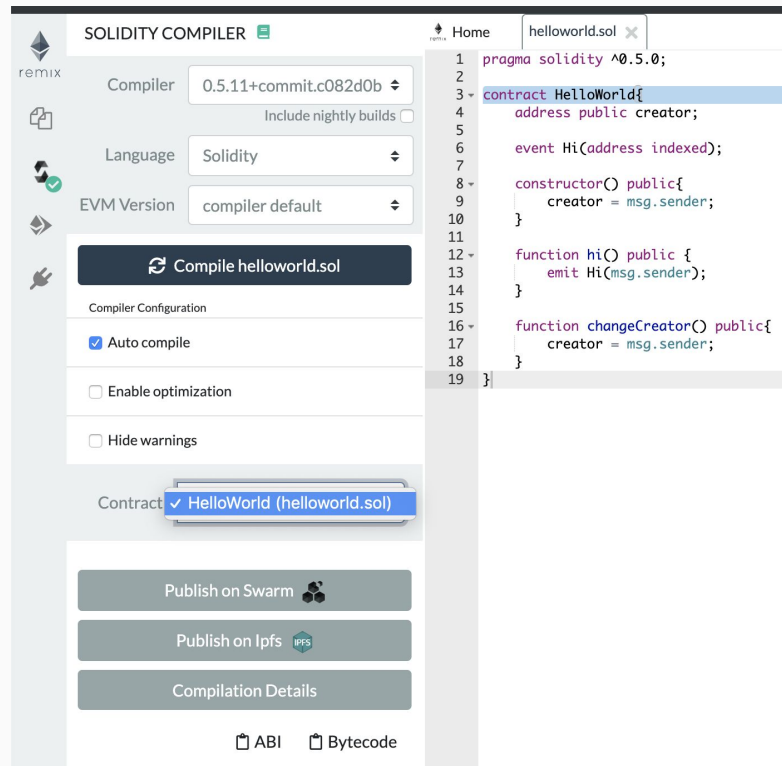
Activate

Compile Solidity Contracts



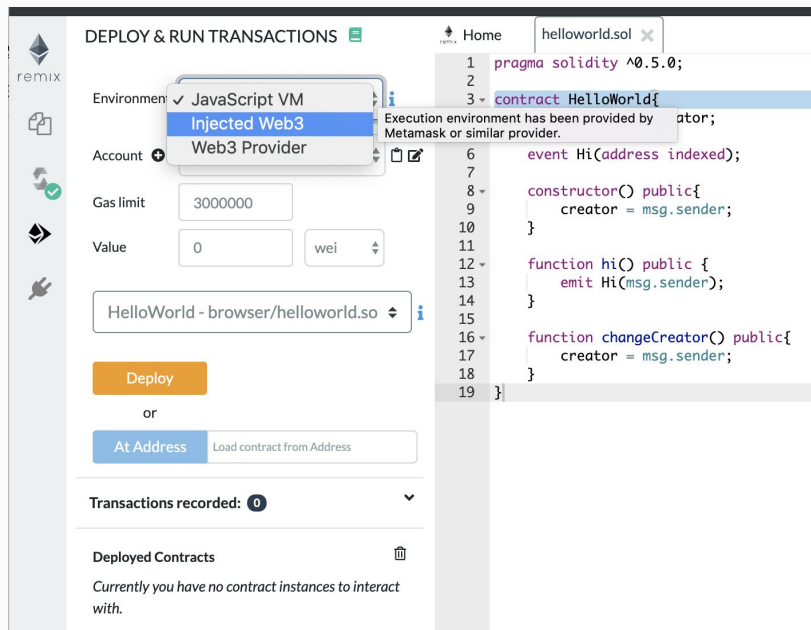
SmartContract 실습

- 컨트랙트 컴파일
 - Solidity Compiler 탭 선택
 - Compile helloworld.sol 버튼 클릭
또는 Auto Compile 옵션 선택



SmartContract 실습

- 배포 환경 설정
 - Deploy & Run Transactions 탭 선택
 - Environment 에서 [Injected Web3] 선택



TEST!

- 기존 소프트웨어 개발과 스마트컨트랙트 개발과의 차이점
 - 오류 발견시 패치 불가
한번 배포된 코드는 수정이 불가능하다.
단, 삭제는 가능함.
 - 디버깅의 어려움
폐쇄적인 환경이 아닌, 퍼블릭한 네트워크 위에서 동작.
실행 중간의 상태를 확인 불가.
Remix의 디버거도 이미 블록에 들어간 트랜잭션에 대해 복기하는 형태