# Blockchain and Cryptocurrency

Assignment 8: Writing a Smart Contract

The 8th assignment is about smart contracts. It will help you to understand the basic mechanism of smart contracts. Through this assignment, you can practice to program smart contract and learn how it runs on the Ethereum network. To create a smart contract, you have to use Solidity. You can refer this web site (☞ https://cryptozombies.io/, https://docs.soliditylang.org/en/v0.8.13/) to understand Solidity.

Description of this assignment:

- This assignment includes creating three special-purpose smart contracts.
- **You must submit a report containing the results obtained as you go through each step and your contract codes.**
- For this assignment, you are encouraged to use Metamask (cryptocurrency wallet) and Remix (Ethereum IDE) to connect to the private network you already built. (of course, you can use other options to write, compile, and test your smart contracts. However, you should deploy your smart contracts on your private network)

**<Step. 1> Run your own private network with rpc options**

In order to connect to **geth** running on the remote server from the local PC, the **rpc** function must be applied when **geth** is executed. The option used at this time is **--http**, and execute **geth** referring to the example below. (--datadir, --http.port, --networkid must be changed to your own)

```
>> geth --datadir ./data/ --http --http.addr 0.0.0.0 --http.port 2232 --http.corsdomain="₩*" --http.api web3,eth,debug,personal,net --vmdebug --networkid 8088 console
```

**<Step. 2> Install Metamask and connect to your own private network and your accounts**

Refer to this URL ☞ https://metamask.io/download/, and download it on the browser. Then, add your private network using **rpc** information. The RPC URL is of type http://yourserverip:rpcport.

After connecting to your private network, you can get your accounts using JSON file in the **geth** keystore and password

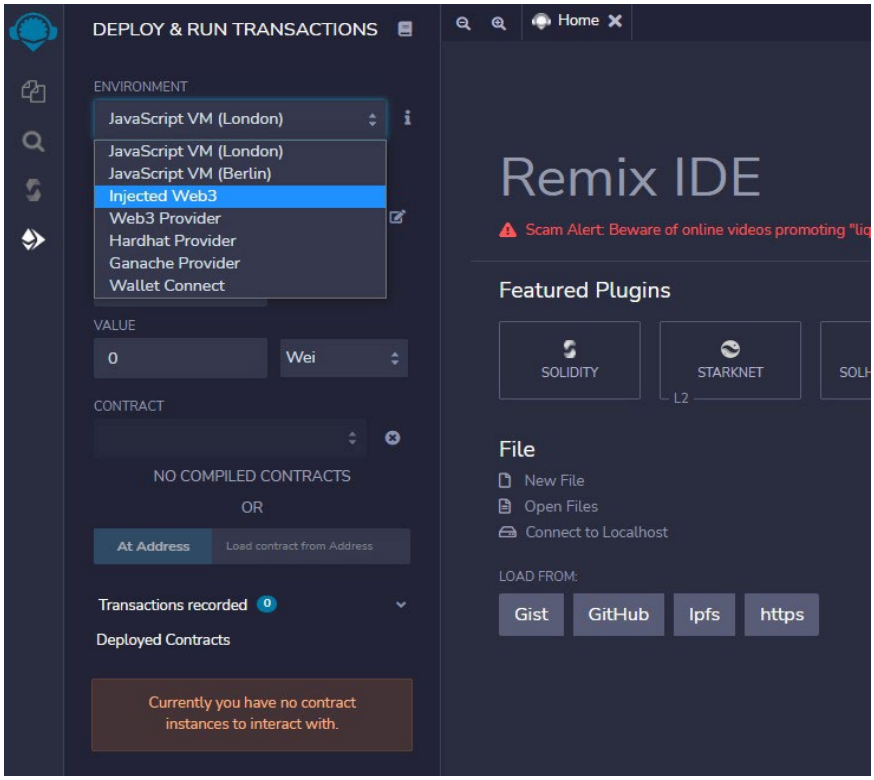
.

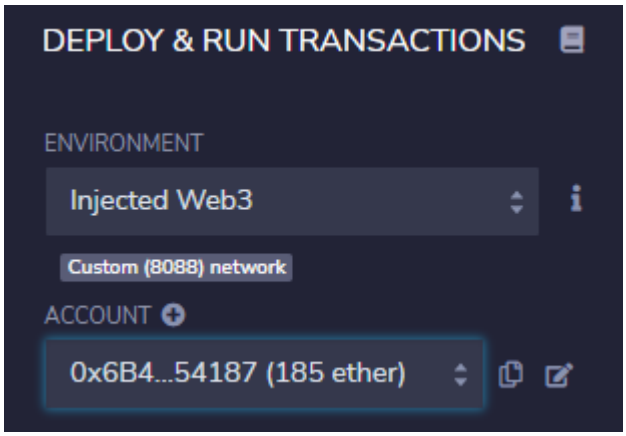
**<Step. 3> Connect Remix to your own private network**

You can create and compile smart contracts using Remix.

※ Go to this URL ☞ https://remix.ethereum.org/.

When deploying your smart contracts to your private network after writing smart contracts, you must set **ENVIRONMENT**. Select **Injected Web3**, then, you can check your accounts linked to the Metamask.




※ Refer to this document ☞ https://remix-ide.readthedocs.io/en/latest/index.html to write, compile, and deploy smart contracts.

<**Step. 4> Create a simple smart contract**

Test the basic example of smart contract below and try to understand how the smart contract works on the Ethereum network.

4-1) Create a new smart contract that contains the code below and compile it.

```
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage
    { uint storedData;

    function set(uint x) public
        { storedData = x;
    }

    function get() public constant returns (uint)
        { return storedData;
    }
}
```

4-2) If you compile the smart contract successfully, publish it on the private network.

4-3) After the block contains the transaction generated by the corresponding smart contract is mined,

- You can check the status of the smart contract in detail. Capture the status and attach it to the report.

4-4) Call functions included in the smart contract (Sending a transaction) and show the results in the report.

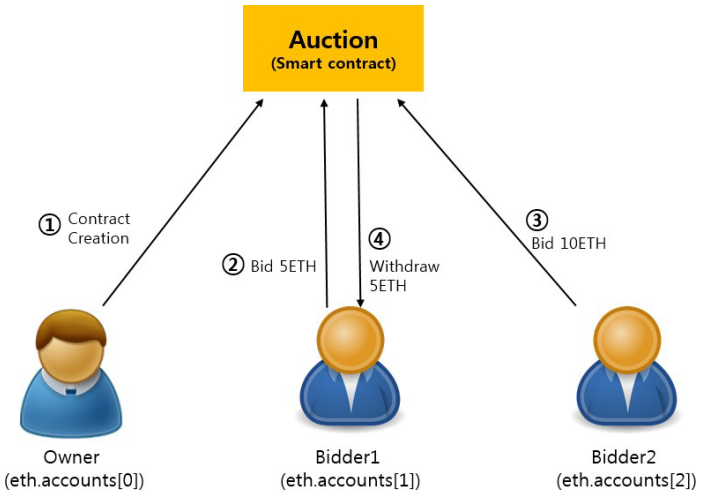
4-5) Add logging function to keep track of who recently changed data

(Then, test it)

**<Step. 5> Create a smart contract for auction**

Scenario is the following:

1. Owner makes Auction smart contract.
2. Bidder 1 bids 5 ETH to Auction smart contract.
3. After that, Bidder 2 bids 10 ETH to Auction smart contract.
4. Because the bid amount of Bidder 2 is higher than the bid amount of Bidder 1, Bidder 1 can withdraw the amount of Bidder 1's bid.




```
pragma solidity >=0.4.16 <0.9.0;

contract Auction {
    address public highestBidder; // The highest bidder's address
    uint public highestBid; // The amount of the highest bid
    mapping(address => uint) public userBalances;   // mapping for the amount to return

    constructor() public {
        // contractor
        // 1. Initialize highest bid and the bidder's address
    }

    function bid() public {
        // Funtion to process bid
        // 1. Check if the bid is greater than the current highest bid
        // 2. Update status variable and the amount to return
    }

    function withdraw() public {
      // Function to withdraw the amount of bid to return
        // 1. Check if the amount to return is greater than zero
        // 2. Update status variable and return bid
    }
}
```

This is an incomplete code of smart contract for auction. You can implement Auction smart contract, filling contents of each function in this code. Also, in order to complete this contract, you can add and modify something such as state variables, functions, modifiers, etc.

5-0) Prepare three accounts for testing your contract.
(**Account 1:** Owner, **Account 2:** Bidder 1, **Account 3:** Bidder 2)
5-1) Complete Auction smart contract and save it as "Auction.sol".
5-2) Compile the code.
5-3) Send transaction to create the contract, on account 1(owner).
5-4) Check the balance of contract account and account 2, the amount of the highest bid and address of the highest bidder.
5-5) Send transaction to execute bid() function in other to bid 5 ETH, on account 2(Bidder 1).
5-6) Check the balance of contract account 1, account 2 and account 3, the amount of the highest bid and address of the highest bidder.
5-7) Send transaction to execute bid() function in other to bid 10 ETH, on account 3(Bidder 2).
5-8) Check the balance of contract account and account 3, the amount of the highest bid and address of the highest bidder.
5-9) Send transaction to execute withdraw() function in other to withdraw the amount of bid, on account 2.
5-10) Check the balance of contract account and account 2.

**<Step. 6> Create a smart contract for crowd funding**

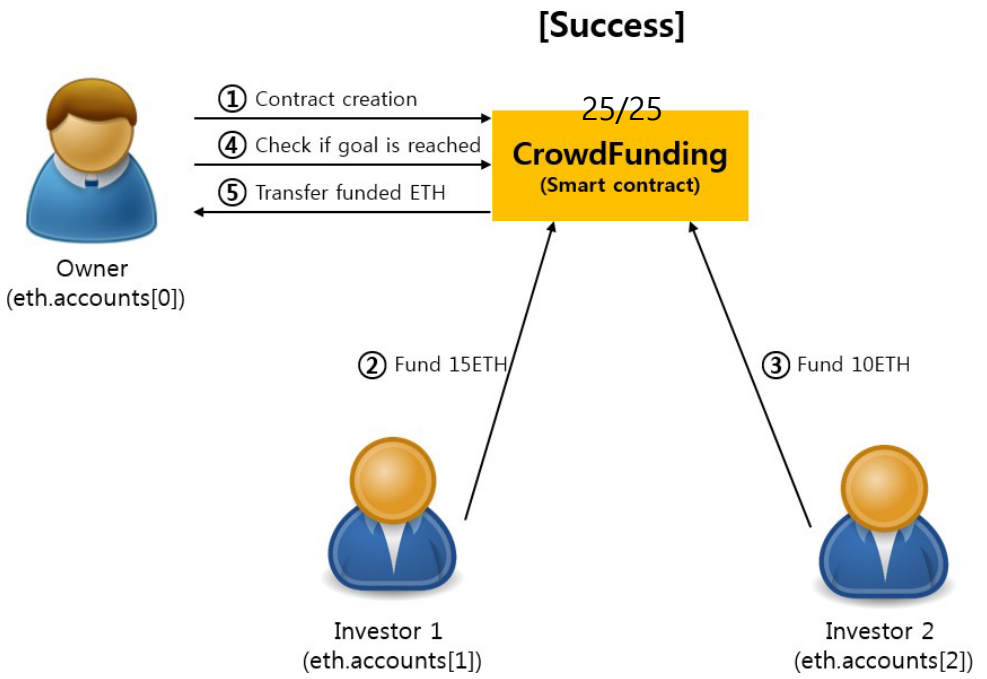
Crowdfunding smart contact creates contracts for fundraising purposes only and collects ETH. An investor can invest in a way that creates a transaction that sends an investor to this contract.

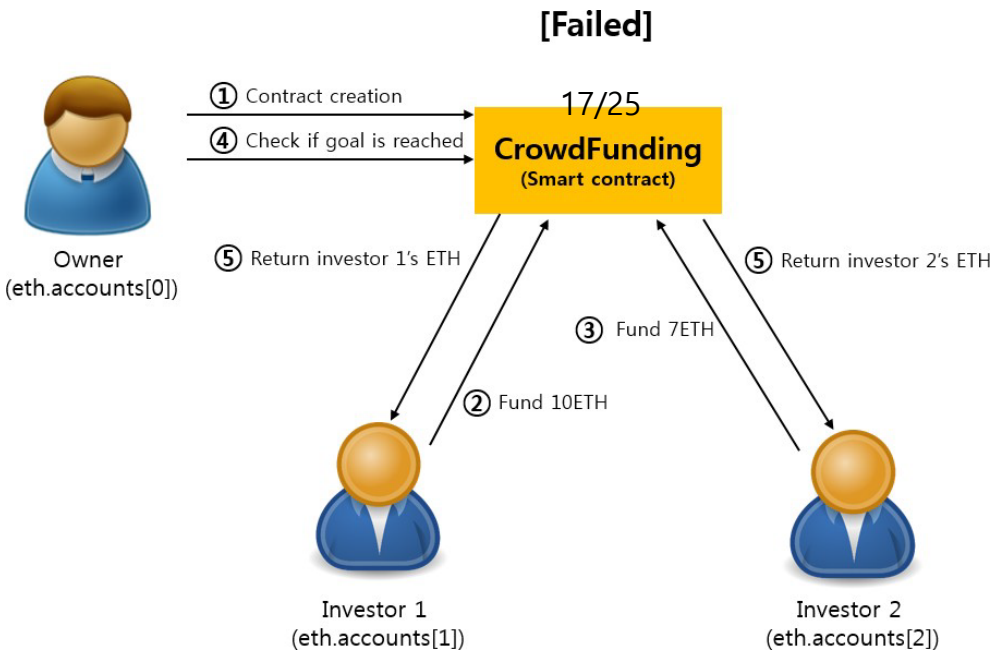
The contract sets up the deadline and target amount for the fundraising activities and remits the collected ETH to the owner of the contract when the target value is achieved at the closing date. If the target value is not achieved, return ETH to the investor.

```
pragma solidity >=0.4.16 <0.9.0;

contract CrowdFunding {
    // Investor struct
     struct Investor {
        address addr; // investor's address
        uint amount;   // investment amount
    }
    address public owner;   // contract owner
    uint public numbInvestors;   // the number of investors
    uint public deadline;   // deadline for this contract to be closed
    string public status; // "Funding", "Campagin Success", "Campagin Failed"
    bool public end;   // the end of funding
    uint public goalAmount; // target amount
    uint public totoalAmount;   //total ammout
    mapping(uint => Investor) public investors;

    // 1. Create modifier to limit to owner

    // Constructor
    constructor(uint _duration, uint _goalAmount) public {
        // Initialize owner, deadline, goalAmount, status, end, numbInvestors, totoalAmount
    }

    // Function to be called when investing
    function fund() public {
        // 1. Check if this crowd funding ended or not
        // 2. Set invest-related info and process funding
    }

    function checkFoalReached () public {
        // 1. Check if this crowd funding ended or not

        // 2. Check if the deadline is past or not

        // 3-1. If this crowd funding is successful, send funded ETH to owner
        // 3-2. If not, return fund-raising to each investor
        // Consider updating status and end
    }

    // 1. Create function to destroy this contract


}
```

This is an incomplete code of smart contract for crowd funding. You can implement CrowdFunding smart contract, filling contents of each function in this code. Also, in order to complete this contract, you can add and modify something such as state variables, functions, modifiers, etc.

There are two scenarios, **Success** and **Failed** of funding.



6-0) Prepare three accounts for testing your contract.
(**Account 1:** Owner, **Account 2:** Investor 1, **Account 3:** Investor 2)

6-1) Complete CrowdFunding smart contract and save it as "CrowdFunding.sol".

6-2) Compile the code.

6-3) Send transaction to create the contract, on account 1(owner).
(When creating the contract, set _duration to 300 (5min) and _goalAmount to 25 ETH)

6-4) Check deadline and end state.

**<<<<<Case of successful fund>>>>>**

6-5) Send transaction to execute fund() function in other to fund 15 ETH and 10 ETH from Investor 1 and Investor 2 respectively, on account 2 and 3.

6-6) Check investment of Investor 1, 2.

6-7) Check total investment of the contract and balance of contract account.

6-8) Check owner's balance (Account 1)

6-9) Send transaction to execute checkGoalReached() function to check the fundraising results
(Do it after the deadline!!!)

6-10) Check deadline and end state.

6-11) Check balance of contract account and owner.

**<<<<<Case of failed fund>>>>>**

6-12) Follow steps from 5-3 to 5-4.

6-13) Send transaction to execute fund() function in order to fund 10 ETH and 7 ETH from Investor 1 and Investor 2 respectively, on account 2 and 3.

6-14) Check investment of Investor 1, 2.

6-15) Check total investment of the contract and balance of contract account.

6-16) Check balance of Investor 1, 2 and owner.

6-17) Send transaction to execute checkGoalReached() function to check the fundraising results
(Do it after the deadline!!!)

6-18) Check deadline and end state.

6-19) Check balance of contract account.

6-20) Check balance of Investor 1, 2 and owner.

<TA>

Jeongheon Kim <kjheon1118@postech.ac.kr>