Cryptocurrencies and Blockchain

Sheet 3, submission by Konstantin Bork, Sidney Steimel & Marvin Ullrich

Assignment. Ethereum Blockchain and Smart Contracts

Our smart contract offers the ability to proof that some item behind a hash existed since the timestamp it gets when it was created. These items with their temperproof timestamp can also be sold.

Use cases:

- Proof you had an idea at some point. This can be useful for example when dealing with patents.
- Proof can be sold.
- Selling digital goods, for example copyrights of a picture.
- Marking art with a code and putting it on the blockchain so ownership can be tracked.

Testing was done this way:

1. Go to: http://remix.ethereum.org/
2. Create new Contract containing this file.
3. Compile
4. Go to tab "Run"
5. Set Environment to "JavaScriptVM"
6. Click "Deploy"
7. Under "Deployed Contracts", select new "Ownership"-contract
8. In the dropdown click "createItem" and set values accordingly (e.g. "aaa", false, 1, "bbb"), click transact. Item has now been created and is not sellable
9. Check if everything went smooth by calling "infoItem"-function, debug info should show the item and return its properties
10. Click "makeItemSellable" and set values accordingly (e.g. "aaa", 5), item is now buyable for others for 5 WEI, not ETH!
11. Select a different account and click "buyItem", now buy the item for 5 WEI. Everything should work smoothly.
12. Test other functions and play around with it, everything works as intended

ownership.sol

```
pragma solidity >=0.4.22 <0.6.0;

contract Ownership {

    // the structure of our item, swarm_info can be used by web apps to
store and load additional
    // information from a swarm file that can be saved as a json file for
example and contain its
    // own structure depending on the app. This way we provide dynamic
interface for apps
```

```
    struct Item {
        address payable owner;
        uint timestamp;
        bool sellable;
        uint price;
        string swarm_hash;
    }

    // since we do not know the length of the used hash, we map a dynamic
string to our Item
    // this way the user can map a regular hash or a swarm hash for
example to his item
    mapping(string => Item) items;

    // Nothing needs to be done here
    constructor() public {

    }

    modifier onlyOwner(string memory _hash) {
        require(
            msg.sender == items[_hash].owner,
            "Only the current owner can do this."
        );
        _;
    }

    // Here we create our item and map the hash to the item
    function createItem(string memory _hash, bool _sellable, uint _price,
string memory _swarm_hash ) public {
        // if the item does not exist, this will always be the address
        require(
            items[_hash].owner ==
0x0000000000000000000000000000000000000000,
            "Item already exists."
        );

        items[_hash].owner = msg.sender;
        items[_hash].sellable = _sellable;
        if (_sellable){
            items[_hash].price = _price;
        } else{
            items[_hash].price = 0;
        }
        items[_hash].swarm_hash = _swarm_hash;
        // timestamp is fixed and can be used as a proof that whatever is
behind that hash existed since at least this timestamp
        items[_hash].timestamp = block.timestamp;
    }

    // onlyOwner checks if owner is sender and also checks if item exists
```

```
    function makeItemSellable(string memory _hash, uint _price) public
onlyOwner(_hash) {
        // Set sellable to true
        items[_hash].sellable = true;

        // Finally, set the price
        items[_hash].price = _price;
    }

    function makeItemNotSellable(string memory _hash) public
onlyOwner(_hash) {
        // Set sellable to false
        items[_hash].sellable = false;
        items[_hash].price = 0;
    }

    function buyItem(string memory _hash) payable public {
        // At first, check if the item is sellable
        require(
            items[_hash].sellable == true,
            "The item cannot be sold."
        );

        // Check if the amount of ethers is high enough
        require(
            msg.value == items[_hash].price,
            "Wrong Price!"
        );

        // Check if the message sender is not the current owner
        require(
            msg.sender != items[_hash].owner,
            "The current owner cannot buy the item."
        );

        items[_hash].owner.transfer(msg.value);
        items[_hash].owner = msg.sender;
        items[_hash].sellable = false;
        items[_hash].price = 0;
    }

    // In case the owner wants to transfer the item to a new owner
    function transferItem(string memory _hash, address payable new_owner)
public onlyOwner(_hash) {
        // Receive ethers
        items[_hash].owner = new_owner;
        items[_hash].sellable = false;
        items[_hash].price = 0;
    }

    // function for a webapp to view an item/hash
```

```
    function infoItem(string memory _hash) public view returns (address,
bool, uint, string memory, uint ){
        // Receive ethers
        return (items[_hash].owner, items[_hash].sellable,
items[_hash].price, items[_hash].swarm_hash, items[_hash].timestamp );
    }

    function setSwarmInfo(string memory _hash, string memory _swarm_hash)
public onlyOwner(_hash) {
        items[_hash].swarm_hash = _swarm_hash;
    }

    // in case the item should be deleted, deleting is good as the
ethereum virtual machine needs to store less.
    function deleteItem(string memory _hash) public onlyOwner(_hash) {
        delete items[_hash];
    }
}
```

Smart Contract Deployment

On the test server, we created a new account with the public address 0x8648dab59cf4ee1b5e8d99275db6ab7fe3ceba10 and transferred 1000 ETH from the coinbase to our new account. Then, we copied the WEB3DEPLOY code from Remix to geth and ran it.

```
var ownershipContract = web3.eth.contract([{"constant":false,"inputs":
[{"name":"_hash","type":"string"}],"name":"buyItem","outputs":
[],"payable":true,"stateMutability":"payable","type":"function"},
{"constant":false,"inputs":[{"name":"_hash","type":"string"},
{"name":"_swarm_hash","type":"string"}],"name":"setSwarmInfo","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":true,"inputs":
[{"name":"_hash","type":"string"}],"name":"infoItem","outputs":
[{"name":"","type":"address"},{"name":"","type":"bool"},
{"name":"","type":"uint256"},{"name":"","type":"string"},
{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","ty
pe":"function"},{"constant":false,"inputs":
[{"name":"_hash","type":"string"}],"name":"deleteItem","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":false,"inputs":[{"name":"_hash","type":"string"},
{"name":"_sellable","type":"bool"},{"name":"_price","type":"uint256"},
{"name":"_swarm_hash","type":"string"}],"name":"createItem","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":false,"inputs":
[{"name":"_hash","type":"string"}],"name":"makeItemNotSellable","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":false,"inputs":[{"name":"_hash","type":"string"},
{"name":"new_owner","type":"address"}],"name":"transferItem","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
```

```
{"constant":false,"inputs":[{"name":"_hash","type":"string"},
{"name":"_price","type":"uint256"}],"name":"makeItemSellable","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"inputs":
[],"payable":false,"stateMutability":"nonpayable","type":"constructor"}]);

var ownership = ownershipContract.new(
{
    from: web3.eth.accounts[1],
    data: # compiled contract,
    gas: '4700000'
}, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
        console.log('Contract mined! address: ' + contract.address + '
transactionHash: ' + contract.transactionHash);
    }
})
```

After some seconds, the confirmation message was displayed:

```
Contract mined! address: 0xb741ffcfc4836fe624e859c89577ce1be0014467
transactionHash:
0xf0d4995133026b0b334c96851d6aef8665d0ff9f7a7c14af3fcd146419f7db25
```

With this line, we get the block number (and some more information) of the contract transaction: eth.getTransactionReceipt("0xf0d4995133026b0b334c96851d6aef8665d0ff9f7a7c14af3fcd146419f7db25")

Our contract is in block 93158. To test our contract on the test server, we created a MD5 hash [MD5 ("This is a secret I found!") = d7814a129206203670350bbf07353f0f] and created a new item for our account. Of course, you should not use MD5 in reality.

```
ownership.createItem("d7814a129206203670350bbf07353f0f", false, 5, "")
"0x530a9d0872ac485858f6913f4bab3c7015f861a703a53be60134695477fcc3fe"
```

We again get the block number (and some more information) of the transaction created above: eth.getTransactionReceipt("0x530a9d0872ac485858f6913f4bab3c7015f861a703a53be60134695477fcc3fe")

This transaction can be found in block 93225.