LESSON SC 14 – External Function Calls

University of West Attica

Department of Electrical and Electronics Engineering

Ioannis Christidis

Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

In this lesson we will learn about `external function calls` and we will make some changes to our SCs.

Let's start by asking some questions:

What will happen if the owner of an `ICR` SC registers an `address` as a microcontroller account in a car that is not actually a microcontroller?

The owner of the SC could just register an `address` that is not a microcontroller rendering the DApp useless, since there will be no guarantee if the cars have a valid microcontroller `address` or not.

Is there a way right now in the SC to check if the `addresses` registered are indeed microcontroller `addresses`?

We have a `mapping` that checks if the `address` is already used in another car but no way to check if the `address` is from a microcontroller account.

Mitigation

Unfortunately, there is no way direct way right now to check if an `address` is valid.

It is important to understand the weaknesses of our DApp, because once a SC is deployed, we cannot alter it.

Now, we can think of multiple ways to mitigate this but for the purpose of this course we will alter our scenario.

The cars will be registered by the IIoTC instead of the `ICR owner`.

Advantages:

1. The `owner` will not be able to register invalid `addresses`.

Disadvantages:

1. We need to trust that the IIoTC will not add invalid `addresses`. The IIoTC however will have no benefit from adding invalid `addresses` since the users will stop using their DApp, resulting in losing money instead.
2. This mitigation makes our approach more centralized which is generally something we want to avoid.
3. We will need to find a way so that the IIoTC will not know the private key (pk) of the accounts used for the microcontrollers.

Mitigation (2)

To ensure that the IIoTC will not have access to the pk of the microcontroller accounts we will create the account after the microcontroller's installation to the car.

Here we must assume that the IIoTC will not have control over the microcontroller once installed in the car.

The typical workflow for registering cars in the `ICR` DApp would be similar to this:

a) A smart car fleet owner makes a deal with the IIoTC:

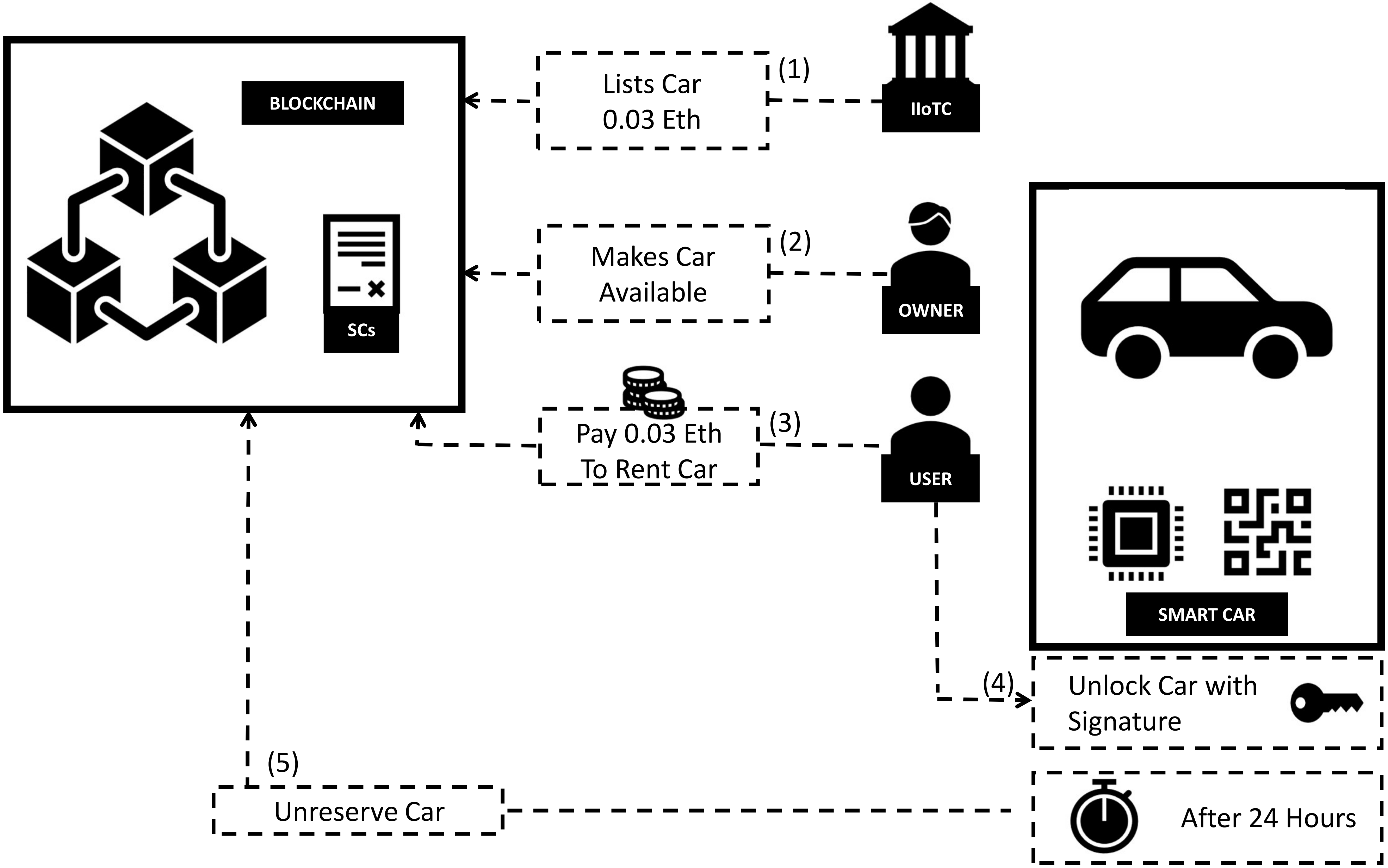
b) The IIoTC creates an `ICR` SC with the fleet owner's `address`.

c) The IIoTC installs the necessary components to the cars (microcontrollers /software/hardware).

d) The microcontrollers are activated, create their accounts and send the `addresses` to the IIoTC.

e) The IIoTC registers the cars on the `ICR` SC.

Then, the workflow continues as we already know.


BLOCKCHAIN
SCs
Lists Car
0.03 Eth
(1)
IIoTC
Makes Car
Available
(2)
OWNER
Pay 0.03 Eth
To Rent Car
(3)
USER
SMART CAR
(4)
Unlock Car with
Signature
(5)
Unreserve Car
After 24 Hours

Create the Function that Registers Cars

But with all that being said we can go to the coding part.

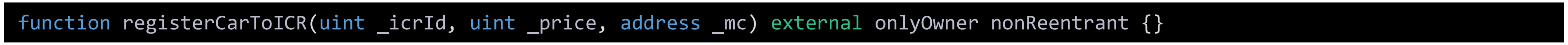
What we need to do is change the `ICR` SC so that only the IIoTC will be able to use the `registerCar` function. However, instead of directly calling the `registerCar` function in each `ICR` SC we will make it so that the `ICRFactory` will call the `registerCar` from the `ICRs`.

As we mentioned before this is an *external function call from a SC to a SC* so what we are going to do, is create a function called `registerCarToICR` that will handle the interaction with `ICR` SCs.

It will take as parameter the ID of the `ICR` we want to register the car, the price of the car and the microcontroller's `address`.

Only the `owner` of `ICRFactory` should call this function as we already discussed before.

Because we are making an external function call, we must make this function `nonReentrant`.



```
function registerCarToICR(uint _icrId, uint _price, address _mc) external onlyOwner nonReentrant {}
```

Preparation

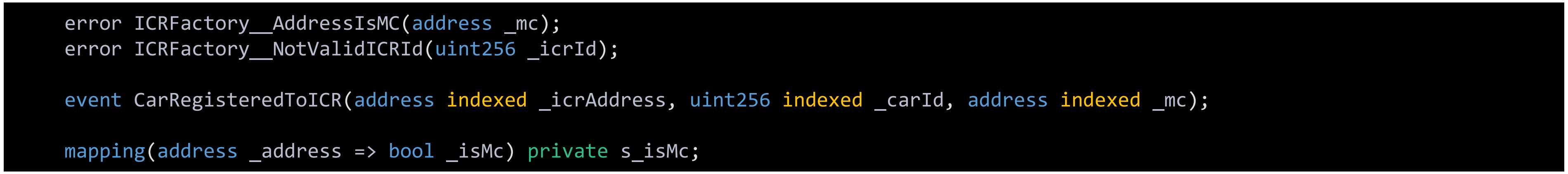
Before we continue to the functionality let's make some `custom errors` to:

1) Ensure that the microcontroller address is not used already in any `ICR` SC.

2) Ensures that the `ICR` SC is valid.

Let's also add an `event` for registering a car to an `ICR` SC.

Finally let's add a `mapping` that keeps track of all microcontrollers used in `ICR` SCs.



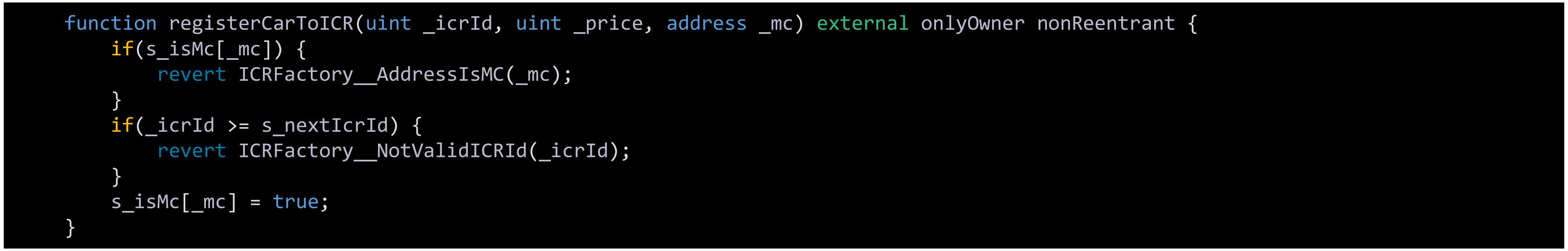
```
    error ICRFactory__AddressIsMC(address _mc);
    error ICRFactory__NotValidICRId(uint256 _icrId);

    event CarRegisteredToICR(address indexed _icrAddress, uint256 indexed _carId, address indexed _mc);

    mapping(address _address => bool _isMc) private s_isMc;
```

Function Explanation

We first check if `_mc` is already used.

Then we check if the `ICR` SC is valid.

If our checks pass, we register `_mc` as used.



```
    function registerCarToICR(uint _icrId, uint _price, address _mc) external onlyOwner nonReentrant {
        if(s_isMc[_mc]) {
            revert ICRFactory__AddressIsMC(_mc);
        }
        if(_icrId >= s_nextIcrId) {
            revert ICRFactory__NotValidICRId(_icrId);
        }
        s_isMc[_mc] = true;
    }
```

Next, we will discuss the interaction with the `ICR` SC.

If you remember in one of our previous lessons, we mentioned that we need two things to interact with any SC deployed on the blockchain:

1. The `address` of the SC
2. The `ABI` of the SC

Now in our case we have the `ICR` SC code `imported` in `ICRFactory` so to interact with the `ICR` we can use it instead of the `ABI`.

Function Explanation (2)

We get the address of the ICR SC from s_icrs mapping:

```
        address icrAddress = s_icrs[_icrId];
```

Next, we use the address to get an instance of the ICR SC since we have the ICR SC code imported.

```
        ICR icr = ICR(icrAddress);
```

Next, we emit the event we created signifying that a car was registered to an ICR SC. To get the _carId for the emittion of the event we do an external function call to the ICR using its getNextCarId function. We already did something similar to the deployICR function to get the owner of the ICR.

```
        emit CarRegisteredToICR(icrAddress, icr.getNextCarId(), _mc);
```

Finally, we make another external function call, this time for the registerCar function.

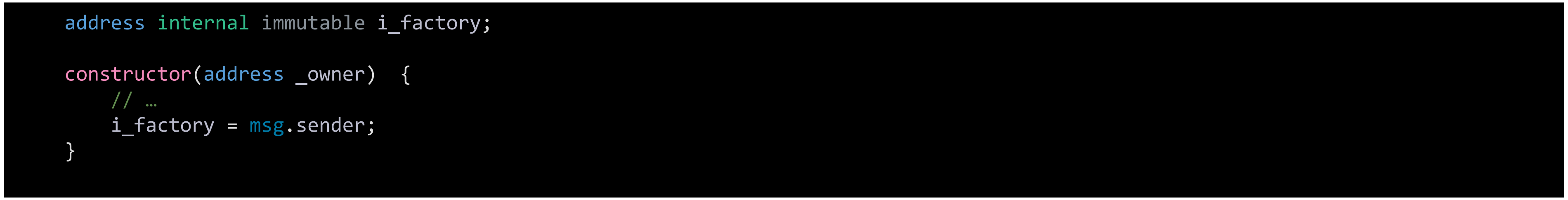
```
        icr.registerCar(_mc, _price);
```

You will notice that we emit the event before we do the external function call that registers the car. This serves two reasons. First, it is easy to get the ID of the next car to be registered by calling the getNextCarId function and second external interactions should always happen in the end of a function if possible.

Altering ICR Contract

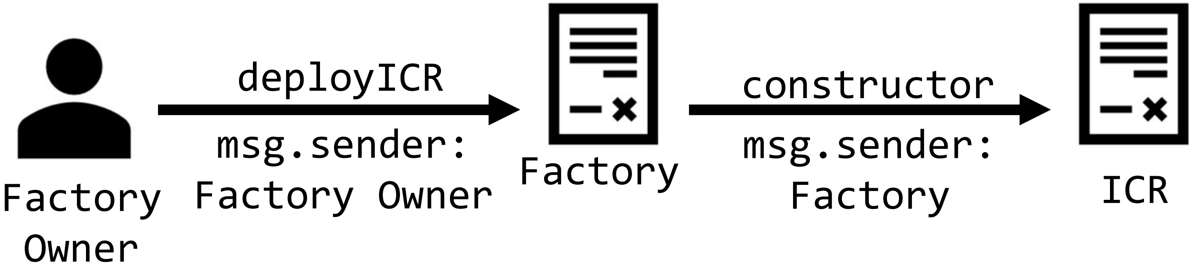
In the ICR SC we need to make a few changes. First, now that ICRFactory is responsible for registering new cars we need to alter the registerCar function.

To do this we will create a new immutable address variable that will store the address of the ICRFactory.



```
address internal immutable i_factory;

constructor(address _owner)  {
    // …
    i_factory = msg.sender;
}
```

Remember that when we are doing external function calls from a SC to another the msg.sender is the SC that makes the call. So, to use this to our advantage, when we deploy the ICR SCs we make the i_factory equal to msg.sender.



Altering ICR Contract (2)

Next in our `registerCar` function we change the check for the owner:

```
    function registerCarToMapping(address _mc, uint256 _price) external nonReentrant  {
         if (msg.sender != i_owner) {
             revert ICR__IsNotOwner(msg.sender);
         }
         //...
    }
```

To check for the `ICRFactory` instead:

```
    function registerCarToMapping(address _mc, uint256 _price) external nonReentrant  {
         if (msg.sender != i_factory) { // new
             revert ICRRegistry__IsNotFactory(msg.sender); // new
         }
         //...
    }
```

And with that we finished our SC.

You should try it out and see if and how it works.

Try to deploy `ICRFactory` and create an `ICR`. Then try to register cars to it.

Outro

This was a difficult lesson, but we finished it.

Next, we will make some last changes to the SC to make sure that we have our functionality ready for the microcontrollers.