LESSON MC6 – Full Example

University of West Attica
Department of Electrical and Electronics Engineering
Ioannis Christidis
Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

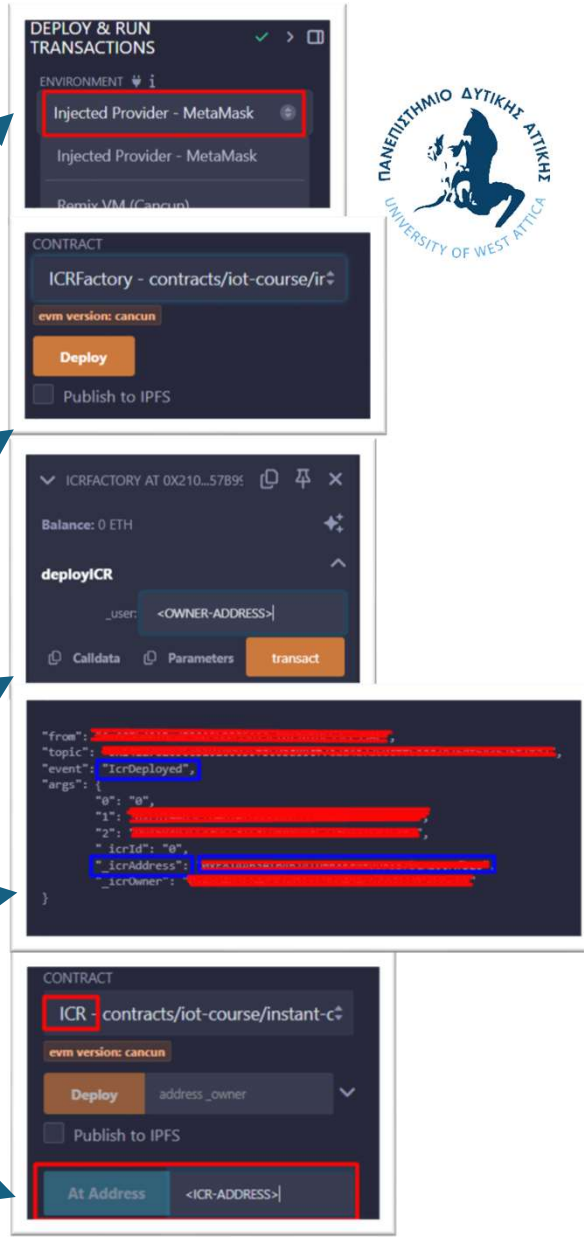
In this lesson we will try our SC and python scripts in Sepolia testnet.

We will need:

1. Metamask Wallet
2. Sepolia ETH for testing (0.3 should be enough)
3. Three Accounts for:
    - IIoTC (0.15 ETH)
    - OWNER (0.05 ETH)
    - USER that will rent the car (0.1 ETH)

Deploy the contracts

1. In Remix IDE open `ICRFactory` compile and on the Deploy and Run Transactions Sidebar go to the Environment and choose Injected Provider – MetaMask. MetaMask will pop up asking you to connect your account in Remix IDE. Make sure that you are on the Sepolia testnet.
2. Then, using the IIoTC account, deploy the `ICRFactory`. Every time you attempt to make a transaction MetaMask will pop up to confirm the transaction.
3. Next with the same account call the `deployICR` function adding as parameter the `address` of `OWNER`.
4. From the event `ICRDeployed` find the newly created `ICR address` and paste next to the At Address button under the Deploy button. Change the SC above from `ICRFactory` to `ICR` and press the At Address button. The new `ICR` SC will appear under the `ICRFactory`.



Set up and Run the microcontroller

In the microcontroller's `.env` file add the following:

```
USERS_PRIVATE_KEY= <USER-PRIVATE-KEY>
OWNERS_PRIVATE_KEY= <OWNERS-PRIVATE-KEY>
HTTPS_URL="https://eth-sepolia.g.alchemy.com/v2/<KEY>"
WSS_URL="wss://eth-sepolia.g.alchemy.com/v2/<KEY>"
FACTORY_ADDRESS='<ICR-FACTORY-ADDRESS>'
```

Next run your script from `main.py`

```
python3 main.py
```

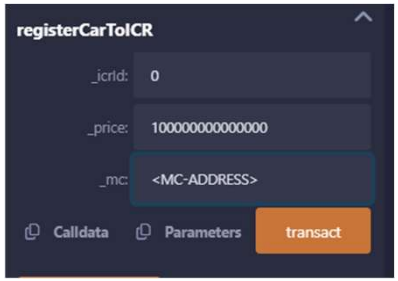
You will receive an outcome similar to this:

```
No account found. Creating a new Ethereum account...
New Ethereum account created:
Public Address: <NEW-ADDRESS-STORED-IN-ENV>
Account saved to .env. Keep this file secure!
Fetch ICR data from event or already existing
Opening Async Web3
Checking if ICR info exists
ICR info do not exist
Try to find matching ICR
Check factory for matching ICR
Matching ICR Address not found.
No matching car found.
ICR Address is the zero address. We will subscribe to Car Register To ICR event.
Subscribing to CarRegisteredToICR event for ICRFactory at <ICR-FCTORY-ADDRESS>
```

Register microcontroller

Our microcontroller created an account (stored `address` and `pk` in the `.env`), searched in the `ICRFactory` to see if it was already registered and because it was not, it subscribed to `CarRegisteredToIcr` event.

Back in Remix IDE, with the IIoTC account, call the function `registerCarToIcr` from the `ICRFactory` with:



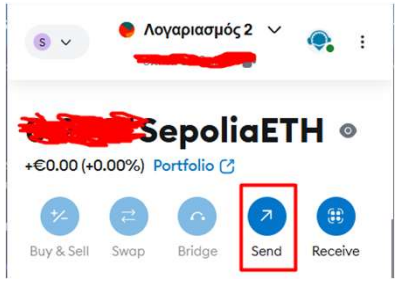
- `ICR ID = 0` (the first ICR),
- `price = 100000000000000` (`0.0001 ETH`, minimum car renting price)
- `mc address` (get it from the console logs)

When the transaction completes, you will see a log to the microcontroller's console.

```
Event received: ICR Address: <ICR-ADDRESS>, Car ID: 0, Microcontroller Address: <MC-ADDRESS>
Match found! Stored ICR Address: <ICR-ADDRESS>, Car ID: 0
Unsubscribed from CarRegisteredToICR events.
```

The `ICR Address` and the `Car ID`, that our microcontroller is registered to, is successfully saved to the `.env`.

Now you should send some ETH to the `address` of the microcontroller (from MetaMask). If you do not do this, you will not be able to change the car's status to AVAILABLE since we added that logic in the `ICR` SC.



Check the Logs

If you check the logs and you will see the following:

```
Connected to the Sepolia testnet successfully!
Chain ID: 11155111
Checking car status
Car is not Occupied
Enter your Message:
```

The first two logs are from the HTTPS connection to the Sepolia testnet.

The next two logs are from the periodic checks to the `ICR SC` to check the status of the car. Since the car is not rented, the logs state it is not occupied. This logs will appear every `3 minutes` since the rent time in the `ICR SC` is `6 minutes`, and we make checks every half of renting time (`6min /2 = 3min`).

```
    uint256 internal constant RENT_TIME = 6 minutes;
```

The last log says `"Enter your message"` and it is from the `unlock_car_by_message` script.

We can try this now to check how it works. In the `ICR SC` we defined that when the car is not `OCCUPIED` the current renter will be the OWNER. So, we can sign a message with the OWNER's pk to unlock the car right now. If the car is `OCCUPIED`, we have to use the pk of the USER that rented the car.

Changes to `signing_message` script (1)

We will do the following changes to the `signing_message.py` from the Lesson MC1 – Intro to web3.py.

We include the pks of the OWNER and the USER that will rent the car later. We also changed the message to "Unlock Car".

```python
from web3 import Web3
from eth_account.messages import encode_defunct
import os
from dotenv import load_dotenv

load_dotenv(override=True)

HTTPS_URL = os.getenv("HTTPS_URL")

# Connect to a Web3 provider
w3 = Web3(Web3.HTTPProvider(HTTPS_URL))

# Private key of the signers
OWNERS_PRIVATE_KEY = os.getenv("OWNERS_PRIVATE_KEY")
USERS_PRIVATE_KEY = os.getenv("USERS_PRIVATE_KEY")

# Message to sign
message = "Unlock Car" # New message
# Hash the message
message_hash = encode_defunct(text=message)
```

Changes to `signing_message` script (2)


```
# … Rest of the code
# Sign the hashed message
owners_signed_message = w3.eth.account.sign_message(
    message_hash, private_key=OWNERS_PRIVATE_KEY
)
owner_address = w3.eth.account.recover_message(message_hash, signature=owners_signed_message.signature)


users_signed_message = w3.eth.account.sign_message(
    message_hash, private_key=USERS_PRIVATE_KEY
)
users_address = w3.eth.account.recover_message(message_hash, signature=users_signed_message.signature)


# Print the signature components
print("Owner's signed message signature: ", owners_signed_message.signature)
print("Owner's address: ", owner_address)


print("Users's signed message signature: ", users_signed_message.signature)
print("Users's address: ", users_address)
```

Now if we run this script, we will receive an output like this:

```
Owner's signed message signature:  b"BEm\xd9SZ'\x98\xb85w\x17.....4(h\x81I\xf0\xacN\xf9\x91\xbeU[T\xd1\x93\xb5\xd0\x00\xa0\xbf3\x81\x1d\xa8.\x1b"
Owner's address:  <OWNER-ADDRESS>

Users's signed message signature:  b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL.....\x87x9a\x95w0\xa5\x1e\x9aMU\x87\xba\xd1Q\x9a\xfa\xa3\x8d-!!\x1c'
Users's address:  <USER-ADDRESS>
```

Getting signed message signature

You can copy the highlighted text which is the OWNER's signed message signature as shown below:

```
Owner's signed message signature:  b"BEm\xd9SZ'\x98\xb85w\x17.....4(h\x81I\xf0\xacN\xf9\x91\xbeU[T\xd1\x93\xb5\xd0\x00\xa0\xbf3\x81\x1d\xa8.\x1b"
Owner's address:  <OWNER-ADDRESS>

Users's signed message signature:  b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL.....\x87x9a\x95w0\xa5\x1e\x9aMU\x87\xba\xd1Q\x9a\xfa\xa3\x8d-!!\x1c'
Users's address:  <USER-ADDRESS>
```

If you paste it to the "Enter your message" you will see that the car will unlock for 60 seconds

```
Enter your Message: BEm\xd9SZ'\x98\....xf\xe4(h\x81I\xf0\xacN\xf9\x91\xbeU[T\xd1\x93\xb5\xd0\x00\xa0\xbf3\x81\x1d\xa8.\x1b
User <OWNER-ADDRESS> trying to unlock the car
Checking if user <OWNER-ADDRESS> is current renter
User <OWNER-ADDRESS> is current renter
Car is unlocked for 60 seconds
Car is locked again.
```

If you paste the USER's signed message signature you will see that the car will not unlock since the USER is not the current renter yet.

```
Enter your Message: \x1a\xea\xee;a9\n\xc2\xd3\x86\x...VBU\xb2,\xbf\xe7z\x13[}\x9a\x95w0\xa5\x1e\x9aMU\x8\xba\xd1Q\x9a\xfa\xa3\x8d-!!\x1c
User <USER-ADDRESS> trying to unlock the car
Checking if user <USER-ADDRESS> is current renter
User <USER-ADDRESS>  is not current renter
```

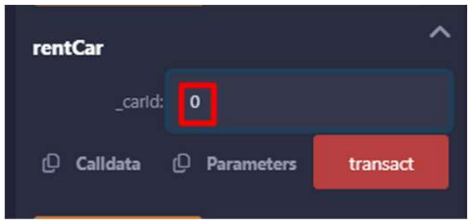
Renting Car

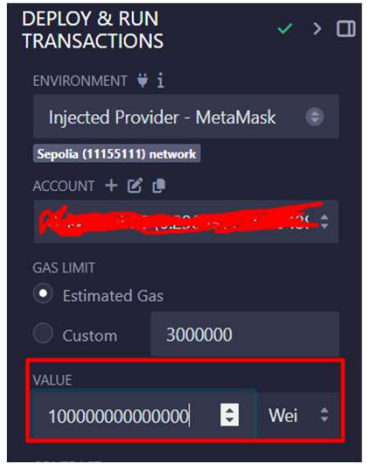
To rent the car, we need to make the car AVAILABLE as the `OWNER` of the car, so back in Remix IDE, in the `ICR SC`, call the function `changeCarStatusOwner` with `Car ID = 0` to change the car's status to AVAILABLE.

Make sure you use the `OWNER's address`, and that you already funded the microcontroller address of the car with some ETH otherwise the transaction will revert.

Next change your account to the USER and rent the car by calling the `rentCar` function with `Car ID = 0.`

Remember that this function is payble so add `100000000000000 wei` (`0.0001 ETH`, minimum car renting price) to the VALUE field on Deploy and Run Transactions sidebar.





Renting Car (2)

Now that the car is rented by the USER, try to unlock it by pasting the signed message signature of the USER (highlighted) as message to the microcontroller.

```
Owner's signed message signature:  b"BEm\xd9SZ'\x98\xb85w\x17.....4(h\x81I\xf0\xacN\xf9\x91\xbeU[T\xd1\x93\xb5\xd0\x00\xa0\xbf3\x81\x1d\xa8.\x1b"
Owner's address:  <OWNER-ADDRESS>

Users's signed message signature:  b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL.....\x87x9a\x95w0\xa5\x1e\x9aMU\x87\xba\xd1Q\x9a\xfa\xa3\x8d-!!\x1c'
Users's address:  <USER-ADDRESS>
```

You will see that the car successfully unlocks to the current renter.

```
Enter your Message: \x1a\xea\xee;a9\n\xc2\xd3\x8...6IVBU\xb2,\xbf\xe7z\x13[}\x9a\x95w0\xa5\x1e\x9aMU\x87\xba\xd1Q\x9a\xfa\xa3\x8d-!!\x1c
User <USER-ADDRESS> trying to unlock the car
Checking if user <USER-ADDRESS> is current renter
User <USER-ADDRESS> is current renter
Car is unlocked for 60 seconds
Car is locked again.
```

Try it with the OWNER and see that that car is staying locked.

Microcontroller makes a transaction

Now, when a periodic check happens (every 3min – half the renting time) you will see the following log:

```
Checking car status
Car status is Occupied. Enabling Change Car Status
Waiting X seconds
```

The microcontroller detected that the car is occupied.

`X = (time of last rent) + (renting time) - (latest block timestamp) + 120 second` (to ensure that a block with the correct timestamp has been minted)

Then, it will log something similar to this:

```
Starting transaction to change car's status
Transaction successful with hash: <TRANSACTION-HASH>
Status Changed
```

This means that a transaction was made that changed the car status from our microcontroller's account. We also received the transaction hash.

The process will continue until the microcontroller stops.

Outro

This concludes our Smart Contract and IoT course!