LESSON MC2 – Create Account

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 continue building our ICR project.

The microcontroller should be responsible of doing these things:

1) Create an account once activated
2) Find the ICR SC and the car that is registered to
3) Unlock the car door for the current renter of the car
4) Change the car status in the blockchain once the renting time is over

We will start from the account creation.

Remember that we assumed that IIoTC does not have access to the microcontroller once installed in the car so by creating an account inside the car no one would know its pk. The microcontroller could reveal its address to the owner and then the owner can send it along with the car's price to the IIoTC so that it can be registered in the ICR SC.

The owner of the car should also be responsible to fund the microcontroller with ETH to make transactions.

Creating `main.py`

We will keep things simple and create an entry point called `main.py` that will connect the rest of our logic together.

As many of the processes are asynchronous, we will use `asyncio` for this script.

We will also use the `load_json` that we created in the previous lesson as we will interact with both `ICRFactory` and `ICR` SCs. From remix, compile them, take their `ABIs` and create two files called `factory_abi.json` and `icr_abi.json` to store them.

We will also use the `connect_to_blockchain` script from the previous lesson.

Lastly, we will also use `dotenv` to store the generated address and pks as well as information from the ICR SC.

```python
from os import path
from dotenv import load_dotenv
import os
import asyncio

import load_json
from https_connection_to_bc import https_connection_to_bc # previously named connect_to_bc

async def main():
    pass

if __name__ == "__main__":
    asyncio.run(main())
```

Explaining `main.py`

Here is how our `main` function will operate:

1) Load environment variables
2) Check if microcontroller's address and pk exists
   - If not create, save and load them
3) Synchronize with `ICR` SC (if needed)
   - Get important information such as the `Car` and `ICR` it is registered to and load them
4) Do these two at the same time:
   - Periodically check if car is occupied (and change status to available if needed)
   - Wait for a signed message from a user to unlock the car

```
async def main():
    # Load environment variables from .env
    # Check if the .env file exists and has PUBLIC_ADDRESS and PRIVATE_KEY
        # Create an address and private key
        # If account already exists, load it
    # Synchronize with ICR Contract
    # At the same time
        # Peridically check if car is occupied (and change status to available if needed)
        # Wait for a signed message from a user to unlock the car
```

Account Creation


```python
async def main():
    # Load environment variables from .env
    load_dotenv(override=True)
    env_file = ".env"
    # Check if the .env file exists and has PUBLIC_ADDRESS and PRIVATE_KEY
    if not path.exists(env_file) or not os.getenv("MC_ADDRESS") or not os.getenv("MC_PRIVATE_KEY"):
        print("No account found. Creating a new Ethereum account...")
        # Create an address and private key
        create_and_save_account(env_file)
        load_dotenv(override=True)
    else:
        # If account already exists, load it
        MC_ADDRESS = os.getenv("MC_ADDRESS")
        print("Account already exists:")
        print(f"Public Address: {MC_ADDRESS}")
```

So , we load our environments, and we check if the `address` and pk exists.

We can now build the `create_and_save_account` function.

If you want, you can build it in the `main.py` file but I will be building it on its own file to keep the project clean.

I will also need to `import` the file on top of `main.py`

```python
from create_account import create_and_save_account
```

Account Creation (2)

DISCLAIMER: This example for key generation is only for the purposes of this lesson and should not be used in production. There are more secure alternatives you could use such as secure elements (e.g. ATECC608A), hardware wallets (e.g. Trezor) and encrypted storage with random number generators (e.g. HWRNG).

But for the purposes of this lesson, we will go with a library that comes with `web3.py`.

```
from eth_account import Account
```

In this function we are going to create a new `account` with an `address` and a pk and store them on the `.env` file.

```
def create_and_save_account(env_file=".env"):
    # Create a new Ethereum account
    account = Account.create()


    public_address = account.address
    private_key = account.key.hex()

    print("New Ethereum account created:")
    print(f"Public Address: {public_address}")
    # print(f"Private Key: {private_key}") # NEVER DO THIS <<------

    # Save the account to the .env file
    set_key(env_file, "MC_ADDRESS", public_address)
    set_key(env_file, "MC_PRIVATE_KEY", private_key)
    print(f"Account saved to {env_file}. Keep this file secure!")
```

Account Creation (3)

You can test the script seperately, if you want, by typing `python3 <your-script-name>` in the console.

```python
from eth_account import Account
from dotenv import load_dotenv, set_key
# Load existing environment variables
load_dotenv()

def create_and_save_account(env_file=".env"):
    # Create a new Ethereum account
    account = Account.create()
    public_address = account.address
    private_key = account.key.hex()
    print("New Ethereum account created:")
    print(f"Public Address: {public_address}")
    # print(f"Private Key: {private_key}") # NEVER DO THIS <<------
    # Save the account to the .env file
    set_key(env_file, "MC_ADDRESS", public_address)
    set_key(env_file, "MC_PRIVATE_KEY", private_key)
    print(f"Account saved to {env_file}. Keep this file secure!")

if __name__ == "__main__":
    env_file = ".env"  # Define the .env file location
    create_and_save_account(env_file)
```

Outro

Next let's go over synchronizing our microcontroller to the SCs.