LESSON MC4 – Unlock Car

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 create the unlock car script. Here is how it will work:

1) User signs a message
2) The microcontroller will recover the address of this message
3) Then it will check the car in the SC to see who the current renter is
4) If the renter's address is the same as the recovered address the car will be unlocked for one minute.

Creating `unlock_car_by_message.py`

For this script, we will use libraries similar to the ones we already used.

To prevent errors with multiple coroutines we will use the `async lock` feature. (This could be manipulated by users of the application, so it is just for this example.)

We are going to define a message we want users to sign in order to unlock the car.

The message will be "Unlock Car" and we will use its hashed version.

Let's also build the hypothetical `unlock_car` logic. It is going to be a simple logic that prints "`Car is unlocked`" and `sleep` for 60 seconds, then it will print "`Car is locked`".

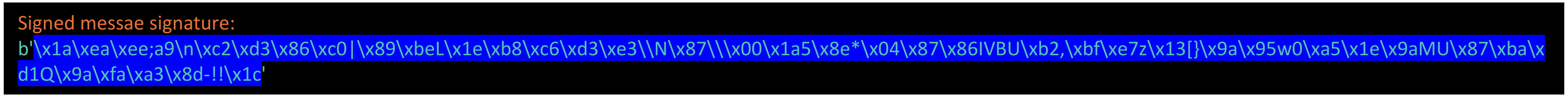
```
import asyncio
from eth_account.messages import encode_defunct

lock = asyncio.Lock()
message = "Unlock Car" # Message
message_hash = encode_defunct(text=message) # Hashed Message


async def unlock_car():
    async with lock:
        try:
            print("Car is unlocked for 60 seconds")
            await asyncio.sleep(60)  # Wait for 60 seconds
        except asyncio.CancelledError:
            print("Unlocking was interrupted before 60 seconds.")
        finally:
            print("Car is locked again.")
```

Build the address extraction logic

Then we are going to build a function that waits for input of the user.

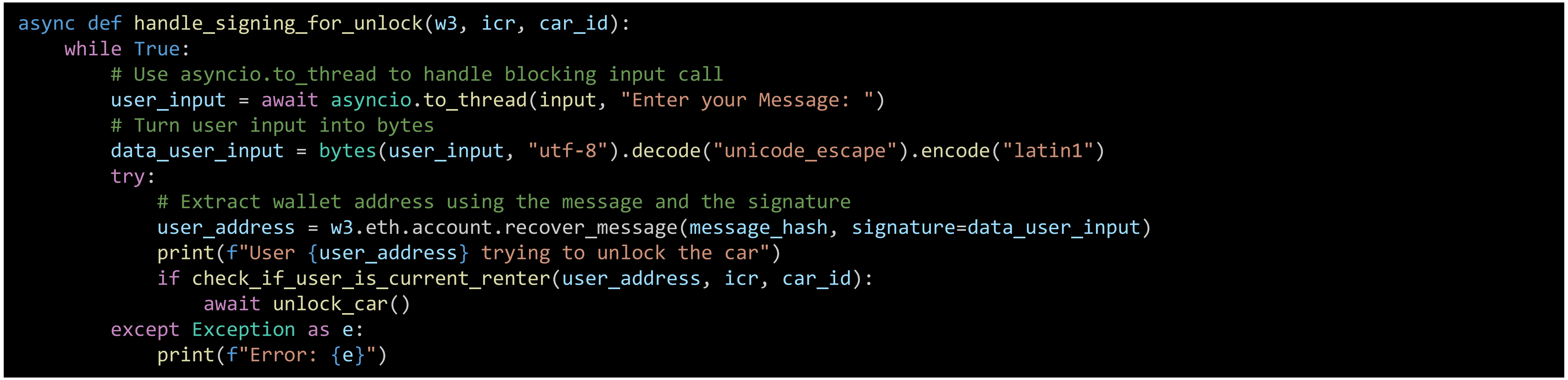
If the input is the signed message signature, it will then convert it into bytes and extract the address that signed it, using the pre-defined message hash.

We saw a sign message signature in the intro to web3.py lesson and it looks like this:



```
Signed messae signature:
b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL\x1e\xb8\xc6\xd3\xe3\\N\x87\\\x00\x1a5\x8e*\x04\x87\x86IVBU\xb2,\xbf\xe7z\x13[}\x9a\x95w0\xa5\x1e\x9aMU\x87\xba\x
d1Q\x9a\xfa\xa3\x8d-!!\x1c'
```

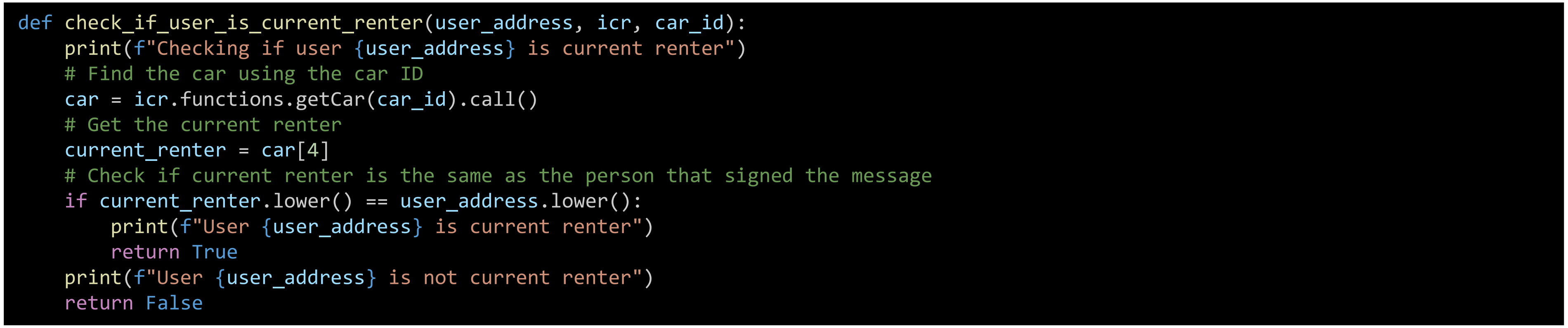
Then, it will call a function that checks if the recovered address is the same as the address of the current renter. If there is a match, the unlock_car logic will run.



```python
async def handle_signing_for_unlock(w3, icr, car_id):
    while True:
        # Use asyncio.to_thread to handle blocking input call
        user_input = await asyncio.to_thread(input, "Enter your Message: ")
        # Turn user input into bytes
        data_user_input = bytes(user_input, "utf-8").decode("unicode_escape").encode("latin1")
        try:
            # Extract wallet address using the message and the signature
            user_address = w3.eth.account.recover_message(message_hash, signature=data_user_input)
            print(f"User {user_address} trying to unlock the car")
            if check_if_user_is_current_renter(user_address, icr, car_id):
                await unlock_car()
        except Exception as e:
            print(f"Error: {e}")
```

Build the `address` comparison logic

Next, we build the address comparison logic which will check the ICR SC for the Car with the correct Car ID and get current renter. It will then check if current renter is the same as the one that signed the message. The function will return `true` or `false` depending on the result.

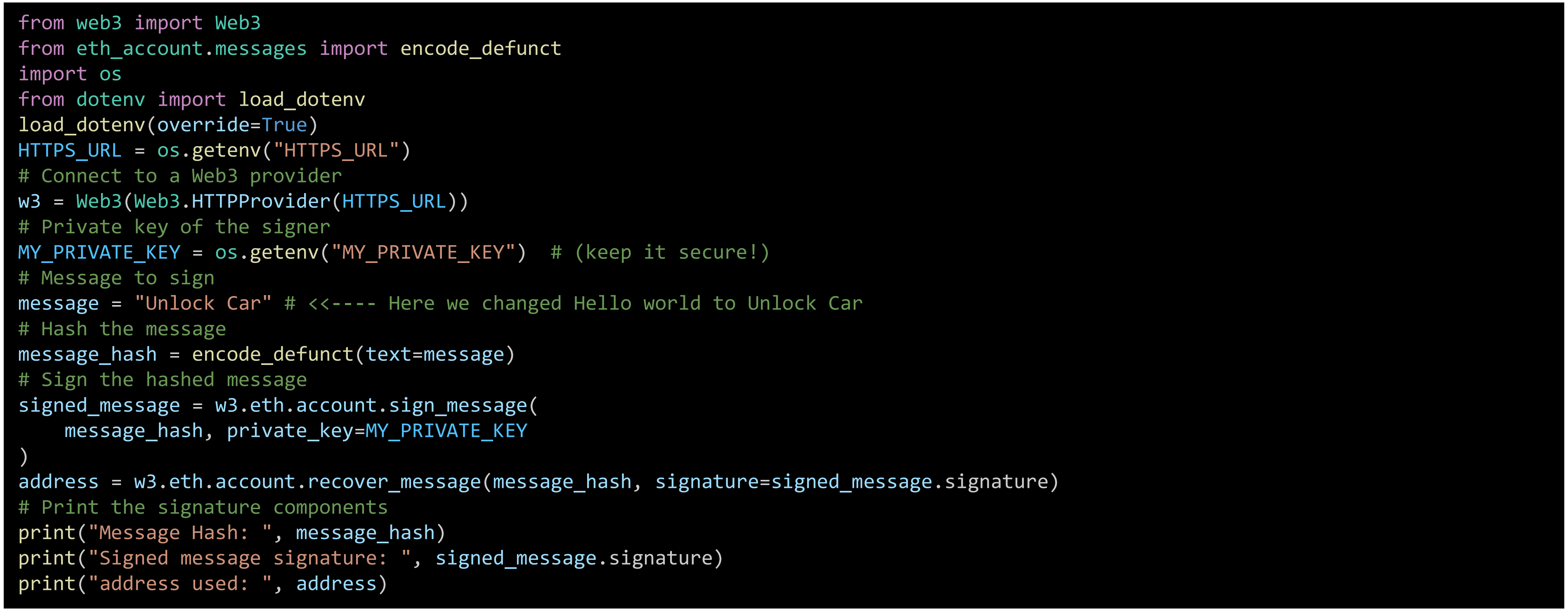


```python
def check_if_user_is_current_renter(user_address, icr, car_id):
    print(f"Checking if user {user_address} is current renter")
    # Find the car using the car ID
    car = icr.functions.getCar(car_id).call()
    # Get the current renter
    current_renter = car[4]
    # Check if current renter is the same as the person that signed the message
    if current_renter.lower() == user_address.lower():
        print(f"User {user_address} is current renter")
        return True
    print(f"User {user_address} is not current renter")
    return False
```

What input should we use? (1)

We can easily get the needed input by using the `signing_message.py` example that we created in the introduction to `web3.py` lesson. We can change the message `"Hello world"` to `"Unlock Car"` and run it.

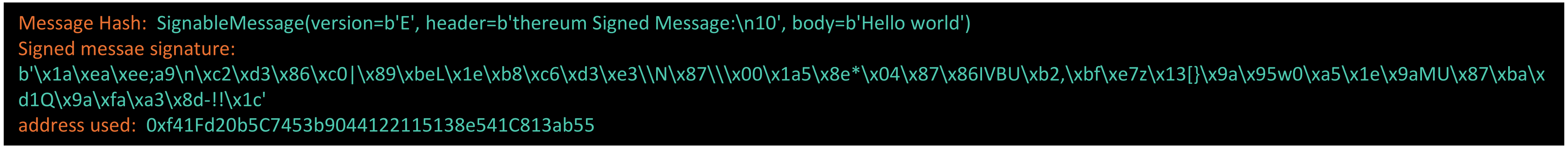
Make sure you add the user's pk in the `.env` file that will rent the car later.



```
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 signer
MY_PRIVATE_KEY = os.getenv("MY_PRIVATE_KEY")  # (keep it secure!)
# Message to sign
message = "Unlock Car" # <<---- Here we changed Hello world to Unlock Car
# Hash the message
message_hash = encode_defunct(text=message)
# Sign the hashed message
signed_message = w3.eth.account.sign_message(
    message_hash, private_key=MY_PRIVATE_KEY
)
address = w3.eth.account.recover_message(message_hash, signature=signed_message.signature)
# Print the signature components
print("Message Hash: ", message_hash)
print("Signed message signature: ", signed_message.signature)
print("address used: ", address)
```

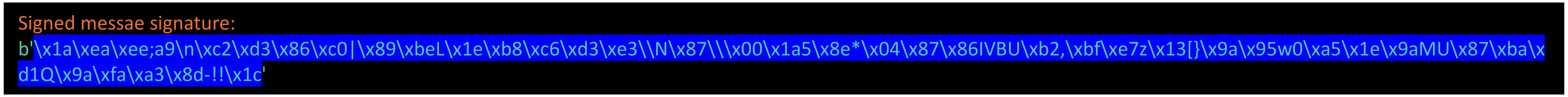
What input should we use? (2)

Now if we run this script, we will receive something similar to this:



```
Message Hash:  SignableMessage(version=b'E', header=b'thereum Signed Message:\n10', body=b'Hello world')
Signed messae signature:
b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL\x1e\xb8\xc6\xd3\xe3\\N\x87\\\x00\x1a5\x8e*\x04\x87\x86IVBU\xb2,\xbf\xe7z\x13[}\x9a\x95w0\xa5\x1e\x9aMU\x87\xba\x
d1Q\x9a\xfa\xa3\x8d-!!\x1c'
address used:  0xf41Fd20b5C7453b9044122115138e541C813ab55
```

Look at the following:



```
Signed messae signature:
b'\x1a\xea\xee;a9\n\xc2\xd3\x86\xc0|\x89\xbeL\x1e\xb8\xc6\xd3\xe3\\N\x87\\\x00\x1a5\x8e*\x04\x87\x86IVBU\xb2,\xbf\xe7z\x13[}\x9a\x95w0\xa5\x1e\x9aMU\x87\xba\x
d1Q\x9a\xfa\xa3\x8d-!!\x1c'
```

We can add the highlighted message as input to the `unlock_car_by_message.py` script to check if our function works, but we will do it later in an example run.

In a real application we would use a wallet for signing messages like MetaMask instead of using a script to get the signed message signature. But for the purpose of this example this will work well.

Outro

Next let's go over the script responsible for changing the status of the car.