Internet of Blockchain of Things (IoBoT)

Duration of the project:

4 months (December 2021- March 2022)

Project Mentors:

Tanmay Ranaware, Shalaka Uday Deshpande

Team Members:

Pranav Agarwal, Ram Aniruth, Saniya Bhargava, P M Prasanna, Saathwik T K

Introduction

Project statement /Aim:

To implement MQTT communication, use blockchain for the authentication (using OTPs Refresh tokens) and integrate the two together.

Literature Review:

The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

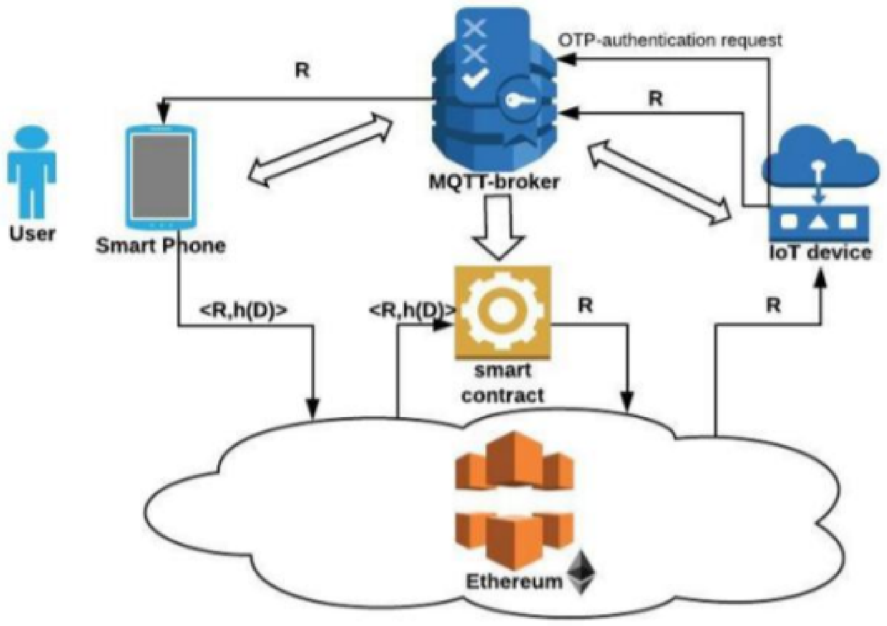
Blockchain on the other hand is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

MQTT has established itself as the industry standard for IoT connectivity. The MQTT protocol consumes less bandwidth, requires less computational power, and transmits data quickly. As a result, constrained devices can use the MQTT protocol. However, by default, the MQTT protocol lacks a security mechanism. To solve this problem, we planned to leverage blockchain to establish authentication using one-time passwords (OTPs) that expire after a certain period of time or refresh tokens that are changed after each connection. The Benefit of IoT and blockchain is that you build trust in your IoT data because each transaction is recorded, put into a data block, and added to a secure, immutable data chain that cannot be changed — only added to.

Methodology

In our project, we are trying to solve the authentication issue in MQTT communication. The step-by-step process for the same is as follows.

1. The user will send a request to the MQTT-broker.
2. The IoT application running on the MQTT-broker will generate an OTP using TRNG (True Random Number Generation) and store it along with the created time.
3. The User will receive the OTP through a standard MQTT message.
4. The subscribed User can now send the OTP to the MQTT-broker through a standard MQTT message.
5. The IoT application will check if the OTP is valid and it will find the time elapsed from OTP generation.
6. If the OTP is invalid or the time elapsed crosses a certain number the IoT device will refuse the connection.



We divided this project into 3 steps:

1. Blockchain: We created a smart contract for OTP authentication using solidity. The smart contract has 3 main functions, i.e., generateOtp(), getAuthenticationToken() and validateOtp(). Initially we tested it on the

localhost and after success we decided to test it on the ropsten test network for more validation.

2. IoT: We hosted an MQTT broker on the cloud server and we were able to publish and subscribe to topics and send messages across raspberry pi and laptop located remotely (clients)

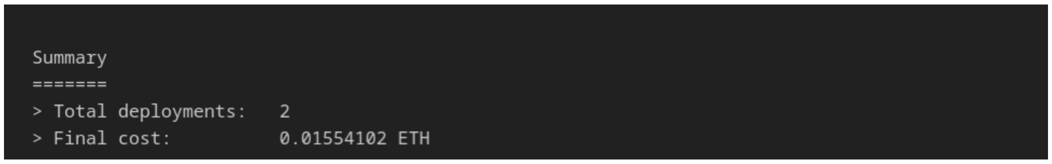
3. Integration: We used the web3.py package to interact with the smart contract which is deployed on the RaspberryPi's localhost. We used the abi generated by truffle and the contract address after deployment to connect to the contract and then created 3 functions (generateOtp(), getAuthenticationToken() and validateOtp()) which can be called by the MQTT client directly using python. We call the generateOtp() and getAuthenticationToken() functions when the user subscribes to a topic and we send the OTP to the user. The user can then publish the OTP and upon receiving a message we validate the OTP received in the payload.

Results

Starting off with creating an **OTP authentication** smart contract using the solidity programming language, we gradually moved on to implement tests for the following :

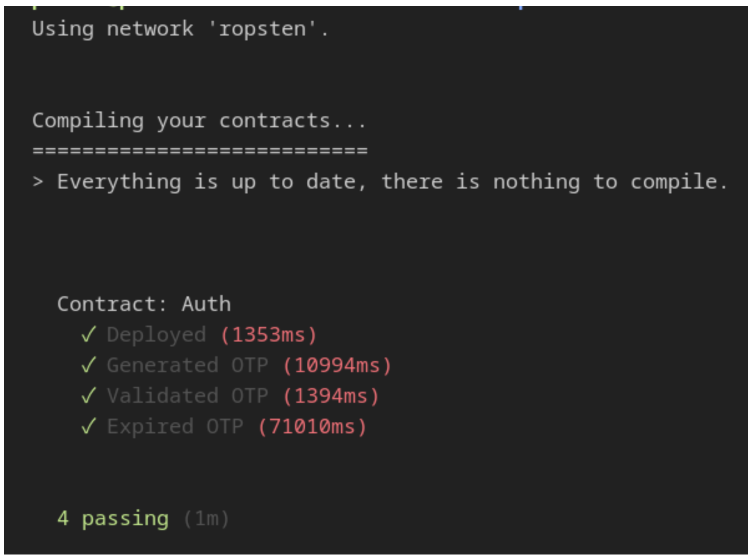
1. OTP creation
2. OTP validation
3. OTP expiration

The smart contract takes 0.01554102 ETH to get deployed.



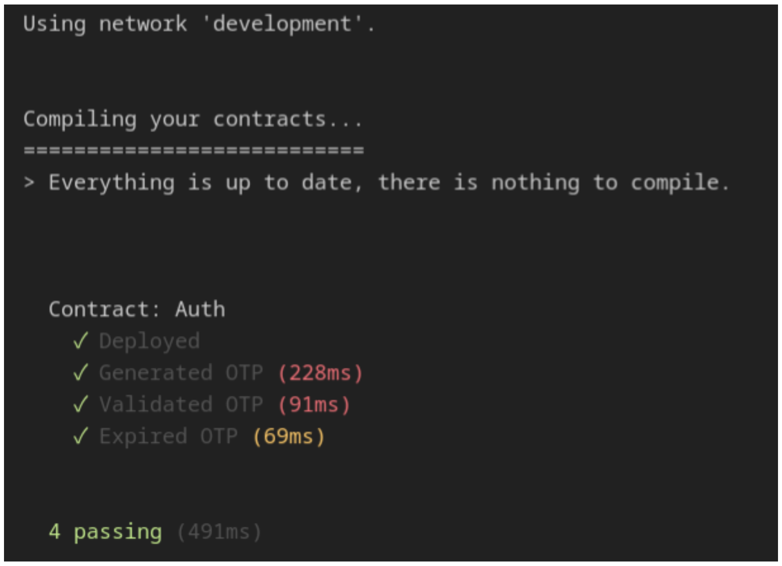
```
Summary
=======
> Total deployments:   2
> Final cost:          0.01554102 ETH
```

The smart contract was then put into the **Ropsten test network**, where we executed the above tests on it. The final result was all of the tests were deployed successfully on the smart contract.



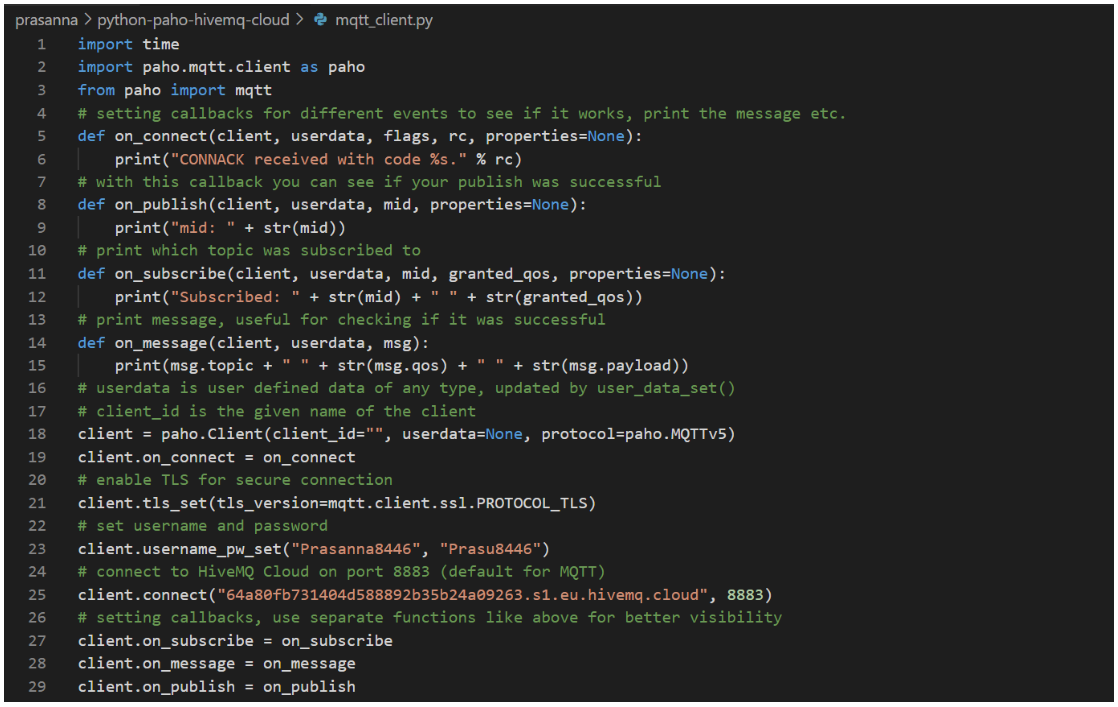
```
Using network 'ropsten'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: Auth
    ✓ Deployed (1353ms)
    ✓ Generated OTP (10994ms)
    ✓ Validated OTP (1394ms)
    ✓ Expired OTP (71010ms)


  4 passing (1m)
```

The contract was then deployed on the localhost for better performance. The complete process takes around 400-500ms on average.

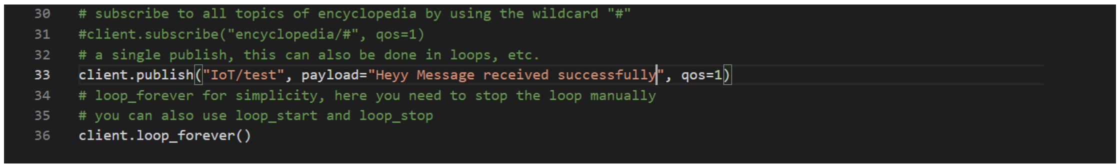


```
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: Auth
    ✓ Deployed
    ✓ Generated OTP (228ms)
    ✓ Validated OTP (91ms)
    ✓ Expired OTP (69ms)


  4 passing (491ms)
```

We were able to publish and subscribe to topics and send messages across clients located remotely via a broker which is hosted on a cloud server

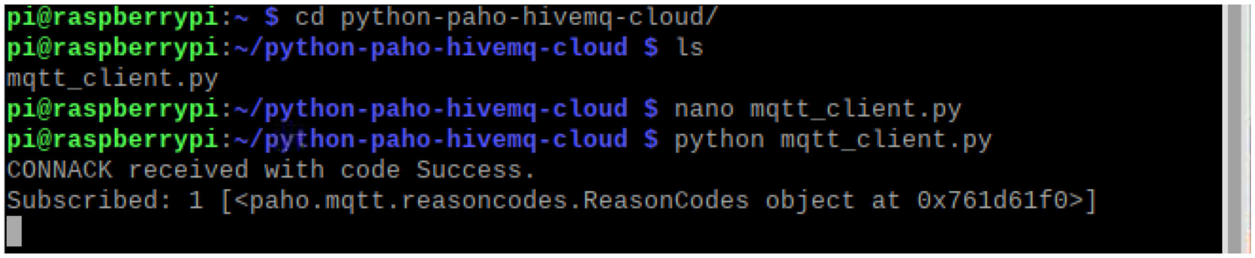


```python
import time
import paho.mqtt.client as paho
from paho import mqtt
# setting callbacks for different events to see if it works, print the message etc.
def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)
# with this callback you can see if your publish was successful
def on_publish(client, userdata, mid, properties=None):
    print("mid: " + str(mid))
# print which topic was subscribed to
def on_subscribe(client, userdata, mid, granted_qos, properties=None):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))
# print message, useful for checking if it was successful
def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))
# userdata is user defined data of any type, updated by user_data_set()
# client_id is the given name of the client
client = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv5)
client.on_connect = on_connect
# enable TLS for secure connection
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
# set username and password
client.username_pw_set("Prasanna8446", "Prasu8446")
# connect to HiveMQ Cloud on port 8883 (default for MQTT)
client.connect("64a80fb731404d588892b35b24a09263.s1.eu.hivemq.cloud", 8883)
# setting callbacks, use separate functions like above for better visibility
client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_publish = on_publish
```



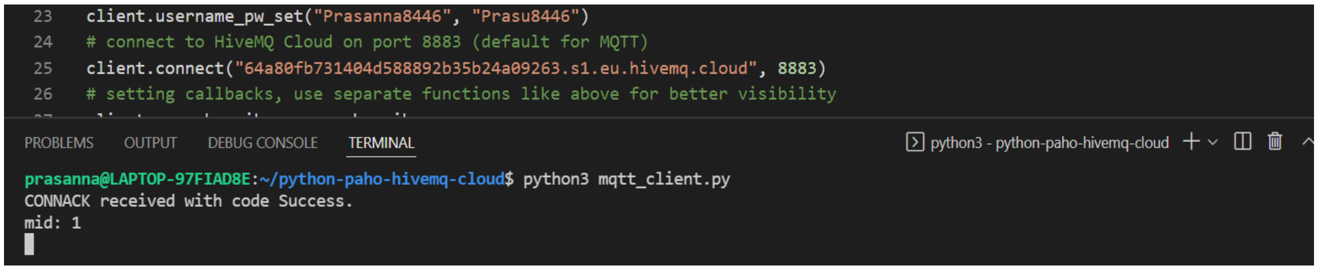
```python
# subscribe to all topics of encyclopedia by using the wildcard "#"
#client.subscribe("encyclopedia/#", qos=1)
# a single publish, this can also be done in loops, etc.
client.publish("IoT/test", payload="Heyy Message received successfully", qos=1)
# loop_forever for simplicity, here you need to stop the loop manually
# you can also use loop_start and loop_stop
client.loop_forever()
```

The below image shows the first client(Raspberry pi) is subscribed to a topic and waiting for the client2 to publish a message



```
pi@raspberrypi:~ $ cd python-paho-hivemq-cloud/
pi@raspberrypi:~/python-paho-hivemq-cloud $ ls
mqtt_client.py
pi@raspberrypi:~/python-paho-hivemq-cloud $ nano mqtt_client.py
pi@raspberrypi:~/python-paho-hivemq-cloud $ python mqtt_client.py
CONNACK received with code Success.
Subscribed: 1 [<paho.mqtt.reasoncodes.ReasonCodes object at 0x761d61f0>]
```

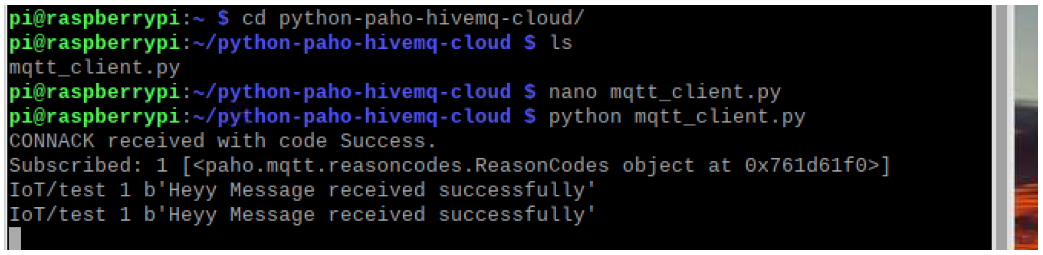
Client 1(laptop) published on the topic IoT/test



```python
client.username_pw_set("Prasanna8446", "Prasu8446")
# connect to HiveMQ Cloud on port 8883 (default for MQTT)
client.connect("64a80fb731404d588892b35b24a09263.s1.eu.hivemq.cloud", 8883)
# setting callbacks, use separate functions like above for better visibility
```

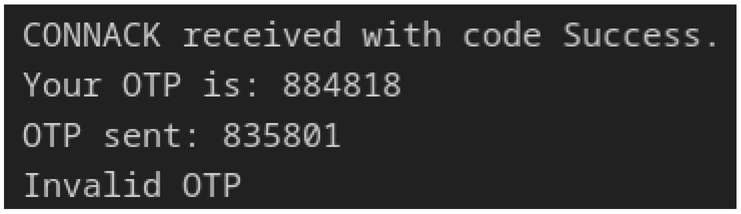
```
prasanna@LAPTOP-97FIAD8E:~/python-paho-hivemq-cloud$ python3 mqtt_client.py
CONNACK received with code Success.
mid: 1
```

And the message is received on client 1



```
pi@raspberrypi:~ $ cd python-paho-hivemq-cloud/
pi@raspberrypi:~/python-paho-hivemq-cloud $ ls
mqtt_client.py
pi@raspberrypi:~/python-paho-hivemq-cloud $ nano mqtt_client.py
pi@raspberrypi:~/python-paho-hivemq-cloud $ python mqtt_client.py
CONNACK received with code Success.
Subscribed: 1 [<paho.mqtt.reasoncodes.ReasonCodes object at 0x761d61f0>]
IoT/test 1 b'Heyy Message received successfully'
IoT/test 1 b'Heyy Message received successfully'
```

We then created a python script to interact with the contract and then called the generateOTP, getAuthenticationToken and validateOTP functions in the MQTT client and were able to authenticate the user. We sent a subscribe request to a topic and received the OTP after which we published the OTP to another topic. Later, we also sent an invalid OTP to the MQTT broker after which we were disconnected from the MQTT connection.



```
CONNACK received with code Success.
Your OTP is: 884818
OTP sent: 835801
Invalid OTP
```

Obstacles faced:

IoT:

- We weren't able to connect the locally installed MQTT broker with the client (When these were in different devices) .Got error messages like broken pipe , disconnected from server etc.

  Solution-
  We tackled the issue by hosting the MQTT broker on a cloud server (HiveMqttcloud) and also used Python Paho MQTT ,which is a client library under the Eclipse Paho project, which can connect to MQTT Broker to publish messages, subscribe to topics and receive the Published message.

Blockchain:

- Because it was hosted on localhost using ganache and we were using "block.timestamp" to acquire the current time, the smart contract for OTP authentication was unable to correctly check the OTP expiry because the block remained the same and we always got the same time.

  Solution-
  The contract was deployed on the ropsten test network, which solved the problem. The OTP expiry worked properly because the test network continued to mine fresh blocks, giving us a near-accurate time.

- Running the contract on Ropsten later turned out to be slow and inefficient.

  Solution-
  We changed our approach slightly by deploying the contract on the localhost and modifying the smart contract. Now, we took the timestamp directly from the IoT device which ensured that the contract not only worked fast, but also did so on the local host itself.

Conclusion

The MQTT communication and the smart contract for OTP authentication are functioning properly and once integrated we will have a strongly authenticated MQTT communication protocol which can be utilized in any IoT device in the future.

Future work

We plan on using refresh tokens (credentials used to obtain access tokens) instead of OTPs given the fact that OTP generation is a very primitive solution. Also the refresh tokens will reduce the number of OTP generation requests by keeping the user signed in.

References

IoT:

Communication between Raspberry Pi and PC using MQTT
https://www.kitflix.com/communication-between-raspberry-pi-and-pc-using-mqtt
How to Use MQTT with Raspberry Pi
https://www.emqx.com/en/blog/use-mqtt-with-raspberry-pi
HiveMQ tutorial
https://www.youtube.com/watch?v=y6PFe3YDr2g
What is IOT?
https://www.javatpoint.com/iot-internet-of-things
https://www.youtube.com/watch?v=6mBO2vqLv38
IOT Architecture
https://www.youtube.com/watch?v=KeaeuUcw02Q
How does MQTT work
https://www.youtube.com/watch?v=EIxdz-2rhLs
https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

Blockchain:

Learn solidity Programming
https://cryptozombies.io/
How to deploy Smart contracts
https://medium.com/openberry/deploying-smart-contracts-with-truffle-1c056b452cde
How to deploy Smart contracts to Ropsten test network
https://jianjye.com/articles/14/how-to-deploy-ethereum-smart-contract-to-ropsten-testnet-via-infura
Research Papers used for references
https://drive.google.com/file/d/1RJQedQO16fvVUNZHIlhe-IFvfitDTSpQ/view?usp=sharing
https://drive.google.com/file/d/1WziYCn0hD80h-aecd2Lj2AaeoIjIyNXz/view?usp=sharing
For interacting with the contract with python
https://web3py.readthedocs.io/en/stable/contracts.html