BLOCKCHAIN AT MCGILL

Oraclize

Implementing Oracles into Solidity Smart Contracts

BLOCKCHAIN AT MCGILL

Oracle

A person or entity regarded as an infallible authority or guid on a topic.

Oracle - Smart Contracts

In blockchain, all information that is on-chain is “trusted” information, i.e. verified through consensus.

But you often need access to information that is not on the blockchain to execute functions/functionalities in your smart contract.

How do you get this information?

Oracle - Smart Contracts

E.g. BTC/USD Price pair at a specific time to trigger an action.

BTC/USD price pair is not available from a trusted source, i.e. internally to the ethereum blockchain, so one must go to external sources to get this information.

Oracle - Smart Contract

You use oracles to gather this information!

But to rely on a new trusted intermediary, the oracle in this case, it would be betraying the security and reduced-trust model of blockchain applications!

So it is important to consider security and trust when considering a source as an oracle, how can we do this reliably and securely?

Oracle Workflow

Smart contract -> Calls for information from oracle

Smart Contract -> Pays for information to the oracle

Oracle -> returns information to contract

Smart Contract -> Acts on data (or not!)

BLOCKCHAIN AT MCGILL

Oraclize - Our Friendly Oracle Service

Oraclize is a very easy implementable oracle service for blockchain applications.

It supports the following blockchains:

Ethereum, Rootstock, R3 Corda, Hyperledger Fabric and EOS.

Oraclize - Interfacing with Ethereum and Smart Contracts

1. Importing the Oraclize API Library:
   a. The following code must be added on top of any Smart Contract that wishes to use Oraclize as its Oracle Provider

```
import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";
```

Oraclize - Interfacing with Ethereum and Smart Contracts

2. Using Inheritance to make sure that the correct functions are usable within the contract:

a. The following code must be added next to the contract header:

```
contract ExampleContract is usingOraclize {
```

b. Your smart contract is now ready to use Oraclize on the Ethereum Mainnet or the Rinkeby/Ropsten/Kovan Testnets

Oraclize - Interfacing with Ethereum and Smart Contracts

2. All queries through Oraclize require the “__callback” function:

    a. The following function must be added to any contract that makes Oraclize queries:

```
function __callback(bytes32 myid, string result) {
        if (msg.sender != oraclize_cbAddress()) revert();
            //do something
        }
```

BLOCKCHAIN AT MCGILL

Lets Try It!

Please open the Shop Contract from Yesterday.

Oraclize - Basic Queries

How could you make sure that the data you receive from an oracle is trustworthy?

i.e. untampered, accurate and reliable

Solution 1: Decentralized Oracles

Accept information from multiple untrusted/sem-trusted parties and cross-reference.

Is this approach viable?

Decentralized Oracles

Although, this is a very good approach for shielding your contract from untrustworthy information, it is **expensive** and **computationally heavy.**

What other limitations can you think of for decentralized oracles?

Limitations - Decentralized Oracles

It requires a predefined standard on data format

It is inherently inefficient: all the parties participating will require a fee and, for every request, it will take time before reaching a sufficient number of answers.

Proof of Authenticity

Although decentralized oracles are a good option for implementing more trustworthy oracles, it is not often worth the effort given that each call has a price associated.

Oraclize provides a new approach to make sure that data is untampered when a call is made. They call this verification Proof of Authenticity.