CRYPTOBOX

An application that fingerprints documents
into the blockchain. A solution developed in Python,
with a smart contract built up in Solidity.

January 2020

Authors	Student ID
Fernando Rey Gaido	19-601-004
Federico Cannes	19-600-907

Table of Contents

1. Overview

One of the major inconveniences we find in everyday life, is trust. We wanted to develop a real-world implementation that can help to provide trustworthy traceability to digital documents.

We have ideated **CryptoBox** as way of securing documents into the Ethereum Blockchain.

2. Basic concepts.

- HASH → A hash function is a mathematical function that converts an input value into a compressed numerical value – a hash or hash value. For the purpose of this work, we will refer to a HASH as a fixed-size bit string value of a file; a hexadecimal string which represents the file/document. We are using MD5 algorithim (*even that have known cryptographic vulnerabilities)* works perfect for this first approach.

- Blockchain → A Blockchain is a digital record of transactions. The name comes from its structure, in which individual records, called blocks, are linked together in single list, called a chain. Blockchains are used for recording transactions. For this project we will be using the Ethereum Blockchain to record transactions.

- Ethereum Blockchain → Ethereum is an open-source, blockchain-based, decentralized software platform used for its own cryptocurrency, ether. It enables SmartContracts to be built and run without any downtime, fraud, control, or interference from a third party. The Smart Contracts are run by/in the Ethereum Virtual Machine.

- Ethereum Virtual Machine (EVM) → The Virtual Machine is an essential part of the Ethereum protocol. For the pouropuse of this work, we will state that the EVM is a way of executing a Smart Contract in a decentralized way, where each 'validator' node runs the Contract and get a result. Once the majority get the same result, it is considered that the contract has been executed correctly (*it is a little bit more complex, but enough for the moment*).

- Ganache → Is a private Ethereum Blockchain client, used to run tests, execute commands, and inspect state while controlling how the chain operates. It gives you the ability to perform all actions you would do on the main chain without costs. We will use Ganache in order not to spend gas while executing our development.

3. How does CryptoBox work?

The program follows this logic: The user has one digital document and he want to be able to probe that

(i) in a specific moment that document existed, and
(ii) he was in possession of that document at that moment.

That's the only thing that can be proved in general, even with official registries or notaries. Imagine the case where someone stole another person invention and goes to a notary and make an act claiming that he's the creator. **CryptoBox** has the regular IP limitations that traditional systems have, but it's a simple, fast and cheap way to stamp every single digital document almost instantaneity, with a certain date and possession.

So, let's imagine that the CryptoBox user records a song, or writes an article, or digitally signs an agreement, or even zips his entire hard disk, and want to be sure that

(i) if anyone later on claim that they invented the same thing previously, he will be able to prove that he has it before (**timestamp**);
(ii) that if anyone modify such document, he can prove that the original file was the one that existed before (**integrity**).

What he will do is run **CryptoBox**, select the file he will want to SAVE.

CryptoBox will calculate the HASH of such document and will ask for extra information to prove that the user is the one uploading it (*Name and Email*).

Once the information is generated, the Python interface will record by calling the smart contract we deployed and will save such information into the Ethereum Blockchain.

That information will be saved under one specific Block and can be consulted later, but it cannot be modified. All the computers of the Blockchain will have the record of such transaction, and the user with the information obtained will be able eventually to demonstrate that the HASH of the document in his possession matches the one saved in such block of the Blockchain. The document entirely is not saved, only the HASH and the additional information introduced.

4. Coding Process

We have built a fully functional Back-End development and a really simple Front-End that will run in the console (*we didn't have time to make a full Front-End application, but eventually will be nice*).

We have developed this program using the following:

(i) Windows operating system,
(ii) Installed Python version 3.7.4,
(iii) SublimeText as Idle for the Python Programming; and
(iv) Remix interface for compiling the Smart Contract in Solidity https://remix.ethereum.org/.

We have installed the following libraries, by using the Command Prompt and running the standard package-management system PIP:

- JSON → https://docs.python.org/3/library/json.html (*Javascript Object Notation*)
- Web3 Library → https://www.npmjs.com/package/web3 (*interact with the Ethereum Blockchain*)
- Tkinter → https://tkdocs.com/tutorial/install.html (*Graphical User Interface)*
- Hashlib → Installed by default with the Python version (*Calculate HASH*)
- OS → Installed by default with the Python version (*we use it to clear the console screen*)

Also, for avoiding spending gas for this project we use a Ganache node. In order to run the program the user will need to download and install Ganache from: https://www.trufflesuite.com/ganache.

Once the Ganache node is running the user will have to update the IP references of the Python Code, and the address of the wallet that will be obtaining de gas from (*you can select any of the available ones*).

Once Ganache is running and the code updated, the user can run **CryptoBox.py** from the console. All the transactions made and interactions could be found in the "Transaction" tab of Ganache.

5. Coding Structure

We structured the code in three different parts:

- **Part 1** → Deploying the Smart Contract. We connect with the Ethereum/Ganache Blockchain and deploy the Smart Contract that we previously coded in the Remix interface.
- **Part 2** → Functions. We generate a function for every different action made in the Program. We avoid using global variables. Each function is explained, we use the same comments types in order to make any function self-explainable and independent from each other.
- **Part 3** → Front End. We use a simple "case" to interact with the user and loop in order to execute each of the functions coded.

6. Smart Contract

(We add the Solidity Smart contract we have created for your reference. All the information of coding in Solidity and how to use the structures were found from different tutorials online. This does not need to be run, the Smart Contract bytecode is embedded in the CryptoBox Python code and will be deployed into the Blockchain once executed.)

```
//We select the version the compiler. We use one of the latest releases, Solidity ^0.5.1
pragma solidity ^0.5.1;

//we start the contract
contract MyContract {
    //this will be our contract list. Is a public int that count how many contracts we
have. We start at 100 is the INDEX
    //of the first contract.
    uint256 public contractCounter = 100;
    //Solidity uses mapping structure value types, such as booleans, integers, addresses,
and structs. It consists of two main parts: a _KeyType and a _ValueType.
    //We create a mapping structure of Struct Ob
    mapping (uint => ObjectContract) public contractList;
    //We create the Struct that will hold the Information to be saved in the Blockchain
(Object / Type Contract)
    struct ObjectContract {
        uint contractNumber;
        string _name;
        string _email;
        string _hash;
    }
```

```
    //This function add one new CONTRACT to the list. We add 'name / email / hash' that the
user will introduce and we use the Contract Counter.
    function newContract(string memory _name  , string memory _email, string memory _hash)
public {
        contractCounter = contractCounter + 1;
        contractList[contractCounter] = ObjectContract(contractCounter, _name,
_email,_hash);
    }

    //By this function we search for the HASH in the list.
    function searchContract(string memory _search) view public returns (uint256) {
        //We loop searching threw the entire contractList
        for(uint256 i=99 ; i < contractCounter + 1  ; i++) {
            //In soloidity strings can't be compared, so we HASH them and compare hashes.
            if(keccak256(abi.encodePacked(_search)) ==
keccak256(abi.encodePacked(contractList[i]._hash))) {
                return i;
            }

        }
    }

    //We retrive the entire registry from the contractList, using the index.
    function viewContract(uint256 _num) view public returns (uint256, string memory, string
memory, string memory) {
        return (contractList[_num].contractNumber, contractList[_num]._name,
contractList[_num]._email, contractList[_num]._hash);
    }

    //Returns how many contract exists, this is in order to loop in the python program.
    function countContracts() view public returns (uint256) {
        return contractCounter;
    }
}
```

7. CryptoBox (Main Python Code)

```
#We use tkinter library as Graphical User Interfaces, in order to make the user select the
file to FingerPrint.
import tkinter as tk
from tkinter import filedialog
#Hashlib is used in order to calculate MD5 hash of the selected file (What we are saving in
)
import hashlib
#We use Json library in order to use Javascript Object Notation, the list of commands
defined as "ABI" define the properties and commands of the Smart Conctract.
import json
#We use Web3 as the library that allows us to interact with the Ethereum Blockchain.
from web3 import Web3
#We use this library to clear the screen --> os.system ("cls")
import os

#We use Ganache, an Etherum development to deploy contracts and run tests.  In order to run
the programm, ganache has to be download and open
#in the computer. The user need to update the line:address = web3.toChecksumAddress(""),
writing the first addres shown on ganache.
#https://www.trufflesuite.com/ganache
#update this link with the URL of your computer.
ganache_url = "HTTP://127.0.0.1:7545"

# Initialising a Web3 instance with an RPCProvider (GANACHE in this case)
web3 = Web3(Web3.HTTPProvider(ganache_url))
web3.eth.defaultAccount = web3.eth.accounts[0]
```

```
#Here we list the functions of the contract Application Binary Interface (ABI). Is the
standard way to interact with contracts in the Ethereum ecosystem,
abi =
json.loads('[{"constant":true,"inputs":[],"name":"contractCounter","outputs":[{"internalTyp
e":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"
function"},{"constant":true,"inputs":[{"internalType":"uint256","name":"","type":"uint256"}
],"name":"contractList","outputs":[{"internalType":"uint256","name":"contractNumber","type"
:"uint256"},{"internalType":"string","name":"_name","type":"string"},{"internalType":"strin
g","name":"_email","type":"string"},{"internalType":"string","name":"_hash","type":"string"
}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[]
,"name":"countContracts","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],
"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"i
nternalType":"string","name":"_name","type":"string"},{"internalType":"string","name":"_ema
il","type":"string"},{"internalType":"string","name":"_hash","type":"string"}],"name":"newC
ontract","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"c
onstant":true,"inputs":[{"internalType":"string","name":"_search","type":"string"}],"name":
"searchContract","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"payable
":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"internalTy
pe":"uint256","name":"_num","type":"uint256"}],"name":"viewContract","outputs":[{"internalT
ype":"uint256","name":"","type":"uint256"},{"internalType":"string","name":"","type":"strin
g"},{"internalType":"string","name":"","type":"string"},{"internalType":"string","name":"",
"type":"string"}],"payable":false,"stateMutability":"view","type":"function"}]')


#We set the bytecode of the complete smart contract generated in Solidity (As mentioned in
previous bullet), in order to be deployed.
bytecode =
"608060405260646000553480156100157600080fd5b50610d708061002560000396000f3fe6080604052348015
61001057600080fd5b506004361061006257600003560e01c8063067bd3b4146100675780636a5bb5d8146100855
78063b17dc3ab1461020b578063dfff292014610229578063feb3f929146103af578063ff3a3afe1461047e575b
600080fd5b61006f610667565b60405180828152602001915050604051809103390f35b6100b1600480360360208
1101561009b57600080fd5b810190808035906020019092919050505061066d565b60405180858152602001806
2001806020018060200184810384528781815181526020019150805190602001908083836000 5b8381101561006f
ff5780820151818401526020810190506100e4565b505050509050908101906020016801561012c57808203805160
0183602003610100a0319168152602001915050519060200190808
38360005b8381101561016557808201518184015260208101905061014a565b505050509050908101906020016801
1561019257808203805160018360200361010000a0319168152602001915050519060200190808
38360005b838110156101cb57808201518184015260208101905061014a565b505050509050908101906020016801
1561019257808203805160018360200361010000a0319168152602001915050519060200190808
38360005b838110156102a35780820151818401526020810190506102885650505050905090810190601f16801
56102d05780820380516001836020036101000a0319168152602001915050519060200190808
38360005b838110156103095780820151818401526020810190506102ee565b505050509050908101906020016801
56103365780820380516001836020036101000a0319168152602001915050519060200190808
3825285818151815260200191508051906020019080838360005b8381101561036f57808201518184015260208101
9050610354565b505050509050908101906020016801561039c57808203805160018360200361010000a03191681
5260200191505b509750505050505050506040518091039 0f35b610468600480360360208110156103c55760008
0fd5b810190808035906020019064010000000081111561036e257600080fd5b820183602082011115610383f45760
0080fd5b803590602001918460018302840111640100000000831117156104165760080fd5b91908080601f016
020809104026020016040519081016040528093929190818152602001838380828437600081840152601f19601f
820116905080830192505050505050509192919290505050610ab1565b604051808281526020019150506040518
0910390f35b610665600480360360608110156104945760080fd5b810190808035906020019064010000000081
11156104b157600080fd5b820183602082011115610 4c357600080fd5b803590602001918460018302840111640
1000000083117156104e557600080fd5b91908080601f016020809104026020016040519081016040528093929
1908181526020018383808284376000818401526 01f19601f820116905080830192505050505050509192919
0803590602001906401000000008111156105485760080fd5b820183602082011115610 55a57600080fd5b8035
9060200191846001830284011164010000000083111715610 57c57600080fd5b91908080601f016020809104026
0200160405190810160405280939291908181526020018383808284376000818401526 01f19601f820116905080
8301925050505050509192919290803590602001906401000000008111156105df57600080fd5b82018360208
20111156105f157600080fd5b803590602001918460018302840111640100000000831117156106135760080fd
5b91908080601f016020809104026020016040519081016040528093929190818152602001838380828437600081
840152601f19601f820116905080830192505050505050509192919290505050610bea565b005b600054815 65b
600160205280600052604060002060009150905080600001549080600101805460018160011615610100020316
6002900480601f016020809104026020016040519081016040528092919081815260200182805460018160011615
```

```
61010002031660029004801561071f5780601f106106f45761010080835404028352916020019161071f565b820
191906000526020600020905b81548152906001019060200180831161070257829003601f168201915b50505050
5090806002018054600181600116156101000203166002900480601f01602080910402602001604051908101604
05280929190818152602001828054600181600116156101000203166002900480156107bd5780601f1061079257
6101008083540402835291602001916107bd565b820191906000526020600020905b81548152906001019060200
18083116107a057829003601f168201915b50505050509080600301805460018160011615610100020316600290
0480601f01602080910402602001604051908101604052809291908181526020018280546001816001161561010
002031660029004801561085b5780601f106108305761010080835404028352916020019161085b565b82019190
6000526020600020905b81548152906001019060200180831161083e57829003601f168201915b5050505050905
084565b60008054905090565b600060608060606001600086815260200190815260200160002060000154600160
0087815260200190815260200160002060010160016000888152602001908152602001600020600201600160008
98152602001908152602001600020600301828054600181600116156101000203166002900480601f0160208091
0402602001604051908101604052809291908181526020018280546001816001161561010002031660029004801
56109635780601f1061093857610100808354040283529160200191610963565b82019190600052602060002090
5b81548152906001019060200180831161094657829003601f168201915b5050505050925081805460018160011
6156101000203166002900480601f01602080910402602001604051908101604052809291908181526020018280
54600181600116156101000203166002900480156109ff5780601f106109d457610100808354040283529160200
1916109ff565b820191906000526020600020905b8154815290600101906020018083116109e257829003601f16
8201915b50505050509150808054600181600116156101000203166002900480601f01602080910402602001604
0519081016040528092919081815260200182805460018160011615610100020316600290048015610a9b578060
1f10610a7057610100808354040283529160200191610a9b565b820191906000526020600020905b81548152906
0010190602001808311610a7e57829003601f168201915b5050505050905093509350935093509193509193565b
600080606390505b600160005401811015610be3576001600082815260200190815260200160002060030160405
16020018082805460018160011615610100020316600290048015610b3a5780601f10610b185761010080835404
02835291820191610b3a565b820191906000526020600020905b815481529060010190602001808311610b26575
b505091505060405160208183030381529060405280519060200120836040516020018082805190602001908083
835b60208310610b8c5780518252602082019150602081019050602083039250610b69565b60018360200361010
00a0380198251168184511680821785525050505050509050019150506040516020818303038152906040528051
90602001201415610bd65780915050610be5565b8080600101915050610ab9565b505b919050565b60016000540
1600081905550604051806080016040528060005481526020018481526020018381526020018281525060016000
80548152602001908152602001600020600082015181600001556020820151816001019080519060200190610c5
3929190610c96565b506040820151816002019080519060200190610c70929190610c96565b5060608201518160
03019080519060200190610c8d929190610c96565b5090505050505050565b8280546001816001161561010002031
66002900490600052602060002090601f016020900481019282601f10610cd757805160ff191683800117855561
0d05565b82800160010185558215610d05579182015b82811115610d04578251825591602001919060010190610
ce9565b5b509050610d129190610d16565b5090565b610d3891905b80821115610d345760008160009055506001
01610d1c565b5090565b9056fea265627a7a7231582003ff8fab74afc9d806871688f504c9e3f3b0983b73b0906
89a47c90a2eca14d364736f6c634300050c0032"

#We select the addres that will execute the Smart Contract in order to run the
transactions.
address = web3.toChecksumAddress("0xe3aa467e29b00Dfa101756CAe8D986Cd1b902156")

#We deploy the SmartContract in the Ethereum Blockchain (Ganache in this case)
Deploy = web3.eth.contract(abi=abi, bytecode=bytecode)
tx_hash = Deploy.constructor().transact()
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)


contract = web3.eth.contract(
    address=tx_receipt.contractAddress,
    abi=abi
)

#This function calculates the MD5 Hash of the selected document.
def getmd5file(archivo):
    hashmd5 = hashlib.md5()
    f = open(archivo, "rb")
    for bloque in iter(lambda: f.read(4096), b""):
        hashmd5.update(bloque)
    return hashmd5.hexdigest()

#This function launches the graphic interface for the user select the document he want to
hash.
def uploadfile():
    root = tk.Tk()
```

```
    root.withdraw()
    file_path = filedialog.askopenfilename()
    #returns the filepath
    return(getmd5file(file_path))

#This function upload a new registry into the Blockchain.
def option1():
    hasher = ""
    contractnumber = ""
    os.system ("cls")
    input("\n\nP<------------------------ Press ANY KEY to SELECT the file you want to hash
    in to the BLOCKCHAIN------------------------>")

    #upload function is called,it actually returns the MD5 HASH of the selected document.
    hasher = uploadfile()
    print ("\n\nThe MD5 Hash of the document is %s." % hasher)
    name = input('\n\nPlease insert your full name: ')
    email = input('\n\nPlease insert your email: ')
    input("\n\n<------------------------ Press ANY KEY to save the information in to the
    BLOCKCHAIN------------------------>")
    os.system ("cls")

    #Writes the information into the Ehtereum Blockchain.
    tx_hash = contract.functions.newContract(name, email, hasher).transact()

    #wait untill the transaction es exectued, the block is mined, and the transaction
    confirmed.
    web3.eth.waitForTransactionReceipt(tx_hash)

    #In order to verify and get the number of register from the smart contract
    #we call "SEARCH CONTRACT", this function loops threw the entire registries
    #until the saved hash is found
    contractnumber = str(contract.functions.searchContract(hasher).call())

    print("\n\nThe contract was saved!\n\n "\

"#############################################################################\n\n\n"\
        "         INDEX NUMBER: %s\n\n         NAME: %s\n\n         EMAIL: %s\n\n
HASH: %s\n\n\n"\
        "#############################################################################\n"\
        "         TRANSACTION RECEIPT: %s \n "\

"#############################################################################\n\n\n"\
         % (contractnumber, name, email, hasher, tx_hash))
    input("<------------------------ Press ANY KEY to CONTINUE ------------------------
>\n\n")

#This function looks for a specific saved contract and returns the information to the user.
def option2():
    os.system ("cls")
    referenceNumber = int(input("Please insert the reference number of the contract you
want to verify: "))
    listReturn = []
    listReturn = contract.functions.viewContract(referenceNumber).call()
    print("Request:\n\n "\
    "#############################################################################\n\n\n"\
    "         INDEX NUMBER: %s\n\n         NAME: %s\n\n         EMAIL: %s\n\n         HASH:
%s\n\n\n"\
    "#############################################################################\n"\
    %   (str(listReturn[0]), listReturn[1], listReturn[2], listReturn[3]))
    input("<------------------------ Press ANY KEY to CONTINUE ------------------------
>\n")

#This function return the list of existing documents saved.
def option3():
    os.system ("cls")
```

```
    contractnumber = contract.functions.countContracts().call()
    print( "List of existing contracts saved in Cryptobox: ")
    for x in range (101, contractnumber+1, 1):
        listReturn = []
        listReturn = contract.functions.viewContract(x).call()

print("################################################################################\n"\
              "         INDEX NUMBER: %s ||        NAME: %s ||        EMAIL: %s ||
HASH: %s"\
              % (str(listReturn[0]), listReturn[1], listReturn[2], listReturn[3]))
    print("################################################################################")
    input("\n\n\n<----------------------- Press ANY KEY to CONTINUE ----------------------
-->")


#This is the inital frame.
os.system ("cls")
print(" WELCOME TO Cyrptobox. \n\n\n\n\n")
input("<----------------------- Press ANY KEY to CONTINUE ----------------------->")

#Main menu.

option = 0
#Case option - "while" Call each function.
while option != 4:
    #Try is in order to halt in case there is an error in the introduced number (or during
the execution).
    try:
        os.system ("cls")
        option = int(input("MAIN MENU:\n\n "\

"################################################################################\n\n\n"\
        "         1) SAVE NEW FILE \n\n         2) RETRIEVE SAVED FILE \n\n         3) LIST
OF SAVED FILES\n\n         4) EXIT! \n\n\n"\

"################################################################################\n\n"\
        "Input: "))

        if option == 1: option1()

        elif option == 2: option2()

        elif option == 3: option3()

        elif option == 4:
            os.system ("cls")
            print("\n\n(Cyrptobox) - ByeBye")

        else:
            Input("Please select one of the corresponding options.")

    #Error --> Loops again (ex: was expected an INT and introduced a STRING.)
    except:
      os.system ("cls")
      input(" Error occurred, lets start again")
```