hashchain

Release 0.6.2

Seraphin Vandegar

Jun 05, 2019

USER DOCUMENTATION:

CHAPTER

ONE

INTRODUCTION

`hashchain` is a Python package developed to join the ease of use of Python with the security of blockchain to certify that your important records haven't been tampered with.

The core module creates a hash chain, attesting that no record can be tempered with, once saved. The blockchain module save a proof of your hashchain permanentely in the most secured way. Then, it's impossible to alter the haschain without causing a discrepancy with the blockchain.

No need for third party certification anymore. No more single point of failure nor certification costs.

Note: The package is in beta release Even though No major change to the hashing mechanism should be expected, we can't provide any kind of grantee. Use it in a production environment at your own risk.

CHAPTER

TWO

DOCUMENTATION

2.1 Getting Started

The hashchain library is a collection of module which contain specific functionality to certify data.

- `records` contains all the functions to hash records and build a hashchain in your database
- `ethereum` contains all the functions to deploy a smart contract on the Ethereum blockchain and interact with it.

See the Modules section for more details about these modules.

2.1.1 Installing hashchain

This Python package is available on PyPi package manager. We recommend installing it in a virtual environment.

1. Open a terminal and run the following command : `pip install hashchain`
2. Import the package in you script with `import hashchain`

2.1.2 Ethereum blockchain requirements

Even if this package provides all the tools to interact with the Ethereum blockchain, you will still need a few elements:

1. A wallet

There are many options to create an Ethereum wallet. My favorite is to use Metamask , which is a browser extension and allows you to use your wallet to login to dApps directly via your browser. For testing purposes, you can create a wallet on the Ropsten test network, which works with fake money.

2. Some Ether

Every transaction on the Ethereum blockchain uses a certain amount of Ether, called gas. The price of the gas depends on the volume of transactions on the network and the amount needed for a transaction depends on the resources needed to execute it. You can buy Ether from $ on an exchange like Coinbase . For testing purposes, enter your Ropsten wallet address on the Ropsten Faucet to get free Ether.

3. An access to the Ethereum network

One option would be to run en Ethereum node but it's not easy and needs a certain amount of resources.

Using a connection provider like Infura is the easiest and cheapest way to get instant access to the blockchain. You just need to register on their platform and get your Ethereum endpoint.

To use the `hashchain` package, you will need your wallet private and public key and the provider_url, which is your Ethereum endpoint

Voilà! You're now ready to use the `hashchain` package.

2.2 Examples of scripts using haschchain

Here are a few code samples showing the usage of the different modules from this package.

Note: You can find a program used to demo this package on this repo: https://github.com/svandegar/hashchain-demo

2.2.1 Records

Implement a hash chain in a MongoDB database

This script implements a hash chain, certifying data integrity in a database in a 7 lines of code.

```
from hashchain import records
from datetime import datetime
import pymongo
import random

# MongoDB connection
MONGO_CONNECTION_STRING = os.environ.get('MONGO_CONNECTION_STRING')
client = pymongo.MongoClient(MONGO_CONNECTION_STRING)
db = client['database-name']
collection = db.collection

# Build random data
data = {
    'timestamp':datetime.now().replace(microsecond=0), # round to seconds to avoid␣
↪problems due to MongoDB datetime precision limitation
    'sensorId':'ERDP-QT24', # dummy sensorId
    'value': random.random()
}

# Hash data along with the hash of the previous record
try:
    last_record = collection.find({"sensorId": data['sensorId']}).sort([("timestamp",␣
↪-1)])[0]
    last_record_hash = last_record['hash']

except: # If this is the first record in the DB
    last_record_hash = None

record = records.Record(data, last_record_hash)

# Save the dictionary output of the record in the database
collection.insert_one(record.to_dict())
```

Verify a hash chain from a MongoDB database

This script verifies the integrity of an existing hash chain in a MongoDB database with two lines of code.

```python
from hashchain import records
import pymongo

# MongoDB connection
MONGO_CONNECTION_STRING = os.environ.get('MONGO_CONNECTION_STRING')
client = pymongo.MongoClient(MONGO_CONNECTION_STRING)
db = client['database-name']
collection = db.collection

# Get the data form the DB
chain = mongo_collection.find({'sensorId': 'ERDP-QT24'},
                                {"_id": 0}).sort([("timestamp", -1)])

db_records = list(chain)

# Verify the chain validity
records.verify(db_records)
```

2.3 Solidity Contrat

Here is the Solidity contract definition built by the Python `hashchain` module.

This contract has been designed to store data privately using as little gas as possible.

This contract has been tested against MythX security standards.

```
pragma solidity 0.5.7;

contract Hashchain {

    // variables
    address  public  owner;
    mapping(bytes32 => bytes32)  public hashChain;

    // events
    event newHashRecorded(bytes32 indexed key, bytes32 indexed value);

    // constructor
    constructor() public {
        owner = msg.sender;
    }

    // functions
    function() external {}

    function record(bytes32 key, bytes32 value) external {
        require(msg.sender == owner);
        hashChain[key] = value;

        emit newHashRecorded(key, value);
    }

    function getHash(bytes32 key) external view returns (bytes32){
        return hashChain[key];
    }
```

(continues on next page)

(continued from previous page)

```
}
```

2.4 Records

class hashchain.records.**Record**(*content*, *previous_hash=None*)

get_content()
Get the original content of the record

Return type dict

Returns dict

get_hash()
Get the hex hash of the record

Return type str

Returns hex string

get_previous_hash()
Get the previous hash

Return type str

Returns hex string

hex()
Get the hex hash of the record

Return type str

Returns hex string

to_dict()
Return a dict of the complete record along with the hex string of the record's hash and the previous hash

Return type dict

Returns dict

update(*new_content*)
Updates the record and recalculated the hash

Parameters **new_content** (dict) – new record's content

Returns None

hashchain.records.**verify**(*records_dicts*)
Verifies a given list of records dicts

Parameters **records_dicts** (list) – list of Records objects

Return type bool

Returns returns True if the list is valid. Raise ValueError if not valid.

2.5 Ethereum

class hashchain.ethereum.connector.**EthConnector**(*contract_abi*, *contract_address*, *sender_public_key*, *sender_private_key*, *provider_url*)

Connector to interact with a smart contract on the Ethereum blockchain.

get_record(*key*)

Get the record hash from the smart contract storage

Parameters **key** (str) – unique key

Return type str

Returns hexadecimal string of the hash

record(*key*, *hash*, *wait=False*)

Records the key and hash in the smart contract storage

Parameters

- **key** (str) – indexed key, used to retrieve the hash
- **hash** (str) – hash of the record
- **wait** – wait for the transaction to receipt before completing

Return type str

Returns transaction hash

class hashchain.ethereum.contract.**EthContract**

Solidity contract encapsulated in a Python object

deploy(*sender_public_key*, *sender_private_key*, *provider_url*)

Deploy the Solidity Smart contract to the Ethereum blockchain

Parameters

- **sender_public_key** (str) – public key of the sender
- **sender_private_key** (str) – private key of the sender
- **provider_url** (str) – address of the Ethereum connection provider

Return type str

Returns hexadecimal string of the hash of the transaction hash

get_txn_receipt()

Wait until the transaction receipt is available and add the address variable to EthContract and returns the transaction receipt of the contract creation transaction.

Return type dict

Returns transaction receipt

PYTHON MODULE INDEX

h

INDEX