Membre du groupe:

GHOFRANE HAMBLI

BM

AMINE KHODJA Ahmed Ramy

ALPHA KANTE

Sujet choisi : Récupération des données en cas de litige

Utilisation de Shamir

Objectif :

- Identifier les nœuds cibles : dans notre systéme les identités des utilisateurs sont fragmenté en des fragments SHAMIRs et on suppose que le nombre de SHAMIRs pour chaque utilisateur est 6 et qu'on à besoin de 4 SHAMIRs pour pouvoir identifier l'identité d'un l'utilisateur, ici les nœuds cibles sont bien les 6 fragments SHAMIRs.
- Atteindre un certain nombre des nœuds cibles (ici on a besoin d'atteindre 4 nœuds)
- Récupérer les informations sur ces nœuds et identifier l'identité de l'utilisateur.

Fonctionnement

1- Identification des nœuds cibles :
Les nœuds se trouvent sur un systéme distribué peer to peer donc le fonctionnement sera comme-suit :
-faire une requête broadcast vers tous les nœuds du système portant l'identifiant de l'user qu'on souhaite identifier ;
-Les nœuds porteurs des fragements SHAMIRs relatifs a l'identité demandé vont repondre a cette requête ;

2-Atteindre un certain nombre (ici 4 nœuds) des nœuds ayant repondu a la requete :

-effectuer un choix des 4 nœuds parmi les 6 cibles (un choix optimal en utilisant un algorithme de routage qui permet d'atteindre les nœuds les moins couteux) ;
-Utilisation de la table de routage pour atteindre les 4 nœuds élus ;

3- Récupérer les informations sur ces nœuds et identifier l'identité de l'utilisateur en utilisant les couples récupérés des 4 nœuds en se basant sur le principe de l'algorithme de SHAMIR.

Pratique :

Dans l'intention de résoudre un litige. Le travail va consister donc a retrouvé un bloc a partir de la blockchain.

Il faudra alors construire la blockchain en premier lieux.

On a alors trouvé un programme sur flask qui permet de créer une blockchain a partir des données qui seront enregistrées dans chaque block.

block = {

'index': 1,

'timestamp': 1506057125.900785,

'transactions': [

{

'id': "8527147fe1f5426f9dd545de4b27ee00",

'source': "a77f5cdfa2934df3954a5c7c7da5df1f",
'destination ' : "b88f5cdfa2934df3954a5c7c7da5df1f",

'prix': 2,
'dixtance': 5,

}

],

'preuve': 324984774000,

'hash_precedent':
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"

}

Pour pouvoir créér une blockchain l'on peux utiliser le git suivant

https://github.com/dvf/blockchain

Gestion des litiges

Pour gerer les litiges l'objectif est de pouvoir retrouver le bloc a partir de son hash

Nous allons utiliser le git qui permet la gestion des blocks

https://github.com/blockchain/api-v1-client-python

Installation

```
$ pip install blockchain

$ git clone https://github.com/blockchain/api-v1-client-python
$ cd api-v1-client-python
$ python setup.py install
```

L’utilisation se presente comme suit

get_block

permet d’obtenir un block a partir de son hash

Params:

```
block_id : str - block hash
```

Usage:

```
from blockchain import blockexplorer

block =
blockexplorer.get_block('000000000000000016f9a2c3e0f4c1245ff24856a79c348069
69f5084f410680')
```

get_tx

obtenir une transaction a partir de transaction hash. Returne un `Transaction` object.

Params:

```
tx_id : str - transaction hash
```

Usage:

```
tx =
blockexplorer.get_tx('d4af240386cdacab4ca666d178afc88280b620ae308ae8d2585e9
ab8fc664a94')
```

get_block_height

Obtenir un tableau de blocks vec une taille precise. Returns an array of `Block` objects.

Params:

```
height : int - block height
```

Usage:

```
blocks = blockexplorer.get_block_height(2570)
```

get_address

Get a single address and its transactions. Returns an `Address` object.

Params:

```
address : str - address(base58 or hash160) to look up
filter : FilterType - the filter for transactions selection (optional)
limit : int - limit number of transactions to display (optional)
offset : int - number of transactions to skip when display (optional)```

Usage:
```python
address = blockexplorer.get_address('1HS9RLmKvJ7D1ZYgfPExJZQZA1DMU3DEVd')
```

get_xpub

Get a single xpub and its transactions. Returns an `Xpub` object.

Params:

```
xpu: str xpub to look up
filter : FilterType - the filter for transactions selection (optional)
limit : int - limit number of transactions to display (optional)
offset : int - number of transactions to skip when display (optional)```

Usage:
```python
xpub =
blockexplorer.get_xpub('xpub6CmZamQcHw2TPtbGmJNEvRgfhLwitarvzFn3fBYEEkFTqzt
us7W7CNbf48Kxuj1bRRBmZPzQocB6qar9ay6buVkQk73ftKE1z4tt9cPHWRn')
```

get_multi_address

Get aggregate summary for multiple addresses including overall balance, per address balance and list of relevant transactions. Returns an `MultiAddress` object.

Params:

```
addresses : tuple - addresses(base58 or xpub) to look up
filter : FilterType - the filter for transactions selection (optional)
limit : int - limit number of transactions to display (optional)
offset : int - number of transactions to skip when display (optional)```

Usage:
```python
addresses =
blockexplorer.get_multi_address('1HS9RLmKvJ7D1ZYgfPExJZQZA1DMU3DEVd',
xpub6CmZamQcHw2TPtbGmJNEvRgfhLwitarvzFn3fBYEEkFTqztus7W7CNbf48Kxuj1bRRBmZPz
QocB6qar9ay6buVkQk73ftKE1z4tt9cPHWRn)
```

####`get_balance` Get balances for each address provided. Returns a dictionary of str to `Balance` objects.

Params:

```
addresses : tuple - addresses(base58 or xpub) to look up
filter : FilterType - the filter for transactions selection (optional)
```
```

Usage:
```python
addresses =
blockexplorer.get_multi_address('1HS9RLmKvJ7D1ZYgfPExJZQZA1DMU3DEVd',
xpub6CmZamQcHw2TPtbGmJNEvRgfhLwitarvzFn3fBYEEkFTqztus7W7CNbf48Kxuj1bRRBmZPz
QocB6qar9ay6buVkQk73ftKE1z4tt9cPHWRn)
```

get_unspent_outputs

Get an array of unspent outputs for an address. Returns an array of `UnspentOutput` objects.

Params:

```
addresses : tuple addresses - addresses in the base58 or xpub format
confirmations : int - minimum confirmations to include (optional)
limit : int - limit number of unspent outputs to fetch (optional)
```

Usage:

```
outs =
blockexplorer.get_unspent_outputs('1HS9RLmKvJ7D1ZYgfPExJZQZA1DMU3DEVd')
```

get_latest_block

Get the latest block on the main chain. Returns a `LatestBlock` object.

Usage:

```
latest_block = blockexplorer.get_latest_block()
```

get_unconfirmed_tx

Get a list of currently unconfirmed transactions. Returns an array of `Transaction` objects.

Usage:

```
txs = blockexplorer.get_unconfirmed_tx()
```

get_blocks

Get a list of blocks for a specific day or mining pool. Returns an array of `SimpleBlock` objects.