第6章 Jupyter Notebookでステップ・バイ・ステップ

ラズパイで試して合点 ブロックチェーン基本機能

佐藤 聖

試す準備

第1部 Appendix2のように，Jupyter Notebookの実行プログラムを格納してあるイメージ・ファイル「Interface201808_RPi.img」を用意します．

ラズベリー・パイが立ち上がったら，コマンド・プロンプトから以下の4つのコマンドを実行してライブラリをインストールします．

```
$ sudo apt-get update && upgrade↵
$ pip3 install graphviz↵
$ pip3 install merkletools↵
$ sudo reboot↵
```

ウェブ・ブラウザで「http://192.168.0.108:8888/tree」にアクセスして，パスワード「interface」を入力してログインします．

IF201808フォルダ配下に「2章.ipynb」が見えるのでクリックします．

体験1…ハッシュ値の計算

ハッシュ値を理解するには実際に計算してみるのがよいと思います．ビットコインを利用するためのオープンソース・ソフトウェアにBitcoin Core（GitHub：bitcoin/bitcoin，https://github.com/bitcoin/bitcoin）があります．これを利用すれば実際に取引されている取引データやブロックを使って調べることができます．ビットコインではさまざまな情報からハッシュ値を算出して利用しているので，ここではハッシュ値の計算だけでなく特徴を見てみます．

IPythonやJupyter NotebookでSHA-256アルゴリズムを使ってハッシュ値を計算してみます．**リスト1**のプログラムを実行するとハッシュ値の計算を体験できます．実際に手を動かしてプログラムを実行してみることでハッシュ関数の特徴を体感でき，Bitcoin Coreなどのツールを使うようになっても理解が深まると思います．

なお，Jupyter Notebookの使い方はAppendix2で紹介します．

● ライブラリの読み込み，関数宣言

リスト1のIn [1]は，ハッシュ値計算用ライブラリを読み込んでおり，In [2]でtext2hash関数を宣言しています．独自の関数を用意しておけば繰り返しハッシュ値を計算するのが簡単になります．

● ハッシュ値の出力

データ長にかかわらず64桁のハッシュ値が出力されることを確認してみます．**リスト1**のIn [3] ～ In [5]では，データとしてa，ab，abcをtext2hash関数に渡して，ハッシュ値を計算しています．

出力は「HASH:」で始まる行がハッシュ値，「LETTERS:」で始まる行がハッシュ値の文字列の長さです．実行結果はいずれも64桁のハッシュ値が表示されました．

皆さんのPCで同じプログラムを実行すると同じハッシュ値が計算されますので試してみてください．関数に任意の長さのデータ（文字，数字，記号の組み合わせでもよい）を渡すだけです．ハッシュ関数アルゴリズムとデータが同じなら同一のハッシュ値が得られることが確認できるはずです．

● ハッシュ値の出力2…1文字変えてみる

次にデータ量が多い場合に，1文字だけを変更してハッシュ関数で計算するとハッシュ値が本当に変化するか確認してみましょう．**リスト1**のIn [6] ～ In [7]を用意して実行してみます．In [6]はデータの最終文字が「c」ですがIn [7]では「0」に変えてあります．

実行するとデータがたった1文字違うだけで，算出されるハッシュ値が大きく異なることが分かります．このように一部のデータが変わっただけでもハッシュ値が変化し，データの改ざんを簡単に検出できます．ブロックチェーンにはハッシュ関数が信頼性の向上に大きく貢献しています．

▶やってみよう問題

任意の300文字以上の文字列（改行のない日本語の

リスト1　文字を変えるとハッシュ値が変わることを体験する

ハッシュ・アルゴリズムの実験

In [1]
```
import hashlib
```
In [2]
```
def text2hash(mytext):
    hash_object = hashlib.sha256(mytext.encode())
    print("HASH:    " + hash_object.hexdigest())
    print("LETTERS: " + str(len(hash_object.
hexdigest())))
```
In [3]
```
text2hash("a")
```
Out [3]
```
HASH:    ca978112ca1bbdcafac231b39a23dc4da786eff8147c
4e72b9807785afee48bb
LETTERS: 64
```
In [4]
```
text2hash("ab")
```
Out [4]
```
HASH:    fb8e20fc2e4c3f248c60c39bd652f3c1347298bb977b
8b4d5903b85055620603
LETTERS: 64
```
In [5]
```
text2hash("abc")
```
Out [5]
```
HASH:    ba7816bf8f01cfea414140de5dae2223b00361a39617
7a9cb410ff61f20015ad
LETTERS: 64
```
In [6]
```
text2hash("abcabcabcabcabcabc")
```
Out [6]
```
HASH:    916f4626f2d02e07085873c17f81157908405190 94e9
4114b706573c9749331f
LETTERS: 64
```
1文字変えた
In [7]
```
text2hash("abcabcabcabcabcab0")
```
Out [7]
```
HASH:    ed1a5a15e462296479d13c0ff1efde93f699bb2a2880
38b1f705ce4f7df5327c
LETTERS: 64
```

マークル・ツリー作成

In [8]
```
import merkletools as mk
```
In [9]
```
mt = mk.MerkleTools(hash_type="md5")
```
In [10]
```
leavies = ["A → B 1BTC", "C → D 1.4BTC", "E → F
0.8BTC", "G → H 4.1BTC"]
```
In [11]
```
mt.add_leaf(leavies, True)
```
In [12]
```
mt.make_tree()
```

マークル・ツリーのグラフ作成

In [13]
```
from graphviz import Digraph
```
In [14]
```
g = Digraph(format="png")
```
In [15]
```
g.attr("node", style="filled")
```
In [16]
```
leaves = mt.get_leaf_count()
```
In [17]
```
for i in range(0, leaves):
    leaf = mt.get_proof(i)
    key_0= list(leaf[0].keys())
    key_1= list(leaf[1].keys())
    g.edge(leavies[i], mt.get_leaf(i))
    g.edge(mt.get_leaf(i), leaf[1][key_1[0]])
    g.edge(leaf[1][key_1[0]], mt.get_merkle_root())
```
In [18]
```
g.node(mt.get_merkle_root(), color="pink")
```
In [19]
```
g.render("./merklegraph")
```
Out [19]
```
'./merklegraph.png'
```

In [20]
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
im = Image.open("./merklegraph.png")
im_list = np.asarray(im)
plt.figure(figsize=(100,80))
plt.imshow(im_list)
plt.show()
```
Out [20]

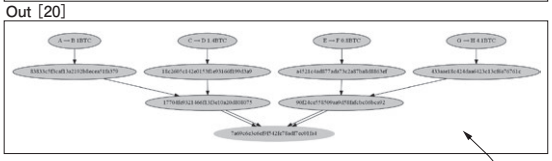

図1に拡大

マークル・ルートのノードのハッシュ値を確認

In [21]
```
print("root:", mt.get_merkle_root())
```
Out [21]
```
root: 7a69c6e3c6ef9f542fe78adf7ec01fa4
```
In [22]
```
import hashlib
```
In [23]
```
print(leavies[0])
```
Out [23]
```
A → B 1BTC
```
In [24]
```
a = hashlib.md5(leavies[0].encode())
```
In [25]
```
a.hexdigest()
```
Out [25]
```
'83833c5f3caf13a2192b8ecea51fa379'
```
In [26]
```
print(mt.get_leaf(0))
```
Out [26]
```
83833c5f3caf13a2192b8ecea51fa379
```
In [27]
```
print(leavies[1])
```
Out [27]
```
C → D 1.4BTC
```
In [28]
```
print(mt.get_leaf(1))
```
Out [28]
```
18c2605c142e0153f1e93166f199d3a9
```

マークル・ツリーに含まれる取引データ（見立てたデータ）を検証

In [29]
```
print(mt.get_proof(0))
```
Out [29]
```
[{'right': '18c2605c142e0153f1e93166f199d3a9'},
{'right': '17704fe9321466f13f3e10a20d808075'}]
```
In [30]
```
print(mt.get_proof(1))
```
Out [30]
```
[{'left': '83833c5f3caf13a2192b8ecea51fa379'},
{'right': '17704fe9321466f13f3e10a20d808075'}]
```
In [31]
```
print(mt.validate_proof(mt.get_proof(1), mt.get_
leaf(1), mt.get_merkle_root()))
```
Out [31]
```
True
```
In [32]
```
print(mt.validate_proof(mt.get_proof(1), mt.get_
leaf(0), mt.get_merkle_root()))
```
Out [32]
```
False
```