Blockchain University end of term exam practice list

#2 page 3~4 ハッシュ関数

- データはどのようにハッシュ関数によって変わるのか
    - ハッシュ値の長さは固定(20バイト、32バイトなど)
    - 不可逆: 逆算不可、唯一（衝突しない前提で）という特性を利用
- 弱いハッシュ関数：MD5、SHA1
- Bitcoinで使用：RIPE160、SHA-256

#2 page 7 取引とブロック

取引ID：取引のsha256sha256ハッシュ - 表示されたときは、常にリトルエンディアン

ブロックIDとハッシュ値が逆になっている：little vs big endian

```
ジェネシスブロックのハッシュ値: (big endian)
6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000
ジェネシスブロックのID: (little endian)
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

#2 Page 8 Base58 : example data

example data 2-> example data 3では末尾の「X」が「Y」に進んだだけといった特徴

- データ：「example data 2」 Base58 : eJdMDAKCH14zqi4DMKX
- データ：「example data 3」 Base58 : eJdMDAKCH14zqi4DMKY

#2 Page 9 Base58Check の仕組み

Base58Checkエンコード：データのsha256sha256から頭4バイトを取って、データの後にくっ付けてbase58エンコード

-> Base58 (data + hash (data) [:4])

#2 Page 15~17 ECDSA

- ECDSAは暗号化ではない
- 秘密鍵から署名を作成 -> 公開鍵から署名の信憑性を検証(一般的な普通の公開鍵暗号の署名方式)
- ECDSAで使用している楕円曲線の方程式を記憶

y^2 = x^3 + ax + b mod p

M_1 + M_2 = M_3

#2 Page 21 secp256k1

- secp256k1

  ビットコインで使われる楕円曲線暗号secp256k1のようなものの中から、secp256k1を見分けられるか

  secp256k1 曲線上の点P:

  P = (55066263022277343669578718895168534326250603453777594175500187360389116729240, 32670510020758816978083085130507043184471273380659243275938904335757337482424)

  ※資料のG(ジェネレータポイント)=の記載に同じ、

  -> Mastering BitcoinではPage 68

#2 Page 21 公開鍵の定義

公開鍵 = 秘密鍵 * 生成点：P = pG

-> G(ジェネレータポイント)に秘密鍵pを掛けると公開鍵が出てくるということ

#2 Page 25-26 署名と検証（RFC 6979）

RFC 6979に沿って、取引を署名する際に用いるランダムな値 k を決定論的に生成することが推奨されている

- RFC 6979の説明 - 取引署名時に推奨される決定的署名決定性署名(Deterministic Signature)

#3 Page 6 エンディアン バイトへの変換は、変換後のバイト文字列を見て２の補数かどうかを判断できるように

（例）2の補数かどうか、難しいものでないものを目でみて判別できるように

- 符号付き絶対値 vs ２の補数
  - 符号付き絶対値 -1

  10000000 00000001

  - ２の補数 -1(1: 000...01のビット反転111...10→ +1)

  11111111 11111111

- ビッグエンディアン vs リトルエンディアン

    - リトルエンディアン　２の補数 -5

ビッグエンディアン２の補数のバイト順序入れ替え

11111111 11111011(ビッグエンディアン)　→　11111011 11111111

#3 Page 12 varint →例を見ておく

varintでは「00」も「fd 00 00」も「fe 00 00 00 00」いずれも10進ゼロ（0）の表現

#3 Page 19 DER 書式 圧縮と非圧縮の違い

公開鍵の書式として、ヘッダバイト[yisOdd]にyが奇数のとき「02」、偶数のとき「03」でxのみで表現できる圧縮[yisOdd][x]を使用

一方、非圧縮は[04][x][y]で、ヘッダバイト「04」とxに加えて、yも含める冗長な表現となるため非効率

#3 Page 21(&20) 電子署名

ブロックチェーンでは: [0x30][l][2][rl][r][2][sl][s][hashtype]

-> 内訳： [DER 署名][hashtype]の内、

- DER 署名：[h][l][rh][rl][r][sh][sl][s]

#3 Page 24,27,42,43

- ブロックヘッダ(Page 42)：[version][prevblockhash][merkleroot][time][bits][nonce]
- ブロック(Page 43)：[magicnum][blocksize][blockheader][txsvi][txs]
- 取引の入力(Page 24)：[txhashbuf][txoutnum][scriptvi][script][seqnum]
- 取引(Page 27)：
    - txid (txハッシュをLEにしたもの):
    - tx ハッシュ:
    - tx:
        - 01000000 (version)
        - 01 (入力の数)
        - 参照先の取引ハッシュ
        - 参照先の出力の索引
        - 入力スクリプトの長さ、106バイト
        - 入力スクリプト
        - nSequence

- 出力の数
- 送金の額, サトシ
- 出力スクリプト
- nLockTime

（例）

・ブロックヘッダに含まれないものは何

・取引に含まれないものは何

・取引の入力に含まれるものは何

#4 OP code

OP_ADD、OP_SUBなど、簡単な例のみ記憶。

-> 例）OP_SUB での順序　先に来たコード（X）-後のコード(Y)がScriptPubkey出力に

- Q.それぞれのscriptPubkeyの解答となりうるscriptSigを解答する場合：
  - OP_ADD OP_7 OP_EQUAL

    `OP_1 OP_6 SIG_1` あるいは `OP_2 OP_5 ..`など、加算の場合は順不同
  - OP_SUB OP_3 OP_EQUAL

    `OP_4 OP_1 SIG_1` あるいは `OP_5 OP_2 ..`など

#4 Page 19 Pay to Pubkey Hash (P2PKH)はどれかを選択させるもの

scriptSig: <署名><公開鍵>

scriptPubkey: **OP_DUP OP_HASH160 <アドレス(PKH)> OP_EQUALVERIFY OP_CHECKSIG**

#4 Page 20 Pay to Pubkey (P2PK) はどれかを選択させるもの

scriptSig: <署名>

scriptPubkey: <公開鍵> OP_CHECKSIG

#4 Page 21 multisigはどれかを選択

scriptSig: OP_0 <sig1> <sig2> … <sigm> ← OP_0はバグ回避のため。必須。

scriptPubkey: OP_m **<pubkey1> <pubkey2> … <pubkeyn>** OP_n **OP_CHECKMULTISIG**

#4 Page 21 Pay to Pubkey Hash (P2SH)はどれかを選択させるもの

scriptSig: OP_0 <sig1> <sig2> … <sigm> <redeemScript>

scriptPubkey: OP_HASH160 **<redeemScriptHash>** OP_EQUAL

redeemScript: OP_m <pubkey1> <pubkey2> … <pubkeyn> OP_n OP_CHECKMULTISIG

-> multisigの出力（scriptPubkey）と同じ構造の**redeemScript**を、前もってハッシュにしてscriptPubkeyに含めて送信するもの

#4 etc. Push 系のOPについて

- リトルエンディアンで数値をプッシュしている
- 例えば、OP_PUSHDATA1 ：次の1バイトをPUSHする、ということ

  *OPコードの詳細は出題されないが最悪の場合は以下wikiを参照：

  https://en.bitcoin.it/wiki/Script

#4 Page 24~25 スタンダード取引 と 取引の検証 の違い

- スタンダード取引はbitcoindで伝播される、非スタンダードな取引でも伝播されないだけで**無効化はされない**
- 取引の検証は、その取引がブロックに含まれた場合には、**不正なブロックになる**

#5 BIPって何の略：BIP = Bitcoin Improvement Proposal

#5 Page 5~ BIP 16: P2SH(Pay To Script Hash), **NOT** EVAL

- P2SH : OP_DUP OP_HASH160 <address> OP_EQUALVERIFY OP_CHECKSIG

-> OP_EVALは別の物

- 一度(BIP 12として）提案されたOP_EVALは現在使用されていない

#5 Page 32 HD 鍵 - BIP 32, BIP 39, BIP 44

例）BIP 32：HD key && BIP 39：MNEMONIC generator

BIP 32の HD 鍵として使用できるシードを生成する基として、人間が覚えられる、保管しやすいパスフレーズを秘密鍵にすること、また、そのシード生成に使用できるwordlistをBIP 39で定義

→ 取引ごとに生成するアドレスの基とする秘密鍵を管理する手間を削減

#5 Page 20 BIP 32で使用する階層表現 （m/0h/1のような）

- 拡張秘密鍵:「xprv」～
- 拡張公開鍵:「xpub」～
- 秘密鍵と公開鍵をパスに沿って派生できる: m/5/6/5/5/2

一つのマスタ秘密鍵mを保管しておいて、それから派生可

#5 Page 25 BIP 39で使用している正規化は「**NFKD**正規化」

#5 Page 29~30 BIP 44: HD ウォレットで統一された階層表現、雛形

BIP 44: m / purpose' / coin_type' / account' / change / address_index

- purpose' = 44, BIPの番号
- coin_type' = 0 (ビットコイン)
- account' = 0, 1, 2, 3 … アカウント
- change = 0 (非お釣り) か 1 (お釣り)
- address_index = 0, 1, 2, 3, 4, … アドレスの索引

→ ビットコインでは「m/44'/0'/」までが固定

（例）「m/44'/0'/0'」はアカウントのマスタ公開鍵

#5 BIP 45の存在意義

BIP 44: 単一署名対応のためmultisigに不向き、だから複数人の公開鍵からどうP2SHのハッシュを生成するか、を標準化するために、BIP 45が定義された。

⇒ しかし、実際にはあまり使われていない

#5 Page 38,39~ BIP 70

- 目的：

  取引先の本人確認をサポート：アドレスのハッシュ値をユーザ間で直接指定だと、第三者が査証したアドレスに送ってしまうリスク、中間者攻撃を抑止

- 流れ：payment request、payment、payment ack 三部構成

1. **payment URI**のなかで ユーザーにペイメントリクエストがあることをr属性の中のURLにて知らせる
2. ウォレットソフトが **payment request** を取得する
3. ウォレットソフトがリクエストについている署名を検証する
4. 支払いに許諾するなら、**payment**をお店に直接送信

→お店は **payment ack**の返事を行う

#5 Locktime系のBIP： BIP 65、BIP 68、BIP 112

- BIP 65: nLockTimeの有用性向上
- BIP 68, BIP112: 相対的なタイムロックの導入

#6 P2Pプロトコル

- 基本：S:getblocks, **R:inv**, S:getdata, {R:blockとかR:tx}

inv:どのブロックの情報持ってますよ、という在庫情報を元に、このハッシュのデータください、とgetdata送信して受信する、という流れ

- getheader -> header：唯一「inv」を介さないやり取り

#6 P2Pプロトコルについて

- SPVでfilterload、merkleblockを受け取ってブロックの存在有無を確認する、といったBasics
    - 例 S:version,R:verack,R:version,S:verack,**S:filterload**,{S:getheaders,R:headers} {S:getdata,**R:merkleblock**,{R:tx}}
- 同期できそうな取引かどうかを判別できるように
    - 例 **S:version, R:verack, R:version, S:verack,** {S:getblocks, R:inv, S:getdata, {R:block}}

#7 課題を参照

https://github.com/nobinuxlab/BlockChainBasic/blob/master/BCUniBronze_07/%237exercise.md