EE 465 Assignment 2

Deep Karman Pal Singh
16D070063

August 21, 2018

Exercise 1. How does the 4-byte nBits field in the Bitcoin block header get converted into a 256-bit target value?

Solution. Bitcoin uses a form of base 256 scientific notation to encode this value.
We have 4 bytes in nBits, which gives us our difficulty as

$$\text{mantissa} \times 256^{\text{exponent - number of bytes in mantissa}}$$

This is the resource I used as reference. □

Exercise 2. How are Bitcoin coinbase transactions guaranteed to have different TXIDs?

Solution. Since the coinbase transactions that produced the bitcoins have different transaction IDs (guaranteed by BIP34), any transaction hashing them to generate a new transaction ID would have different inputs and hence we claim that double SHA256 collision would not occur (rather, they occur with a vanashingly small probability).
The only case that remains is if the same miner manages to mine two blocks with the same reward, the TXID would be the same since (also we still need to guarantee that all initial coinbase transaction have different TXIDs for the above argument to hold). BIP34 ensures that, by adding the height of the block to each coinbase transaction, ensuring the inputs for the double SHA256 hash are different so the outputs (i.e. the TXID) must also be different.
These are the resources I referred to for this question. □

Exercise 3. Convert the following scripts into their hexadecimal bytecode representations. For convenience, represent all data such as PubKeyHash and PubKey1 as all zero bytes. Hint: See script.h in the Bitcoin github repository

- OP_DUP OP_HASH160 PubKeyHash OP_EQUALVERIFY OP_CHECKSIG
- OP_2 PubKey1 PubKey2 PubKey3 OP_3 OP_CHECKMULTISIG

- OP_HASH160 RedeemScriptHash OP_EQUAL

Solution. We have tje following encodings of the above strings:

- 0x76 0xa9 PubKeyHash 0x88 0xac
- 0x52 PubKey1 PubKey2 PubKey3 0x53 0xae
- 0xa9 RedeemScriptHash 0x87

I referred to Bitcoin source code for this question. □

Exercise 4. Describe response scripts which will unlock the following challenge scripts. All data items in the challenge scripts have an implicit data push operator before them which pushes the item onto the stack.

- OP_2DUP OP_SHA256 Hash1 OP_EQUALVERIFY OP_SHA256 Hash2 OP_EQUALVERIFY
- OP_SIZE OP_ROT OP_SIZE OP_NIP OP_EQUAL
- OP_IF OP_DROP PubKeyB OP_CHECKSIG OP_ELSE OP_DROP PubKeyA OP_CHECKSIG OP_ENDIF

Solution. We have the following response scripts:

- String2 String1 (Note that SHA256(String1) = Hash1 and SHA256(String2) = Hash2)
- StringA StringB (both should be of the same "size", i.e. length to be valid and unlock the challenge script)
- SigB OP_0 OP_1 (OR) SigA OP_0 OP_0

This is the reference I used for the above question. □