Blockchain
Blockchain Development
Week 1

1 Linux

If using another operating system create a virtual machine with Ubuntu. If you are already using Linux then you should not need to do anything. Much of what we do will be based on Ubuntu 18.04 LTS (64 bit).

Next week we will introduce splashtop; this week we are going to look at the virtual environment for linux. Install Oracle's Virtual Box (see `https://www.youtube.com/watch?v=EOibXehwYWE` for MS Windows installation). Then download the virtual machine from smerf.net - this will take 15-30mins depending on your connection strength and bandwidth - and click on the CST3550.zip link. Decompress this file and make sure you have 20Gb of hard drive space.

All development will take place in this environment. Get the 3550 VM working as a guest in Virtual Box. The password is 'cst4025'.

Open a terminal and as a simple exercise create a directory Blockchainand a subdirectory 1. Save all your work in this directory.

2 Cryptographic Hash

MD5 [**?**] is possibly the most popular hashing algorithm, this does not mean it is the best. For a review of hashing algorithms see [**?**, **?**]. In this exercise we are going to use Secure Hash Algorithm (SHA, [**?**]) with 256-bit mode output.

Create 8 files as shown in Table 1, you can do this manually or write a program.

Filename	**Content**
0.txt	0
1.txt	1
2.txt	2
3.txt	3
4.txt	4
5.txt	5
6.txt	6
7.txt	7

Table 1: Eight Files with a single digit

Let's complete a hash for the first file, '0.txt', type the following:

```
shasum -a 256 0.txt
```

The '`shasum`' is the command. The '`a`' switch is followed by the value of 256, which is the algorithm selected. Finally, the filename is provided. The output should be as follows:

```
5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9 0.txt
```

You should have 8 files, find the cryptographic hash for SHA using 256 output? The output to all the files is shown below:

```
5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9 0.txt
6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b 1.txt
d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35 2.txt
4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce 3.txt
4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a 4.txt
ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d 5.txt
e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683 6.txt
7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451 7.txt
```

3 Merkle Tree

Merkle Trees are binary hash tree. With files '0.txt' and '1.txt' build a single hash tree as shown in Fig. 1

- shasum -a 256 *.txt > files.sha

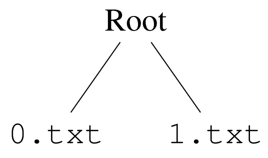

Figure 1: Simple binary hash tree: Concatenate hashes from '0.txt' and '1.txt':
5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.
Hash this to get the root:
cf1ba4395883ffc50f87999888e810feb5deb92498f3c9cd48a4ef7e063a3b37 [2]

Calculate the merkle root for all eight files? If you allow for linebreaks between each hash, the answer for the root should be:
9182e69210a41e2592ae15f11bb237bd45642f173e285e311c4054282b04e0d7

4 Cryptographic Hash Functions

Hashing is an important component to the blockchain, slight variations in the file cause a huge variation in the resulting hash. Often hashes are represented mathematically, as shown in Eq. 1

$$H_a(x) = d \tag{1}$$

where H is the hash function, a is the hashing algorithm used (in our case SHA256), x is the message and d is the digest. How would it be possible for Eq. 2 to be correct.

$$H_a(x) = H_a(y) \tag{2}$$

[2]This is the result of converting the hexidecimal to a string and inserting new lines between each hash.

4.1 Properties

Important properties of cryptographic hash functions are:

1. **Pre-image resistant**
2. **Second pre-image resistant**
3. **Collision resistant**

Write definitions for each of these properties above?

4.2 Nonces

Nonces play an important part in blockchain consensus algorithms, in particular the proof of work (PoW) consensus algorithm used in Bitcoin. The nonce is added to the data and is a way of changing the digest without changing the data. Eq. 3 shows how nonces are used:

$$H_a(concat(x, n)) = d \tag{3}$$

where H is the hash function, a is the hashing algorithm used, d is the digest, x is the original data message, and n is the nonce. The function 'concat(x,n)', concatenates the two strings. By changing the value of the nonce the digest can change, without altering the data.

The next exercise is to test Proof-of-work consensus algorithm. The nonce is 4 characters in length and is added to the original message. The target is a hex value less than the digest, so $H_a(x + n) < H_a(x)$. Anything less than this is accepted. Write a program to complete this?

4.3 Create a blockchain

In previous sections you created blocks and their associated hashes. In this section the aim is to create a blockchain of 3 files: `0.txt, 1.txt` and `2.txt`. To be a chain, each subsequent block requires the hash of the previous block. This is then appended or inserted somewhere in the file. For simplicity, the hash of the previous blockchain is appended to the end of the file.

- `shasum -a 256 0.txt >> 1.txt`
- `shasum -a 256 1.txt >> 2.txt`

There is a chain formed. Is this secure on its own. Can we break it for malign purposes. Change the content of `0.txt` from 0 to 900 and answer the following questions:

1. What do I need to do to make this work and change the *genesis* block.
2. Why would the solution to the above problem not work on a full blockchain, what is missing from our primitive blockchain?

4.4 Proof of Work

Let us try proof of work on a simple example. The underlying theory of proof-of-work, PoW, is that is requires a solution that is difficult or takes some computational time to complete but once the solution is found can easily be verified with minimal computation.

To help us with this we are going to download a python program that is going to run on the linux machine you have just created. But first, some terminology. The proof-of-work algorithm deployed by most blockchain relies on a 'nonce' value. This value is the solution and distributed to all other nodes in the system. What is the nonce? The nonce is an integer value that is inserted or appended to the block and increments by 1. This changes the hash of the block. The idea is to create a hash value that is less than the target value. If the hash results in a larger value then increment the nonce by 1 and try again. This can be represented in the pseudo code below:

1. nonce=0
2. target is x
3. while (hash(block & nonce) $>$ x)
4. do
5. increment nonce
6. done
7. distribute the nonce and exit

To experiment with this a simple PoW program has been written in python for you, follow the instructions below:

1. download code: `git clone https://www.github.com/iangmitchell/simplePoW`
2. enter directory and type: `cd simplePoW`
3. run program and type: `python nonce.py`