Problem Set #2

Sam Prestwood – swp2sf

Problem 1:

What are the assumptions necessary to support Satoshi's claim that it is more profitable for a greedy attacker with a majority of the mining power "to play by the rules"? (other than the assumption that the greedy attacker is a "he")

Satoshi seems to be assuming that the attacker wants to get rich (i.e. get a lot of Bitcoin). It's conceivable that the attacker may want to reduce global trust in the blockchain, in the hopes that it would bring down Bitcoin as a whole.

Problem 2:

At the end of Section 11, Satoshi presents a table for p < 0.001, where it is listed "q=0.45 z=340". What does this mean in plain English, expressed in a short sentence?

If an attacker has a probability of 0.45 of finding the next block, a person needs to wait for 340 blocks before 99.9% sure that their transaction was accepted into the global blockchain.

Problem 3:

For that same table, what would the z values be for p < 0.05 (instead of p < 0.001)?

```
P < 0.05
q=0.100000     z=4
q=0.150000     z=4
q=0.200000     z=6
q=0.250000     z=7
q=0.300000     z=11
q=0.350000     z=18
q=0.400000     z=37
q=0.450000     z=138
```

I used the following C code to calculate these results:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
        poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
int main()
{
    int num_blocks;
    float q;
    double result;
    printf("P < 0.05\n");
    for(q = 0.1; q < 0.5; q += 0.05)
    {
        num_blocks = 0;
        result = AttackerSuccessProbability(q, num_blocks);
        while(result > 0.05)
        {
            result = AttackerSuccessProbability(q, num_blocks);
            num_blocks += 1;
        }
        printf("q=%f\tz=%d\n", q, num_blocks);
    }
    return 0;
}
```

Problem 4:

(a) There is a problem with this scheme unless the bytes of data[i] indicate that it is in fact from ith index. What is this problem?

I assume the database scheme is to save the each data entry *and* its associated signature on the server; you have no local copies of the signatures. In this case, you would be able to verify that data[i] matches its signature, but would have no way of knowing that data[i] is actually from index i.

(b) Suppose we perform a write at position i, both data and signature. Later on, when we read it back, if the signature matches the data, can we be sure that it is indeed the data item we wrote? Explain.

No. If the signature matches, we can be sure that we wrote the entry, but we have know way of knowing if the entry is at the correct index.

Problem 5:

(a) What is the write/read/verify procedure for this system?

Reading:

- Download the entire database
- Take a hash of entire database
- If the hash is matches your local hash, then read data[i]

Writing:

- Download the entire database
- Take a hash of the entire database
- If the hash matches your local hash, update date[i]
- Hash the entire database and save the result locally

Verify:

- Download the entire database
- Take a hash of the entire database
- If the hash matches your local hash, then database hasn't been tampered with

(b) How does the cost of reading and writing to the database scale with n (the number of records)?

The cost scales linearly with respect to n. In other words, you must access the entire database each time you want to read or write.

Problem 6:

(a) What is the write/read/verify procedure for this system?

Verify:

- Download data[i], its value in the tree (a hash) and its sibling in the tree, data[i]'s hash's parent and its sibling, etc. all the way to the root of the tree
- Check that the hash of data[i] equals its value in the tree
- Concatenate the hash of data[i] and it's sibling hash; check that these equal their parent
    - Continue this process up to the root of the tree
- If all the hashes match, then the tree is valid

Read:

- Verify tree
- Read data[i]

Write:
- Verify tree
- Update data[i]
- Re-verify tree
  - This time updating the hash of data[i] with the hash of its new data
  - Update the parent hashes along the tree up to the node
- Save the new tree

(b) How does the cost of reading and writing to the database scale with n (the number of records)?

Reading and writing scales logarithmically with respect to n

<u>Problem 7:</u>

(a) If a mining pool has 15% ($\alpha = 0.15$) of the total network hashing power, how many blocks is it expected to find in a day?

(24 hours/day) * (60 minutes/hour) / (10 minutes/block) = (144 blocks/day)
0.15 * 144 = 21.6
On average, the pool should find 21-22 blocks each day.

(b) Obtain a general formula for expected number of blocks a mining pool with α fraction of the total hashing power should find in t minutes.

Blocks/minute = (blocks/day)/(minutes/day)
= (α * (24 hr/day) * (60 min/hr) / (10 min/block)) / ((24 hr/day) * (60 min/hr))
= α / 10 (blocks/minute)
==> α * *t* / 10

<u>Problem 8:</u>

Assuming all of the miners are honest, what is the expected number of orphaned blocks per day for an honest mining pool with hashing power α and latency L (as simplified above).

Consider two miners, A and B
Total # of orphan blocks = (# of orphans B found when A found good blocks) +
(# of orphans A found when B found good blocks)
In 1 day, A finds 144 * α blocks and B finds 144 * (1- α) blocks
# of orphans B found when A found good blocks = (α * 144) * (1- α) * (L / 60) / 10
# of orphans A found when B found good blocks = ((1- α) * 144) * α * (L / 60) / 10
Total # of orphans = (α * 144) * (1- α) * (L / 60) / 10 + ((1- α) * 144) * α * (L / 60) / 10
= (12/25) * L * α * (1 – α)

<u>Problem 9:</u>

How does this change if the mining pool is mining selfishly?

We can use the same equation from problem 8 and say that A is selfishly mining and has a majority of the hashing power. Whenever B finds a block, A will reveal a block from its lead, meaning that every block B finds will be an orphan block.

Total # of orphans = (# of orphans B found when A found good blocks) + (# of blocks B found that A trumped with a block from its lead).
$= (\alpha * 144) * (1- \alpha) * (L / 60) / 10 + ((1- \alpha) * 144)$
$= (1 - \alpha) * 144 * (\alpha L/600 + 1)$

<u>Problem 10B:</u>

1. Do “anonymous” online market such as Silk Road and Agora improve the anonymity of its buyers by using escrow between buyers and sellers?
2. Is it illegal to buy a non-illegal item (e.g. someone is self-publishing a book and selling it) from an “anonymous” market such as Silk Road or Agora?