Jacob Freck
Prof. David Evans
CS 4501
10/8/15

Problem Set 2

Problem 1. In Section 6, Satoshi writes: *"The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth."*

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")

assumptions:

1. there are only 2 ways to mine: the default strategy and a fraudulent one in which the dishonest node tries to double spend
   - we already learned about greedy mining - in which the attacker doesn't necessarily double spend, but rather just wastes other's hashing power
2. "acting dishonestly will undermine the validity of his own wealth"
   - an attack on the bitcoin network is an attack on the value of bitcoin
   2a. the attacker will be noticed by others (if others doesn't know there is an attack, then the value would not be affected)
3. an attack requires a majority of cpu power (it does not)
4. potential attacks are attacking only for reasons of financial interest
   - the incentives are such that there is a financial benefit to being honest, but do nothing to deter attackers who have other motives
   - ex. a state gov. thinks that bitcoin is devaluing their currency, so they attack the network. they think that this will destroy bitcoin value and people will return to using state currency. Perhaps they profit when people use state currency, so devaluing bitcoin is of little consequence (they don't care if the coins they double spend are worthless, their goal is entirely beyond bitcoin).

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?

z is the number of blocks linked after the transaction to ensure that transactions legitimacy
q is the probability the attacker finds the next block
p is the probability an honest node finds the next block

If an attacker has 45% of the hashing power of the network, one should wait for 340 blocks after their transaction to ensure a probability of less than 0.1% that the attacker can reverse that transaction.

If an attacker has 45% of the hashing power of the network, one must wait for 340 block confirmations to ensure a less than 0.1% probability that the attacker can reverse the transaction.

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=0.1       z=3
q=0.15      z=3
q=0.2       z=5
q=0.25      z=6
q=0.3       z=10
q=0.35      z=17
q=0.4       z=36
q=0.45      z=137
```

C code:

```c
#include <math.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 argc, const char* argv[] ) {

  printf("P < 0.05\n");

  int i;
  double arr[] = {.10, .15, .20, .25, .30, .35, .40, .45};

  for(i = 0; i < 8; i++) {
    double sum = 1;
    int z = 0;
    while (sum >= 0.05) {
      sum = AttackerSuccessProbability( arr[i], z);
      if (sum < 0.05) {
        printf("q=%G \t z=%d\n", arr[i], z);
      }
      z+=1;
    }
  }
}
```

Problem 4. One naive approach would be to just sign each record data with a private key, and verify the signature on reading it.

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

Assumptions:

- Nothing about the data in data[i] reveals what it's position is

The problem is such that if I only have the private key stored locally, with no information about the data records themselves, then I can verify that whatever is sent to me from the network is valid. However, I cannot verify that if I request the *i*th item, that I am actually receiving the *i*th item. I only know that what I receive is a record that I once signed. It could, however, be any record from 0 to n-1, it need not be the *i*th record.

(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.

Assumptions:

- Nothing about the data in data[i] reveals what it's position is

No, we do not know that the data item is the item we wrote at position *i*. All we know is that the requested data, if it is verified, is a signature-data pair.

In other words, we know that the signature goes with the data (because we can verify it), but we do not know that this is the *i*th item that we wrote. The network could just send back some other data-signature pair, which would still be verified, but not be the *i*th item.

However, even if we disregard the assumption the assumption that data[i] has no data about it's position, we still cannot be sure it is indeed the data item we wrote. Say, we overwrite data[i], and then read it back and verify. The cloud computing service could send back either the old record or the new, and both would be verified (as they were both signed by the private key). We wouldn't know whether the original data was actually overwritten or not, we would only know that at one time we signed this data with the private key, and that it should be in position i.

Problem 5. Another approach would be hashing the concatenation of all records in the database. This hash is a small item that can be stored locally.

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

Write:

- request from the server all of the records
- concatenate all of the records including the to-be-written record in order
- hash that concatenation

Read:

- get data[i]

Verify:

- know h = hash(all_records)
- read from cloud database
  - get all other records from 0 to n-1
  - concatenate in order
- hash that concatenation, hash(all_cloud_records)
- compare to local hash(all_records)
  - if (hash_all records) == hash(all_cloud_records) then data[i] is valid

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

- to read is constant - just request the record you want
- to verify that what you read is valid, it is linear
  - have to request all of the records from the cloud, hash, and then compare to local hash(all_records)
- to write is linear
  - have to request all n records
  - have perform operation on records (concatenate)
  - have to hash concatenation
  - have to send to network

Problem 6. Instead of using the concatenating all the records linear, they were organized into a Merkle

tree.

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

- what do I store?
    - merkle tree root
- what does the cloud store?
    - all of the records
    - intermediate merkle steps

Write (recreate verification method and send to network):
- hash to-be-written record
- request log(n) hashes needed to complete the merkle tree from
- perform log(n) hashes to recompute value of merkle tree root
- send to-be-written data to network

Read (request from network):
- request data[i] from network

Verify:
- request data[i]
- hash data[i]
- request log(n) other intermediate merkle values needed to complete tree
- compute merkle root with hash of data[i] and the other requested values
- compare computed merkle root to known merkle root

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

all operations are logarithmic or better, so it will be on the order of log(n).

Problem 7.
(a) If a mining pool has 15% (α = 0.15) of the total network hashing power, how many blocks is it expected to find in a day?

expected blocks in a day - (24hr * 60 minutes) / 10min per block = 144 blocks per day
0.15 chance * 144 blocks = 21.6 blocks per day

21.6 blocks per 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.

.1 = number of blocks found in a minute (expected: 1 block per 10 minutes)
t = number of minutes
α = fraction of hashing power (percent chance for each block)

Equation:

$$E = \alpha * .1t$$

Problem 8. Assuming all of the miners are honest, what is the expected number of orphaned blocks per day for an honest mining block with hashing power α and latency *L* (as simplified above).

144α - miner's proportion of blocks per day
L/60 - number of minutes delay before network sees this block
- & time at risk for another block to be found
E = (1 - α) * .1t - expected number of blocks found by network in t minutes

English: 144α times per day the network has *L*/60 minutes to find another block

E = (1-α) * [.1 (144α * (*L*/60))]

Problem 9. How does this change if the mining pool is mining selfishly? (For this question, assume that the selfish mining pool learns of a block announced by the rest of the network as soon as it is announced, so will immediately announce any withheld blocks at that time. That is, you may still assume the simplified latency *L* model, but that the selfish mining pool has a spy in the other supernode with a low-latency direct connection to the mining pool.)
note: a = α
Assumption:

- it is not possible for the selfish pool to have an orphaned block
- they will publish and know of blocks published by the network instantly

- selfish miner will get lead of 2 blocks 144aa times per day
- once they have the lead, they expect to keep it for $\frac{a}{1-2\alpha}$ forward steps
- Each step takes 10/α minutes.
- time selfish node will keep the lead:

$$T = \left(\frac{a}{1-2a}\right)\times\left(\frac{10}{a}\right) = \left(\frac{10}{(1-2a)}\right)$$

- meanwhile network finds blocks at E = (1-α) * .1t
- number of blocks network will find in time selfish node keeps lead:

$$OBlocks = (1-a) \times \left(.1 \times \left(\frac{10}{1-2a}\right)\right)$$

English: 144 alpha squared times per day, the selfish mining node will take a lead of 2 in which they expect to take $\frac{a}{1-2\alpha}$ forward steps. Each step takes at 10/a minutes. During the time taken for each step, the network will mine E blocks that will become orphaned at an expected rate of E = (1-α) * .1t. Putting this together, we get that the network will mine OBlocks number of blocks each time the selfish miner takes a lead.

In sum, we expect E number of blocks to be orphaned, where

$$\mathrm{E} = 144a^2 \times \left((1-a) \times \left(.1 \times \left(\frac{10}{1-2a}\right)\right)\right)$$

Problem 10A. In class, we saw how pool-hopping can be used to game a “proportional” reward scheme. Design a simple reward scheme that eliminates pool-hopping incentive. In particular, derive the expected reward for your scheme and show that it does not depend on time since last block (or any variable controlled by the miner).

Reward scheme: have multiple tiers of difficulty easier than bitcoin's difficulty, which pay certain amounts based on their difficulty.
For example d1 may be 1/2 the difficulty of the bitcoin network's difficulty and worth 1/100th the coinbase transaction. Then, d2 may be 1/4 the difficulty of the bitcoin network's difficulty and worth 1/1000th of the

coinbase transaction, and so on.

Thus the pool operator pays for difficulty, similar to how the network pays for it: find a share of a certain difficulty, get paid immediately.

This reward system has nothing to do with any values related to the bitcoin network other than difficulty. There is no time component, and no relation to the block last mined by the pool.

The expected reward (E) for this system is the summation of the probability any of the pool's difficulty tiers are reached times the reward at the level reached.

$$\mathrm{E} = \sum_{k=1}^{n} \mathrm{re}ward_{dk} \times probability_{dk}$$

In this system, the miner has no control over any of the variables: the reward is set by the pool operator and the probability of reaching a difficulty level is a function of their hashing power and nothing more. Unlike the proportional system, the work done by a miner is affected in no way by the work done by other miners.

There is only an agreement between the pool operator and the miner. In fact, there is not even a meaningful connection between the miner and the network at large in terms of the miner receiving coin. If the miner (and the pool) never won a block on the network, the miner would still get paid for hitting the pool's difficulty marks. Effectively, then, this solution is risky for the pool operator, but it does alleviate worries of pool hopping. There is no difference between mining in this pool at any time since the last block found.