Problem Set 2

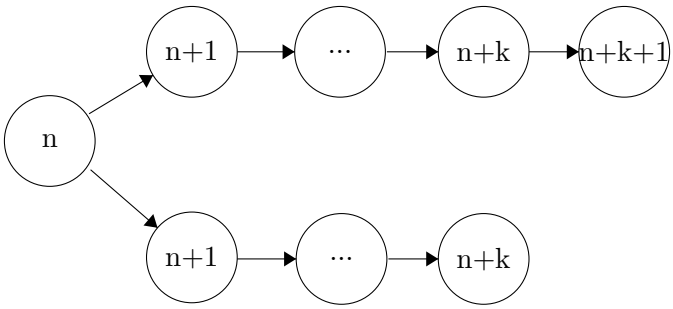
Luke D. Gessler (ldg3fa)

CS 4501-001

October 7, 2015

Problem 1

Reiterating, the alternative to playing by the rules that Satoshi discusses here would be to spend money and then try to reclaim it. It could be reclaimed by choosing and extending a node in the blockchian that existed before we spent the money. This would be block n in the diagram below. Because we control over 50% of the bitcoin network's hashing power, as time goes on it because almost guaranteed that we will be able to generate a longer blockchain, the one on top, that excludes the transactions in which we spent money.



The assumption behind why Satoshi thinks an evil person would not want to do this is that it would undermine trust in bitcoin as a currency. If people lost trust in the currency, its value would fall precipitously, and people would abandon it. This is an outcome the evil person would not want, as if she has invested enough in hardware to gain a majority of the network's hashing power then she would want the currency to remain valuable to maximize the return on her investment in hardware and time.

The assumption seems reasonable. It is hard to imagine that people would continue to use a currency even if it were possible for payments you receive to be erased from history. Note that even payments *not* made by the party controlling the majority of the hashing power could be erased. If the person with the majority wanted to, she could offer a service to others: erasure of a transaction for a fee.

Problem 2

"If an attacker controls 45% of the network's hashing power and she wants to construct a parallel chain from 340 blocks behind the latest node on the blockchain, she has a 0.1% chance of success."

Problem 3

Using a Python port of Satoshi's C code and some manual twiddling of the parameters, the smallest z values that produce $P < 0.05$ were found. See below.

q	z	P
0.10	3	0.01317
0.15	3	0.04422
0.20	5	0.02742
0.25	6	0.04994
0.30	10	0.04166
0.35	17	0.04504
0.40	36	0.04876
0.45	137	0.04934

Problem 4

(a) Unless you are also storing an array of the signature of all n entries locally, you have no way of knowing that the item the service returned to you is in fact `data[i]` and not `data[i']`. All you know is that this element, `data[?]` is one that you wrote earlier to some unknown index.

(b) No. As discussed above, we have no way of knowing whether our cloud provider gave us back `data[i']` instead of the one we requested. So a signature match proves only that we have *some* item that we wrote earlier, not that it is exactly the one we requested.

Problem 5

Assume the cloud provider has an API for us, something like `get(index: int), write(index: int, value: EntryType)`.

(a) **Write.** Call `write(i, v)`. Retrieve all other n entries and put them into memory, concatenate them in order, hash them, and write the hash to local storage.[1]

Read. Call `read(i)`.

Verify. Called after read. Retrieve *all* n entries, concatenate them in memory, and hash them. Compare hash to local hash. Verified if the hash is identical, otherwise something has changed.

(b) Writing is $O(n)$. There is no way to avoid this because all we store locally is the hash of the serialization of the *entire* database. Reading should be $O(1)$, but if we're reading then we really must also verify, and verification is $O(n)$ as we must retrieve all items in the database so we can hash their concatenation. Thus reading and writing scale linearly with the number of records n.

Note that if we wrote the index alongside the data (say at the very beginning) then we could bring the verification step down to $O(1)$, as the index would be signed with our key and we would not need to recalculate the hash.

[1] The same could also be achieved if the service has a function like `hash()` that implements the "concatenate everything and hash it" functionality we want. This avoids the requirement of having a potentially large amount of local memory.

Problem 6

We assume that instead of keeping just a data array on the service we can maintain a full Merkle tree on top of our data array. We also assume we can keep a copy of the hash of the root of the Merkle tree locally.

(a) **Write.** Write to the given index and propagate the changes up the Merkle tree. Update the locally-stored Merkle root. Updating the Merkle tree's values is $O(\log n)$.

Read. Retrieve whatever is at the index. This is $O(1)$.

Verify. Hash what we just retrieved. Recall that we have been maintaining an entire Merkle tree on the service. Request the $O(\log n)$ hashes we need to calculate the root hash. Compare the computed hash with the locally-stored hash. If they match, we can be reasonably sure that our data is intact.

(b) As seen above, reading and writing are both $O(\log n)$ operations.

Problem 7

(a) $E(\#\text{blocks}, \alpha = 0.15, t = 1 \text{ day}) = \dfrac{1 \text{ block}}{10 \text{ min}} \times 1440 \text{ min} \times 0.15 = 21.6 \text{ blocks}$

(b) $E(\#\text{blocks}, \alpha, t) = \dfrac{1 \text{ block}}{10 \text{ min}} \times t \times \alpha$

Problem 8

There are two events we care about here that cover all possible outcomes. For some two nodes A and B, only two things will happen:

(a) A mines a block and broadcasts it. B does not find a block in the following L seconds, so it picks up the new block and starts mining on it.

(b) A mines a block and broadcasts it. B finds a block in the following L seconds, so one of the two blocks is orphaned.

We need to find out how many times (b) is expected to happen.

Call the first event in (b) X_{1A}, where A mines a block. Call the second event in (b) X_{2B}, where B mines a block within L seconds.

To find the number of orphaned blocks network-wide, we want to find $E(X_{1A} \wedge X_{2B}) + E(X_{1B} \wedge X_{2A})$ These two events are independent, so we can write them as $E(X_{1A})E(X_{2B}) + E(X_{1B})E(X_{2A})$

From now on, let A be the pool with hashing power α and B be the other node with hashing power $1 - \alpha$. For dimensional convenience, assume L is given in minutes.

Using the formula from 7(b), $E(X_{1A}) = E(\#\text{blocks}, \alpha, 1440\text{min}) = 144\alpha$ and $X_{1B} = E(\#\text{blocks}, 1 - \alpha, 1440\text{min}) = 144(1 - \alpha)$

Also from 7(b), $E(X_{2B}) = E(\#\text{blocks}, 1 - \alpha, L) = 0.1L(1 - \alpha)$, and $E(X_{2A}) = E(\#\text{blocks}, \alpha, L) = 0.1L\alpha$

So finally, we have the expected number of orphans to be:

$(144\alpha)(0.1L(1 - \alpha)) + (144(1 - \alpha))(0.1L\alpha) = 2 \times (144\alpha)(0.1L(1 - \alpha))$

Problem 9

Because A does not announce blocks until it knows it will have a one-block lead, and because it has an instantaneous connection with B, it will never have its blocks orphaned. So we need only consider the case when A orphans B's blocks.

Let $P(R^i)$ mean "the probability selfish miner A has i blocks in reserve. This is analogous to each of the states in Eyal and Sirer (2013).

To find this, we need to find $P(R^{>1})\wedge$ B finds a block). Note that this is the same as $P(R^{>1})P$(B finds a block), which is also the same as $(1 - P(R^1) - P(R^0))P$(B finds a block)

The probability of B finding a block is $1 - \alpha$. Eyal and Sirer have already calculated $P(R^1)$ and $P(R^0)$. We put these together to find the probability of an orphaned block[2] and multiply by 144 to get the expected number of orphaned blocks in a day:

$$144 \times (1 - \frac{\alpha - 2\alpha^2}{\alpha(2\alpha^3 - 4\alpha^2 + 1)} - \frac{(1-\alpha)(\alpha - 2\alpha^2)}{1 - 4\alpha^2 + 2\alpha^3} - \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}) \times (1 - \alpha)$$

This does not account for the latency L between A and B. If B found enough blocks in that time to destroy A's lead, then there would be no orphaned blocks, though I'm unsure how to account for this in the equation.

Problem 10B

1. Bitcoin wallets, if used correctly, are somewhat hard to associate with personally-identifying information. This could enable any number of illegal activities like money laundering, tax evasion, and sale and purchase of contraband. How are the IRS, FBI, and other government agencies trying to address these problems?

[2] I sum together state 0 and state $0'$