# CompSci 461/661 HW2

## Doublespend Probability

### Goals

The goal of this assignment is to validate Satoshi's equation for the probability of success of the doublespend attack and visualize the results.

This is not a group assignment. All work should be on your own. Do not show your code to others.

### The doublespend attack

As a reminder, the doublespend attack works as follows. Assume that the latest block on the blockchain is block $B_0$.

- The attacker sends a transaction $\mathcal{H}$ to the Bitcoin network that moves coin from an address that she controls to an address that the merchant controls.
- The merchant waits for a sequence of $z$ blocks $(B_1, \ldots, B_z)$, where $z \geq 1$ and $\mathcal{H}$ appears in block $B_1$.
- The merchant hands over the goods to the attacker.
- The attacker releases a chain of blocks, $B'_1, \ldots, B'_{z+1}$. Block $B'_1$ has block $B_0$ as its previous. Contained in $B'_1$ is transaction $\mathcal{F}$, which moves all the coin from the attacker's address to a second address in her control.
- If honest miners haven't reached block $B_{z+1}$ yet, the attacker wins. At that point, the miners will accept block $B'_1$ and the blocks the attacker mined afterwards; which means that the attacker has both the goods and her coin.
- If that attack hasn't won, she can continue attacking, using her mining power to race ahead 1 block more than the honest miners.

Satoshi's paper tells us that:

Given an attacker that controls a fraction $q$ of the mining power, and a merchant that waits for $\mathcal{H}$ to be $z$ blocks deep before releasing goods, the probability $\phi$ that the attacker can mine enough blocks to overtake the blockchain is:

$\phi = 1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (\frac{q}{p})^{z-k}\right)$

where $p = 1 - q$ is the remaining mining power; and $\lambda = \frac{zq}{p}$.

As discussed in class, this is actually the probability to *tie* the main chain, not exceed it. We shall use this equation – Satoshi's equation – so you can compare your results with their paper.

**Doublespend Mining Calculation**

This assignment is for you to determine this probability in 3 different ways, in Python. You will write 3 functions that each take 2 parameters, z – the embargo interval, and q – the attacker's fraction of the network hash power.

**First, create python function that returns the value as calculated above. It should look like:**

```
def satoshi(q,z):
  # Your code here
  return probabilityOfAttackerSuccess
```

Note that this ought to be around 5-10 LOC, depending on how much math you cram on one line. I'm including these LOC estimates to give you a gut check on whether you are misunderstanding the assignment and doing something completely different.

**Second, use a Monte Carlo simulation.**

Your simulation will need to run many thousands of *trials*. In each trial run, you play one instance of the doublespend game using parameters $q$ and $z$. You will simulate two adversaries: one attacker $A$ with mining power $q$; and an honest miner with $p = 1 - q$ power, supporting the honest merchant. Each trial is a new run with everything reset. The final doublespend probability can then be calculated by the number of times the attacker succeeded divided by the total trials.

You need to think about when the simulation is over. There are the win cases to code up of course, but also in theory the attacker can mine their chain forever. For the purposes of this assignment, if the honest chain is at or beyond z, and the honest chain is 35 or more blocks ahead of the attacker's chain, consider that a loss for the attacker.

Do not actually run a mining algorithm! Since the attacker's probability to find a block is its proportion of the network hash rate, you can just generate a random number between 0 and 1 and if this number less than or equal to q, model the attacker finding a block. Otherwise model the honest chain finding a block.

```
def monte(q,z, numTrials=50000):
  # Your code here
  return numAttackerSuccess/numTrials
```

Note the simulator can be done in about 15 LOC (helper function), and the monte function could be an approximate 5 LOC loop that calls your helper function.

*Hint: Do not be alarmed if these results differ a little from Satoshi's equation! What?? Yes, Satoshi's equation is not quite correct!*

**Third, let's investigate this discrepency by creating a Markov chain calculator.**

Your simulation above should have a single decision point per loop: whether the attacking or honest chain found a block. It uses a random number to determine which branch to take. Instead of doing that, modify the code to recursively traverse *both* possibilities. If the attacker's lead is A, the final probability is therefore F(q) = q*F(A+1) + (1-q)*F(A-1). That is, its the probability of the attacker finding a block times the answer to the problem starting at the state where the attacker's lead has increased by one, plus the same for when the attacker's lead was reduced by 1.

Just like the Monte Carlo technique, you need to think about when to stop descending into this recursion. Just like in the prior case, the attacker could mine forever. But rather than cutting off the calculation arbitrarily, in this case cut it off at a point where any further result will be immaterial to an answer given with 5 significant digits. How can that be done? Well, for example, looking at the recursive equation you can see that moving the attacker's lead forward by N blocks would result in a probability of (qˆN)*(A+N). qˆN is getting small quickly. You can therefore calculate and pass the probability of the code taking a particular path into the recursive call and if that probability is less than a very small number, say 1.0/1e60, return a loss for the attacker.

*Hint: keep a cache of calculated states, e.g. attacker has found 5 blocks, honest 3, and return the cached value if you encounter this state again. This will MASSIVELY speed up your routine*

Note, although this seems complicated, its actually about 10 LOC!

You should discover that the Markov chain answers are much closer to your Monte Carlo simulation than to Satoshi's equation!

**Submission Details**

Modify a python file we have created called dspend.py and submit it. This file contains a Test() function for your use. Do not change the existing function declarations in an incompatible manner, but you may add defaulted arguments to the end of them, and of course define your own helper functions, variables, etc. I expect to be able to import your code and call the specified functions with different values of q and z, just like in the Test() code.