Blockchain Cryptography

FNCE 385/885 Assignment 2

Chris Hua

11/18/2016

Part 1: Create Keys

As per class, we define some $n = p * q$, or 161. Then, given $e = 13$, the public key is a composition of 161 and 13.

We can find the private key by evaluating the modulo condition for each d between 1 and a given max number, then returning the first one that is equal to 1. This operation is vectorized and as such much faster in R than a loop.

```
find_priv_key <- function(p, q, e, max_iter = 100) {
    d <- 1:max_iter
    val <- (e * d) %% ((p - 1) * (q - 1))
    min(which(val == 1))
}
find_priv_key(7, 23, 13)
```

```
## [1] 61
```

The private key is then 61.

Part 2: Sign a message

To do the modulo math necessary to sign a message, we need to do modular exponentation (generally). We borrow the implementation in package `numbers` to do this.

```
sign_message <- function(pk, n, m) {
    numbers::modpower(m, pk, n)
}
sign_message(61.0, 161.0, 12.0)
```

```
## [1] 124
```

Part 3: Check if a message is authentic

To check if a message is authentic we can create another function to check the encryption.

Message 1:

```
## [1] FALSE FALSE
```

Message 2:

```
## [1] FALSE FALSE
```

Message 3:

```
## [1] FALSE FALSE
```

None of these are authentic!

Part 4: Mining Bitcoin

We find a nonce that, when appended to the message and hashed via md5, starts with 3 zeroes. Our process is as follows:

```
prove_work <- function(m, k, start = 1000, max_iter = 5000) {
    for(i in start:(start+max_iter)) {
        item <- paste0(m, i)
        hash <- digest::digest(item, algo = "md5")
        if(substr(hash, start = 1, stop = 3) == "000") {
            return(i)
        }
    }
    -1 # answer is outside search bounds
}
```

Then our result is given by:

```
## [1] 1267
```

Part 5: Bitcoin game theory

a) There is some optimum search start point, we can call it i'. Since we have approximately equivalent compute power, we will assume that we can check each number in equal amounts of time.

Then, if my competition starts at 0, and I start at 1, in expectation I will dominate them, since I will check each number greater than 1, *before* they can.

We don't show it here but the greatest concentration of "hash solutions" is at a relatively low number. We performed a search for 1000 hash solutions with $k = 3$ digits, and with hash values between 0 and 20000, and examined the distribution. The average value was about 3600, and 80% were under 5000. The insight is that there is a long tail of values to check for any given message, and that there might be increased efficiency if we do a greedy search - cap the number of iterations we try for each block, and aim to solve as many 'easy' blocks as possible.

We should combine the 'greedy' approach with a larger start value in our search against otherwise clueless competitors.

b) In the face of competition using the same strategy, we would move to larger start numbers. As we noted above, there is a tradeoff between the start point i and the number of blocks solvable within k iterations from the start point.