# Homework 2

Jared Roesch

Spring 2015

## Description

My implementation is below. It seems to correctly predict every byte, but there is some question about what "signifigantly different" means. I used the critieria that if the porportion of $T_S/T_R > 1$, and if not a 0.

## Implementation

```
import math
import random

def mon_pro(a_bar, b_bar, n_prime, r):
    t = a_bar * b_bar
    m = (t * n_prime) % r
    u = (t + m * n) / r
    if u >= n:
        return (True, u - n)
    else:
        return (False, u)

def assert_odd(n):
    if n % 2 != 1:
        raise Exception("n must be odd")

def mod_exp(M, e, n, n_prime, r):
    assert_odd(n)
    M_bar = M * r % n
    x_bar = 1 * r % n
    subs = []
    for i in bin(e)[2:]:
        (sub1, x_bar) = mon_pro(x_bar, x_bar, n_prime, r)
```

```
        if int(i) == 1:
            (sub2, x_bar) = mon_pro(M_bar, x_bar, n_prime, r)
            subs.append(sub1 or sub2)
        else:
            subs.append(sub1)

    return (subs, mon_pro(x_bar, 1, n_prime, r))

def messages():
    for i in range(0, 100):
        yield random.randrange(1, 10000000003, 2)

# Setup parameters for RSA
p = 929
q = 389
n = p * q
r = 240113
r_inv = 16576
n_prime = 269338
totient_n = 360064 # EulerPhi[n]
e = 138577 # CoprimeQ[e, tn]
d = 94513 # PowerMod[e, -1, tn]

# Store the timing information leaked by M_i^e mod d for each message M_i.
results = []

for M in messages():
    (subs, result) = mod_exp(M, d, n, n_prime, r)
    # Store the timing information for each bit.
    for i in range(0, len(subs)):
        try:
            results[i].append(subs[i])
        except IndexError:
            results.insert(i, [subs[i]])

# Loop through the results computing the porportion of True/False,
# and print out the relevant information for each bit.
for (i, result) in enumerate(results):
    n_true = 0
    n_false = 0

    for answer in result:
        if answer == True:
            n_true += 1
        else:
            n_false += 1
```

```
    print "------------------------"
    print "# of true: " + str(n_true)
    print "# of false: " + str(n_false)
    print "actual bit of d: " + bin(d)[2:][i]
    guess = None
    if n_true/n_false > 1.0: # does this correctly capture signifigantly different?
        guess = 1
    else:
        guess = 0
    print "guess bit of d: " + str(guess)
    print "------------------------"
```

Output:

```
------------------------
# of true: 0
# of false: 100
actual bit of d: 1
guess bit of d: 0
------------------------
------------------------
# of true: 45
# of false: 55
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 77
# of false: 23
actual bit of d: 1
guess bit of d: 1
------------------------
------------------------
# of true: 73
# of false: 27
actual bit of d: 1
guess bit of d: 1
------------------------
------------------------
# of true: 70
# of false: 30
actual bit of d: 1
guess bit of d: 1
------------------------
------------------------
```

```
# of true: 56
# of false: 44
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 53
# of false: 47
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 62
# of false: 38
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 80
# of false: 20
actual bit of d: 1
guess bit of d: 1
------------------------
------------------------
# of true: 59
# of false: 41
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 66
# of false: 34
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 75
# of false: 25
actual bit of d: 1
guess bit of d: 1
------------------------
------------------------
# of true: 82
# of false: 18
actual bit of d: 1
guess bit of d: 1
```

```
------------------------
------------------------
# of true: 61
# of false: 39
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 64
# of false: 36
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 61
# of false: 39
actual bit of d: 0
guess bit of d: 0
------------------------
------------------------
# of true: 77
# of false: 23
actual bit of d: 1
guess bit of d: 1
------------------------
```