Quant. Comp. HW - 2

Steven MacCoun

Oct. 18, 2005

1 Simon's Problem

2 Modular Exponentiation

Here was my python code:

```
#Modular Exponentiation
import math

def modular_exponentiation(base, exponent, modulus):
         c = 1
         for e_prime in range(1, exponent+1):
                    c = (c * base) % modulus
        return c

print modular_exponentiation(1234, 1234*1234, math.pow(10, 10))
```

And the output was:

$$\boxed{3102217216.0}$$

3 RSA Misuse

I first tried to solve this as strictly a math problem, but had little success, in large part because I thought that the gcd(e1, e2) was somehow irrelevant to the problem. However, I noticed that normally the exponents are the same value when performing RSA, so I scoured google to see if there was some well known attack where you have a common modulus with different attacks. Turns out that it is fairly well documented, and Simmons wrote a paper on it a while back.

The basic idea is: Since

$$gcd(e_1, e_2) = 1$$

, then

$$\exists u, v \; s.t. \; e_1 * u + e_2 * v = 1$$

You can use this to fact to find M:

$$(M^{e_1} mod n)^u (M^{e_2} mod n)^v = M^{e_1 * u + e_2 * v} mod n = M mod n$$

4 Prime factorization

Problem: Consider n=121932632103337941464563328643500519

(a) How many bits is n?

```
print len(str(121932632103337941464563328643500519))
```

Output:

$$\boxed{36}$$

(b) Find if n is prime with program that runs in less than one second.

```
def miller_rabin_pass(a, s, d, n):
         a_to_power = pow(a, d, n)
         if a_to_power == 1:
                  return True
         for i in xrange(s-1):
               if a_to_power == n - 1:
                        return True
               a_to_power = (a_to_power * a_to_power) % n
        return a_to_power == n - 1

def miller_rabin(n):
        #compute s and d
```

```
        d = n - 1
        s = 0
       while d % 2 == 0:
                d >>= 1
                s += 1

        #Run several miller_rabin passes
        for repeat in xrange(20):
                a = randint(2, n-1)
                if not miller_rabin_pass(a, s, d, n):
                        return False
        return True

print miller_rabin(n)
```

(b)
(c)
(d)
(e)