1. Use n; e and d from the last homework. Suppose that you try random a to factor n. Try 100 random a’s. How many of them allow you to factor n? Estimate the probability of success.

Code (SageMath):

︠n= 1259531756783983515701499777642110356794201569384295868500005799617750548880147110509521944049285041602433244172023804646590835427723055191592144638318476432867385429617360121

d= 879829162542850074748838973716462641470292321076843078870413133138541894315167534655428516005898396122103324293925057981802023330186106794090644952807381680714475934931163153

e = 65537

R = Integers(n)

suc\_crt = 0

k = e\*d-1

flag = 0

for i in range(100):

if (k%2==0):

k = k//2

else:

break

for i in range(100):

a = randint(1,n);

p = gcd(power\_mod(a,k,n)-1,n)

if(p!=1 and p!=n):

suc\_crt = suc\_crt + 1

print "The probability of success is: "+str(suc\_crt)+"%"

Result:

The probability of success is: 46%

2. Examine the certificates of your browser, and find the RSA public key n and e (in decimal) for <https://www.google.com>.

Steps (Using Google Chrome):

Open google website-----F12-----View Certificate-----Certificate Path---Details-----Public key

The public key is RSA (2048 Bits):

d8 c2 32 55 db 74 b0 94 4d a6 af 94 2a 7d 27 e1 60 b1 05 6b 45 9b f1 4b 26 ec 09 a9 7a b1 82 e5 f7 f5 ed 94 b5 13 81 d5 d2 09 a4 f7 63 2f ff 74 6a b9 eb 79 5a 8f 8f 61 54 cc 0c 57 1e b1 b4 15 7f b6 82 17 cc 6d 94 0e cb 9b b8 01 45 f2 1f 85 32 89 eb a6 a5 f8 1e f7 c6 22 96 66 2a 97 fd 8c 2a da 0d ee 8a 56 a9 f8 55 06 c5 3c 2d 00 dc f6 ed 35 86 3a 78 e0 31 4e 36 83 7e 17 0a 6d 53 90 ee 8f f8 d0 79 cf d0 a2 4a 5f df d1 6a 5a 62 04 4b 42 3d e3 d3 01 e1 6d b3 2b e6 24 ff 7c a0 40 7c 43 a8 2e 16 7c 50 09 94 5e ae 7a c1 30 d8 f7 9b 65 8b 88 ac 54 11 a4 19 c8 d4 69 eb 20 ab 0d e0 6f 98 b7 b5 5b 46 39 b6 e7 18 40 55 f5 88 30 2e 24 05 9d 66 2d f8 6d 08 75 d9 f3 c6 2d 9d 8d ff 07 57 97 55 c1 a1 9c b9 9c d5 01 97 7e 81 b7 b6 23 e3 12 15 0a 53 58 ce 47 b7 81 6d 17 6d f3 (In hexadecimal)

Code (SageMath to convert n into decimal):

s=" "

p\_k\_dec=int(s,16)

print p\_k\_dec

Result:

27363235796357477967947954422762696643808881019696516505802146333573365190241480818259543443618198051202365432711086234502015175164666605662360427020346941099098530520281807237088859956074718992401103805568876399737097347108738804311653367183269919397890155997196402994809226104095574816840603528410500288012435220029258191751157767234389694918452290007870967090866964986423332679104449526150312128199770284030638863911399885964780921626530028835043699599959011942531755513660470072559583547464621766629345727179529725280446589818750406835518314370192161035390608660937830006561853510300636066137179954122296254754291

E (24 Bits):

65537

3. Suppose that we decide to use e = 65537 as the RSA public exponent. Can we use prime numbers that are congruent to 1 (mod e) to generate n? Why? Find a prime p satisfying:

• p ≡ 1 (mod e);

• 21000 ≤ p ≤ 21004;

• The first 9 decimal digits of p is your ID number.

Explain your approach.

1. We cannot use this number to generate n:

Assuming p is the generator of n, then we know that n=p\*q

If p=1(mod e), then p-1 divides e, then p-1=k\*e(k>=1)

According the definition of keys: gcd(e, (p-1)(q-1))=1

However, here we have gcd(e, (p-1)(q-1))= gcd(e, k\*e\*(q-1))=k\*(q-1)!=1

Hence, we cannot use this number to generate n

1. Find prime p:

Code(SageMath):

e = 65537

id = 113383597

P = 10^293\*id

#Find the first value p that p=1(mod n)

while (P < 2^1001):

if(mod(P,e)==1):

print P

break

else:

P = P + 1

#Find the value p that p=1(mod n) and is a prime number

while (P < 2^1001):

if(P.is\_prime()):

print 'The prime number p can be:\n'+str(P)

P = P + e

else:

P = P + e

Result:

11338359700000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000054912

The prime number p can be:

11338359700000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000023189473