Using Sage we can simulate an RSA Encryption and Decryption.

sage: # randomly select some prime numbers

sage: p = random\_prime(1000); p

191

sage: q = random\_prime(1000); q

601

sage: # compute the modulus

sage: N = p\*q

sage: R = IntegerModRing(N)

sage: phi\_N = (p-1)\*(q-1)

sage: # we can choose the encrypt key to be anything

sage: # relatively prime to phi\_N

sage: e = 17

sage: gcd(d, phi\_N)

1

sage: # the decrypt key is the multiplicative inverse

sage: # of d mod phi\_N

sage: d = xgcd(d, phi\_N)[1] % phi\_N

sage: d

60353

sage: # Now we will encrypt/decrypt some random 7 digit numbers

sage: P = randint(1,127); P

97

sage: # encrypt

sage: C = R(P)^e; C

46685

sage: # decrypt

sage: R(C)^d

97

sage: P = randint(1,127); P

46

sage: # encrypt

sage: C = R(P)^e; C

75843

sage: # decrypt

sage: R(C)^d

46

sage: P = randint(1,127); P

3

sage: # encrypt

sage: C = R(P)^e; C

288

sage: # decrypt

sage: R(C)^d

3

Also, Sage can just as easily do much larger numbers:

sage: p = random\_prime(1000000000); p

114750751

sage: q = random\_prime(1000000000); q

8916569

sage: N = p\*q

sage: R = IntegerModRing(N)

sage: phi\_N = (p-1)\*(q-1)

sage: e = 2^16 + 1

sage: d = xgcd(e, phi\_N)[1] % phi\_N

sage: d

237150735093473

sage: P = randint(1,1000000); P

955802

sage: C = R(P)^e

sage: R(C)^d

955802