Building an RSA-key generation:

DUE 12/7/2017
Submission format: Submit your code plus 5 output samples into the RSA-key generation folder in Cocalc

---

To build an RSA crypto system, you first need to find two random primes. Your primes will consist of 7 bits. You will set the first and the last bits to be 1 and randomly choose one-by-one the internal 5 bits. We could have used only 7 bits to store it, but to simplify the program, store it as a standard 32 bit integer, so there will be 25 leading 0 bits.

You may use any pseudorandom routine you like to generate random numbers. Very likely, your programming language has one. To generate a random bit, get a (pseudo-)random number, and extract the least significant bit to be your random bit.

Print, for one of the random numbers you got, produce a trace showing how the 5 individual bits were obtained, so show your 5 random numbers and the extracted bit for each.

You will use the variant of the Miller-Rabin algorithm (and not as it is generally presented) to test if your 7-bit number n is a prime. You will pick some random a's, such that $0 < a < n$. You can do it either by a process similar to getting a candidate prime above, or to simplify your program, you can just get a pseudorandom number and "cut it down to size" by computing its remainder modulo n, and discarding it if it is 0. If your number passes the Miller-Rabin test for 20 values of a, you may declare it as prime. For completeness, make sure that one of the n's turned out not to be a prime number. So, if you immediately find the prime numbers you need, pick any number you know not to be prime and perform the test on it. If the test says it is possibly a prime, look for another number that you know not to be a prime.

Print, for one n that turned out not to be prime and the a for which the answer was "not prime," produce a trace.

Print, for one n that turned out to be prime and one a for which the answer was "perhaps prime," produce a trace.

Now, given two primes p and q you found, you will get $n = p \times q$. Note that p and q should be different, so you need to check for this. In a "real" search for large number the probability that $p = q$ is so small that there is no need to check for this, but as you aee working with very small numbers, you need to check for that.

Pick a small number to be the public key e. It has to be relatively prime with $\varphi(n)$. To check if it is relatively prime and to find a multiplicative inverse to serve as the private key d, use the Extended

Euclidean Algorithm, which you will code. If e is not relatively prime with φ(n), then find another small number. Do not do it randomly, start with 3 and go up until you find an appropriate e. In the extremely unlikely case (which would not happen in a real RSA implementation), that all the values of e you try do not work, just pick another pair of random primes and continue from there. And if the smallest e you find is big which here means bigger than φ(n) that is also fine for your project.

Show how you used the algorithm on the various candidates for e until you got the right one. (If you are lucky, the first e you tried, worked—this is fine too.). Produce a trace for each e just as we had in class for the Extended Euclidean algorithm. For the value of e found, find the corresponding value of d and normalize it so that it is positive and smaller than φ(n). (If it happens that d = e, this will be fine too for your project, though of course not in a real RSA system.) Print, list, the value of d. So, the public key of Alice, is really the pair ⟨n, e⟩ and the private key of Alice is ⟨n, d⟩; only she knows the latter, more precisely only she knows d. but you need to check for it in your implementation

Print a line, list the following for Alice, both as integers and as sequences of bits: p,q,n,e,d

SAMPLE OUTPUT:

```
b_5|01110010110011100011001010010111|1
b_4|11011001110101100101010111100001|1
b_3|11010010010000100100011111100001|1
b_2|00101101101011111101100010000101|1
b_1|11100000110001100100110010110000|0

Number|125|00000000000000000000000001111101

n=125, a=28
i |xi |z |y |y
6 |1 |1 |1 |28
5 |1 |28 |34 |77
4 |1 |77 |54 |12
3 |1 |12 |19 |32
2 |1 |32 |24 |47
1 |0 |47 |84 |84
0 |0 |84 |56 |56
125 is not prime because 28^124 mod 125 != 1

n=79, a=15
i |xi |z |y |y
6 |1 |1 |1 |15
5 |0 |15 |67 |67
4 |0 |67 |65 |65
3 |1 |65 |38 |17
2 |1 |17 |52 |69
1 |1 |69 |21 |78
0 |0 |78 |1 |1
79 is perhaps prime
```

```
e = 3
i |qi |r |ri+1 |ri+2 |si |ti
1 |1584 |4752 |3 |0 |1 |0
e = 4
i |qi |r |ri+1 |ri+2 |si |ti
1 |1188 |4752 |4 |0 |1 |0
e = 5
i |qi |r |ri+1 |ri+2 |si |ti
1 |950 |4752 |5 |2 |1 |0
2 |2 |5 |2 |1 |0 |1
3 |2 |2 |1 |0 |1 |-950
4 | |1 | | |-2 |1901

d = 1901
p = 67, q = 73, n = 4891, e = 5, d = 1901
p = 00000000000000000000000001000011
q = 00000000000000000000000001001001
n = 00000000000000000001001100011011
e = 00000000000000000000000000000101
d = 00000000000000000000011101101101

p = 107, q = 89, n = 9523, e = 3, d = 6219
p = 00000000000000000000000001101011
q = 00000000000000000000000001011001
n = 00000000000000000010010100110011
e = 00000000000000000000000000000011
d = 00000000000000000001100001001011
```