CMPUT 299, Winter 2018, Assignment 9

All assignment submissions must conform to the Assignment Submission Specifications *posted on eClass. Ensure that your submission follows these specifications before submitting your work.*

You will produce a total of three files for this assignment: “hackRSA.py” for problem 1, “a9.pdf” or “a9.txt” for problems 2 and 3, and a README file (also in either PDF or plain text). **If your code requires any external modules or other files to run, include them in your submission as well. Your submission should be self-contained.**

Problem 1

The security of the RSA cipher depends on there being an infinitely many prime numbers: generating large prime numbers is, as far as we know, far easier than factoring the products of these primes, meaning that, no matter how fast computers become, we will always be able to find products of larger prime numbers than potential cryptanalysts can factor. In short, the number of keys we can produce is growing much faster than the number of keys that can be hacked in a reasonable amount of time.

If the number of primes were finite, this would not be true; we would eventually “run out” of primes to use in our RSA keys, and computers could eventually become fast enough that any product of two prime numbers could be factored quickly.

In this exercise, you will put this idea into practice by building a RSA cipher hacker which works under the assumption that there are finitely many prime numbers. This shows that the infinitude of primes, as proven by Euclid, is necessary for RSA to be a secure cipher.

As you know from reading chapter 24, in the public key consisting of n and e, n is a product of two prime numbers, p and q, and e is relatively prime with $(p-1)\times(q-1)$. All that is needed to break a message encrypted with n and e is d, the modular inverse of e modulo $(p-1)\times(q-1)$. If we know what $(p-1)\times(q-1)$ is, we can do this easily, using the “findModInverse” function in “cryptomath.py”.

The real-world strength of RSA comes from the fact that, since there is no known efficient method of factoring a number such as n, discovering p and q is hard, and therefore discovering $(p-1)\times(q-1)$ is hard. We know we need to find the inverse of e mod *something*, but we would need to factor n to determine what that *something* is.

Suppose, for example, that were are no prime numbers greater than one million. This would imply that there are exactly 78498 prime numbers, and we could list them all quickly using the “primeSieve” function in “primeSieve.py”. Since p and q would have to be on this list, we could find them by simply dividing n by every number on the list. This means that, under this supposition, *RSA ciphers could be easily hacked by brute force!*

Create a module called “hackRSA.py” which contains a function “finitePrimeHack”. The “finitePrimeHack” function should take 3 parameters: a threshold t, and n and e, which together form a public RSA key. The function should assume that there are no prime numbers larger than t. It should then discover the values of p and q, compute the value of d (the inverse of e modulo $(p-1)\times(q-1)$) and returns a list containing p, q, and d in that order, assuming $p \leq q$ (that is, the smaller of the two primes should come first; this constraint is simply to fix the ordering of primes in your solution).

For example:

```
>>> hackRSA.finitePrimeHack(100,493,5)
[17, 29, 269]
```

Problem 2 Short answer

Included with the assignment are five public key files, each specifying a value of n and e, generated using the "makeKeyFiles" function from the textbook code. Using your code from Problem 1, for each of the five key files, determine the values of p, q, and d (that is, the two primes whose product is n, and the remaining part of the private key). Again, to fix the order of the primes, assume $p \leq q$. Report your results in a table, like the one below (obviously, your table should have the values filled in), in your a9.pdf or a9.txt.

pubkey file	p	q	d
a9-0_pubkey.txt			
a9-1_pubkey.txt			
a9-2_pubkey.txt			
a9-3_pubkey.txt			
a9-4_pubkey.txt			

Problem 3 Short answer

Included with the assignment are five ciphertexts, created using the "encryptAndWriteToFile" function from the textbook code, encrypted with the public keys from Problem 2. Decipher them, and include the plaintexts, in order, in your a9.pdf or a9.txt.