**USC UPSTATE**

**CSCI 455: Computer Security**

**Spring 2019**

**Lab 4**

**RSA** is a public encryption scheme. Its security assumptions are based on complexity theory: computing the product of two prime numbers is easy (polynomial time), but there is no efficient algorithm for factoring them back (so far, all factorization methods are in the non-polynomial class).

The keys for the RSA algorithm are generated the following way:

1. Choose two different large random prime numbers ***p*** and ***q***
2. Calculate ***n = pq***
   * ***n*** is the modulus for the public key and the private keys
3. Calculate the totient: ***phi =(p-1)(q-1)***.
4. Choose an integer ***e*** such that 1 < ***e*** < ***phi***, and ***e*** is coprime to ***phi***
   * ***e*** is released as the public key exponent
5. Compute ***d*** to satisfy the congruence relation ***d\*e ≡ 1 mod phi (the remainder of d\*e / phi is 1)***
   * ***d*** is kept as the private key exponent

The public key is made of the modulus ***n*** and the public (or encryption) exponent ***e***.

The private key is made of the modulus ***n*** and the private (or decryption) exponent ***d*** which must be kept secret.

Encrypting a message: ***c = m ^ e (mod n)***

Decrypting a message: ***m = c ^ d (mod n)***

Example:

p = 29, q = 31

n = p \* q = 29 \* 31 = 899

phi = (p -1) \* (q – 1) = (29 – 1) \* (31 – 1) = 840

e = 11

d \* e ≡ 1 mod phi => (d \* 11) / phi will give us a remainder of one.

(611 \* 11) = 6721 and 6721 / 840 = 8 with remainder 1 => d = 611

C = M^e mod n

C = 119^11 mod 899 = 595

M = C^d mod n

M = 595^611 mod 899 = 119

**Problem 1**: Decrypting RSA with Known factorization

You have the ciphertext as follows. In order to decrypt it, you need to factorize n into p and q, compute phi and find d. Then we can find the original message m.

c = 28822365203577929536184039125870638440692316100772583657817939349051546473185

n = 70736025239265239976315088690174594021646654881626421461009089480870633400973

e = 3

Note:

1. you can do known factorization here: <http://www.factordb.com>.
2. Useful gmpy2 functions:

*invert(e, phi)* returns d such that d \* e == 1 modulo phi, or 0 if no such y exists.

*powmod(x, y, n)* returns (x^y mod n).

*mul(x,y)* returns x \* y.

**Problem 2**: Decrypting RSA with Fermat Factorization

Implement and try out [Fermat's Factorization Algorithm](https://en.wikipedia.org/wiki/Fermat%27s_factorization_method)! Then try to break the following RSA key and obtain the original message m.

c = 654564125967811572957608485461509223541781197895608920296825435452302563551217882689453762450350456257099687251554693360645992257362168460115089842875072530869254099617858153458510730488327127628978127748004507636893613507344065845140647694349616219705757465949239924311260160127009283418952554522720051840260714703523494071411559772701875928237248989122625648657235677768486515417771976078417365256201505968603934443986411140514722785883888625061210731765750448

n = 1209143407476550975641959824312993703149920344437422193042293131572745298662696284279928622412441255652391493241414170537319784298367821654726781089600780498369402167443363862621886943970468819656731959468058528787895569936536904387979815183897568006750131879851263753496120098205966442010445601534305483783759226510120860633770814540166419495817666312474484061885435295870436055727722073738662516644186716532891328742452198364825809508602208516407566578212780807

e = 65537

Note: Useful gmpy2 functions

* *is\_square(x)* returns True if x is a perfect square, False otherwise.
* *isqrt(x)* returns the integer square root of an integer x. x must be >= 0.