Assignment RSA

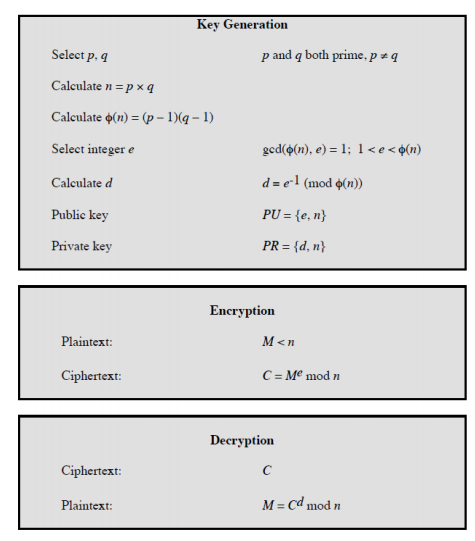
**Learning Objectives:**

* Explore the RSA algorithm, and write a RSA cracking program <https://brilliant.org/wiki/rsa-encryption/>
* Understand message decomposition and construction.

**Background Information Preamble**

* The RSA algorithm paper (1977):

<https://people.csail.mit.edu/rivest/Rsapaper.pdf>



You may use the Extended Euclidian Algorithm by hand, or use something like this applet: <http://bilhamil.com/cryptoapps/ExtendedEuclideanAlgorithm.html> to help calculate your private key *d*, and check your work.

The extended Euclidean theorem tells us that for:

*ax + by = gcd(a,b)*

It can be shown that:

* “a” is 
* “b” is *e*
* “y” is *d ,* our private key.

In other words:

*x + ed = gcd(a,b) = 1*

See examples from lecture notes.

# Part 1: How hard can it be?

Cracking a RSA code (given only public key e, n) requires these steps:

1. Find 2 prime-factors, of the public key modulus.
2. Now you have p, and q; Then, calculate the totient Ø.
3. Now you have (e, Ø). So, you can use Extended Eulers or iterative method to find “d”.

Let’s have a crack at it yourself, step by step.

You can use any programming language.

1. Write a function ***isPrime(integer num)***

that returns true if “num” is prime, and false otherwise.

The algorithm involves seeing if the number is divisible by all the prime numbers less than

Some numbers you can test with: 779, 817, 893, 1007, 1121, 1159, 1273, 1349, 1387, 1501, 1577, 1691, 1843, 667, 713, 851, 943, 989, 1081, 1219

1. Write a function ***primeFactor(integer n)***

that takes input the integer modulus “n”,

and find all the prime-factors of that number.

This is called prime-factorization.

If there are only 2 non-zero prime-factors, we are in luck (in terms of cracking RSA!)­­­

Some numbers you can test: 923, 949, 1027, 1079, 1157, 1261, 323, 391, 493, 527, 629, 697, 731, 799, 901, 1003, 1037, 1139, 1207, 1241, 1343, 1411, 1513, 1649, 437, 551, 589, 703, 779, 817, 893

1. Now let’s suppose you are working for the “good guys”, and given: “e”, and “Ø”. Write a function ***calculatePrivateKey (integer e, integer Ø)***

that calculates “d”(the private key).

Some (e, Ø) pairs you can test with: (131, 328), (79, 328), (23, 84),

(113, 576), (157, 168), (37, 84), (277, 576), (499, 576)

1. Finally, using the functions you have already done, create a program that takes 2 input parameters, “e”, and “n”.

***rsacrack {e} {n}***

***Usage example: rsacrack 3 187***

You input the public key (e, and n), and it provides a private key “d” (ie, crack the code, using brute force!)

(FYI: By 1996, 40-bit keys could be cracked in about 4 hours by a cluster of workstations)

Some (e, n) pairs you can test with: (131, 415), (79, 415), (37, 129),

(23, 129), (113, 679), (157, 215), (277, 679), (499, 679)

Provide bash (or make file) that runs with some sample inputs.

Submit zip folder.

# Part 2: Encrypt and Decrypt a Message

You want to send the password “LOL” (three 8-bit asci characters) to your BTech friend, whose public key is (7,187). Since the modulus is 187, use 7 bits for your message blocks. Show work.

|  |  |
| --- | --- |
| Message encoded into bits: |  |
| Message separated into 7-bit messages: | M1 =  M2 =  M3 =  M4 = |
| Encrypt each message using public key: | C1 =  C2 =  C3 =  C4 = |
| Suppose your friend’s totient (phi) is 160, what is the private key? |  |
| Decrypt the cipher messages above to check your work: | M1=  M2=  M3=  M4= |