# Cryptography Homework 5a—Create an RSA Private/Public Key Pair in Python

## Required reading

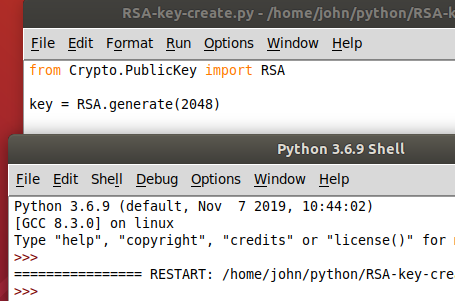
Crypto 5 slides

In this lab you will create an RSA public/private key pair using the PyCryptodome library in Python. This key pair will not include all the data it needs to be a digital certificate. We will cover digital certificates later, but the main purpose of a certificate is to securely publish a public key. We will “securely publish” our public keys in the next lab by giving them to our lab partners in person.

## Generate an RSA key

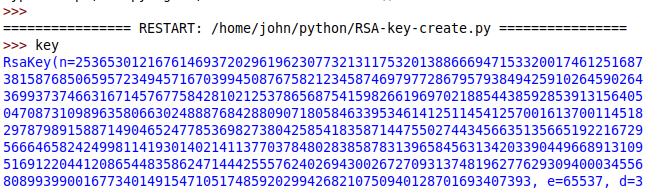
We will use the example in <https://pycryptodome.readthedocs.io/en/latest/src/examples.html>, in the section “Generate public key and private key.” This will create a file with a private key, stored in the file private.pem or whatever you choose to name it, and a public key stored in public.pem.

Note that the previous example on the site, “Generate an RSA key,” generates the private key and protects the private key with a password. This is a common practice; in order to use the private key or see its contents you must use the password. We will assume you can keep your private key secure and will not put a password on our private key.

Use the following code to create an RSA key. It assumes that your installation of PyCryptodome in the earlier lab was successful.  
from Crypto.PublicKey import RSA  
key = RSA.generate(2048)  


The code ran and created a key but we haven’t told it to print anything so there is no output.

## Examine the RSA key

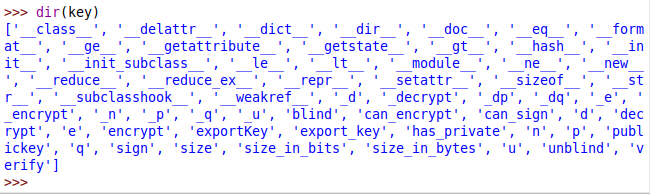
Use the Idle Shell window, or your terminal, to look at the contents of the key variable.  
  
<snip>

You should see:  
RsaKey(n= a really long number,  
 e=65537,  
 d= a really long number,  
 p= a really long number,  
 q= a really long number,  
 u= a really long number)

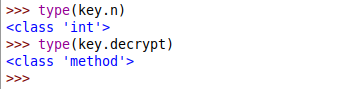
Those numbers (except for u) should be familiar to you, since you’ve studied the slides in the Cryptology 4 Module, Intro to Public Key Encryption.

* N is the modulus and should be about 2048 bits long (about 616 digits in base 10). It is created by multiplying the random primes p and q.
* e is the exponent that will be used for the public key. According to NIST the public key shouldn’t be any smaller than 65537, so that’s the number PyCryptodome chooses by default
* d is the exponent that will be used for the private key, and it should be long
* p and q are the prime numbers that were chosen to create n. They each should be about 1024 bits long.
* u is the inverse of p mod n. u = p-1 mod q. It is used to make the RSA calculations faster. You may see Chinese Remainder Theorem (CRT) associated with it, but that’s beyond the scope of this course.

### Some Python

Execute the command dir(key).  


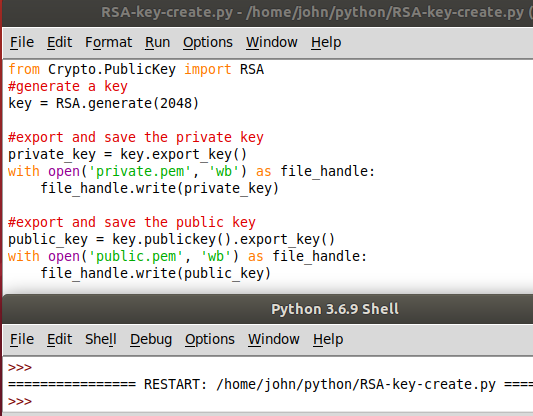
In Python, the dir(object) command shows all the properties and methods associated with object. The terms that begin and end with double underscores (‘\_\_class\_\_’ for example) are normally for system use or for advanced users and are called “dunder” variables.

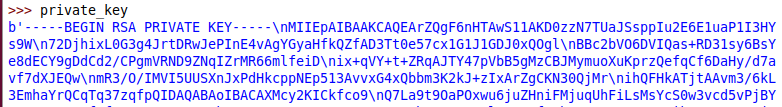
There are several terms in dir() that will be useful to us, like n, p, q, decrypt, encrypt, export\_key, size\_in\_bits, and more. To see what those terms are, use the type() command. For example, n is an integer parameter and decrypt is a method.  


## Public and Private Keys

Once you create a key pair, you often need to use the private and public keys repeatedly. For that reason, you will often save the keys to files. The process of creating public and private keys from the key is called exporting.

There are several formats for saving keys. The one that PyCryptodome uses is .pem, which is described here. <https://serverfault.com/questions/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file> . It is a binary file and is base64 encoded so that it may be sent through a text-only medium.

Execute the following code. It uses the “with open” method for saving the files so we won’t need to worry about forgetting to close the files. We haven’t asked for output yet, so there shouldn’t be any.  


In the Python Shell type private\_key and public\_key to see what they look like. You should see that they are base64 encoded with text headers and footers (----END PUBLIC KEY----, etc.) If you remove the headers and footers you can decode the base64, but you’ll end up with binary data.  
  
<snip>

## Hand In

Create a key, then export and save both the public and private keys to files. To answer these questions, you will need the gcd and findModInverse functions from cryptomath.py.

1. Verify your answer to question 1. Read the public.pem file you saved into a variable (for example, “pub”,) then convert it to a key with the method RSA.import\_key(pub). What does the public key contain? Do the same thing for the private key. What does the private key contain?
2. The private\_key and public\_key variables both contain base64 encoded text. The private key is much longer than the public key. What does the private key need to have that the public key does not (and should not) have?
3. Check to see if the RSA key generation agrees with what was covered in the Crypto 4 module. Extract p, q, and d from the private key (private\_key.p, private\_key.q, etc.)
   1. Compute n = p \* q. Is the result the same as private\_key.n ?
   2. Compute L = lcm( (p-1) \* (q-1) ) Remember that   
      lcm(a, b) = a \* b // gcd(a, b)
   3. Compute d = e-1 mod L. Is the result the same as private\_key.d ?   
      Note: e-1 mod L is the same as findModInverse(e, L)

Note: The lecture notes use lambda = lcm( (p-1) \* (q-1) ). In Python lambda is a reserved word, so I’ve changed it to L.

Note: The Pycryptodome computes the inverse of e using lambda, and not phi = (p-1)\*(q-1) because it is more efficient.