Homework 11: RSA Implementation

Now that we know how to produce random large primes quickly and easily (March 24th lecture) we can implement RSA encryption in Python.

RSA key generation

Let's begin with RSA keys, both public and private. For this we will need the Miller-Rabin code as well as the extended Euclidean algorithm. We illustrate a worked example in Python's REPL:

```
>>> from MillerRabin import *
>>> from xgcd import *
>>> a = 2**500; b = 2**501 # set upper & lower limits
>>> p = findPrime(a,b)
>>> q = findPrime(a,b)
>>> N = p*q
```

The theory of RSA says that we should pick an exponent e such that $\gcd(e, \varphi(N)) = 1$, where $\varphi(N) = (p-1)(q-1)$. It is common, but not required, to pick $e = 65537$. Whatever e you pick, make *sure* that it is relatively prime to $\varphi(N)$. Notice that 65537 is a prime number, so it is more likely to be relatively prime to $\varphi(N)$. Then we need to compute $d = e^{-1} \pmod{\varphi(N)}$.

```
>>> phi = (p-1)*(q-1) # Euler's phi(N)
>>> e = 65537 # a common choice
>>> gcd(e,phi) # check that e is invertible mod phi(N)
1
>>> g,x,y = xgcd(e,phi) # extended EA
>>> d = x % phi # should be the inverse of e mod phi(N)
>>> e*d % phi == 1  # check that it works
True
>>> public_key = (e,N)
>>> private_key = (d,N)
```

In case you are curious, the actual numerical values of the keys computed above are shown below:

```
>>> public_key
(65537, 2363249279436650963156489446903734168938059158248308336143929674130580836057087310610443419
360570873106104434192719017312753712261618897181078367314514335277493957530891677423341209492164314
774233412094921643143921128117921868382943383533141778108855038069185858856790161252290665767758730
612522906657677587301829651892237562413271367402777280763221142545903504 1)
>>> private_key
(73442937690596564027355876169321686709433527436476030154330249588808672792429768306762441247403485910074440031187280297697957961870922775946325439319863240460868473437120480041657372462759063791097545404311930124559745125550750667015177492331703870108230690226462743686663047581350518156690641520623, 236324927943665096315648944690373416893805915824830833614392967413058083605708731061044341927190173127537122616188971810783673145143352774939575308916774233412094921643143921128117921868382943383533141778108855038069185858856790161252290665767758730182965189223756241327136740277728076322114254590350411)
```

Note that I inserted line feed breaks in the above numbers in order to make them fit on one page without overflowing into the margin.

RSA encryption and decryption

Now that we have our RSA keys, it is trivial to implement RSA encryption and decryption. Pick any number x less than N. This number represents your plaintext message, or part of it. The RSA theory says that the corresponding ciphertext is $y = x^e \pmod N$.

```
>>> x = 12345 # test plaintext message
>>> y = modpower(x,e,N) # corresponding ciphertext
```

Also, RSA decryption is given by $x = y^d \pmod N$. Let's test that.

```
>>> deciphered = modpower(y,d,N)
>>> deciphered == x  # did it work?
True
```

Notice the use of the stand-alone `modpower` function in both RSA encryption and decryption. While this is not so important when the exponent (e for instance) is small, it is absolutely necessary when the exponent is very large (as d will be here).

Representation

There is one remaining problem to be solved before we have a complete system: we need a method of converting arbitrary text into (a sequence of) integers. We need this because messages are given as strings while RSA encryption and decryption algorithms operate on integers (residues less than N). Let's agree to call this problem the *representation problem*. Actually, we face this problem in any public-key cryptosystem.

Given a string of characters, how can we represent it as an integer? One way is to use Python's builtin `ord` function. If you look this up, you will see that if `c` is a character in some string then `ord(c)` returns an integer between 0 and 1114111. Note that here I'm talking about Python 3, which uses Unicode to encode characters of strings; Python 2 was different. Anyway, the inverse function of `ord` in Python is the function `chr`, which accepts as input any integer between 0 and 1114111 and returns the equivalent character whose Unicode representation is defined by that number.

Given a string `s` of characters, we can apply the `ord` function to each character to obtain a list of integers. If s_k is the kth character, then let's set $d_k = \mathrm{ord}(s_k)$. Then we can define an integer representation of the string s by the rule

$$\mathrm{rep}(s) = \sum_k d_k R^k = \sum_k \mathrm{ord}(s_k) R^k$$

where $R = 1114112$ is the **radix** (number of distinct symbols) of the Unicode system. Then $\mathrm{rep}(s)$ is an integer representation of the string s.

We can also go backwards, from the integer $z = \mathrm{rep}(s)$ back to the original string $s = \mathrm{unrep}(z)$. To do that, we apply the division algorithm (long division) to write z (uniquely) in the form

$$z = qR + r, \qquad \text{where } 0 \le r < R.$$

Needless to say, q and r are integers. Setting $s_0 = \mathrm{chr}(r)$, we then have $s = s_0 \| \mathrm{unrep}(q)$, where $s = \mathrm{unrep}(z)$ is the original string and the symbol $\|$ is the mathematical symbol for concatenation. This is a recursive description, but we know how to turn recursion into iteration.

The above considerations lead to the Python code in the file `represent.py`, which is discussed in greater detail in the presentation on this topic posted for Tuesday 24th March on Sakai. You should view that presentation now, if you have not already done so. What we are doing is *almost* equivalent to expressing z in base R; "almost" because there is an order reversal in the above idea that is slightly easier to work with in code.

The Unicode system is truly an astounding human accomplishment. It is a system (actually, several systems) of binary encoding of all the symbols of all human languages, both past and present! This includes all of the dead languages that we know about. Read the Wikipedia article on this fascinating topic if you wish to learn more.

We are still not done with this topic. The problem is that if the string s is too long, then its representation $\mathrm{rep}(s)$ can be larger than N, the modulus for our RSA system. If that ever happens then our RSA breaks down, so we have to ensure that it can never happen. The solution is easy: if s is too long then we simply break it into blocks of characters each short enough (say of length B) to satisfy the requirement $\mathrm{rep}(s) < N$, and apply RSA encryption and decryption functions block by block. A little thought reveals that

$$\text{the maximum length } B \text{ of each block is the integer part of } \log_R(N)$$

because we just need to make $R^B < N$, as $\mathrm{rep}(s) < R^B$ if the length of s is B.

The above considerations lead to the code for `replist` and `unreplist` functions, given in the file `represent.py`. The function $\mathrm{replist}(s, B)$ accepts a string s and a blocksize B as inputs and returns a list of integer representations of the blocks (of length B) of the string s. The function $\mathrm{unreplist}(l)$ performs the inverse operation, turning its input list l back into a string.

```
>>> from represent import *
>>> s = "I can't be 100% sure!" # my test string
>>> x = rep(s)
>>> unrep(x)  # should give s back again
"I can't be 100% sure!"
```

If you are curious to see what integer value x has here, you can simply evaluate to print it, as usual:

```
>>> x
28647859164562768939182761642792495735948631406714064568544357153113689129800075
58838537258955941401026636343208963219652681
```

Note that once again I inserted an extra linefeed in the number to make it possible to display on a printout. We can also, if we desire, convert our string s into a sequence of integers by using $\mathrm{replist}(s, B)$ with a chosen blocksize.

```
>>> mylist = replist(s,3) # block length 3
>>> mylist
[122883344957513, 48408698945633, 121642099409012, 60821067530341,
45926138773552, 145225857302560, 40961215627378]
>>> unreplist(mylist) # should return the original string
"I can't be 100% sure!"
>>>
>>> mylist = replist(s,2) # block length 2
>>> mylist
[35651657, 108068963, 43450478, 35651700, 112525410, 54591520, 53477424,
35651621, 130351219, 112525426, 33]
>>> unreplist(mylist)
"I can't be 100% sure!"
```

Notice that the `unreplist` function does not need to know the block size.

Summary of RSA implementation

Putting everything together, we get a setup for using RSA to send and receive messages of arbitrary length, for arbitrary Unicode strings. Suppose that Bob wishes to send a secret message to Alice, whose public RSA key is (e, N). We assume that only Alice knows the corresponding secret key d. Given a string s, Bob does the following:

1. Compute the blocksize $B = \lfloor \log_R(N) \rfloor$.
2. Compute $x = \text{replist}(s, B)$.
3. For each integer in the list x, compute the corresponding $y = x^e \pmod N$. This results in a list of integers that we can call y.
4. Send the list y to Alice.

When Alice receives the list y from Bob, she uses her secret key d corresponding to the public exponent e to compute:

5. For each integer in the list y, compute the corresponding $x = y^d \pmod N$. Call the resulting list x.
6. Compute $\text{unreplist}(x)$, which is the original message string s.

If Alice desires to send a reply message to Bob, she uses *Bob's public key* to encrypt it, repeating the above steps 1–4. When Bob receives her message, he uses steps 5–6 with *his* corresponding secret key to decrypt the message.

Assignment

In a new file, define three Python functions for implementing RSA. At the top of the file, include import statements to import needed code from other files as appropriate. The functions should have the following names and parameters:

```
def RSAkeygen(bitsize):
    """returns a triple (e,d,N) defining RSA public and private keys,
    where N = p*q is a product of two random primes of roughly bitsize
    bits each"""

def RSAencrypt(e,N,s):
    """returns a list y of RSA encrypted integers, where s is a string,
    using the given keypair (e,N)"""

def RSAdecrypt(d,N,y):
    """returns a string s, where y is a list of RSA encrypted integers,
    using the given keypair (d,N)"""
```

Test your code by generating RSA keys, and encrypting and then decrypting various test messages. For example, you might try encrypting and decrypting the following French text, in which all the various accents and line feeds should be preserved:

```
msg = """
Contrairement à une opinion répandue, le Lorem Ipsum n'est pas simplement du
texte aléatoire. Il trouve ses racines dans une oeuvre de la littérature
latine classique datant de 45 av. J.-C., le rendant vieux de 2000 ans. Un
professeur du Hampden-Sydney College, en Virginie, s'est intéressé à un
```

```
des mots latins les plus obscurs, consectetur, extrait d'un passage du
Lorem Ipsum, et en étudiant tous les usages de ce mot dans la littérature
classique, découvrit la source incontestable du Lorem Ipsum. Il provient en
fait des sections 1.10.32 et 1.10.33 du "De Finibus Bonorum et Malorum"
(Des Suprêmes Biens et des Suprêmes Maux) de Cicéron. Cet ouvrage, très
populaire pendant la Renaissance, est un traité sur la théorie de l'éthique.
Les premières lignes du Lorem Ipsum, "Lorem ipsum dolor sit amet...",
proviennent de la section 1.10.32.

L'extrait standard de Lorem Ipsum utilisé depuis le XVIè siècle est reproduit
ci-dessous pour les curieux. Les sections 1.10.32 et 1.10.33 du "De Finibus
Bonorum et Malorum" de Cicéron sont aussi reproduites dans leur version
originale, accompagnée de la traduction anglaise de H. Rackham (1914).
"""
```

This is an example of a triple-quoted string in Python. You can Google it to learn more. Note that you may need to use Python's `print` function to format this string for nice printing in the terminal. If you are feeling ambitious, try a passage in another language, for instance try Chinese:

```
msg =
"""
"示界康藝流灣燈葉就人開服拿里提下服了開才的費上小她性入調看毛病寫可不對不戲又陸
沒節使英便文報，小坐他被上與：不木病觀展事，為養連研院，到人我高散喜代小化老上
三子小著？了生者產無沒斯平世服會如且氣響化上問的時對散如天……城條型男、樣少導麼
活，導山生什本意樹度多人看土安消還時持人，題能當用，記身量我算成光我但的人眼動
有片營：手市政亮：長子友銀著的現重世把好集我公應市不學。因議商亞說媽笑相的出招
檢著家導導的你，裡習著得的不養設治起立半的助孩式一望南這天乎舉回興孩至間管過冷
價；是結他政麼紅，看必間，客東和仍得興全原見有寫仍定感言百上是以間不青在有加遊
康中廣來當會苦推飛成看為須學天懷位那育真……安病上黑量，美紀從樂我：局日政，他黨
有險任孩，說加之，到農多。國對紀醫多不操，都土常小目力小考去麼：北近速亞的麗國
……後所軍總會招象的除制，太了坡發如最、或司一大老理年入為？動共東司做，局車中人
再山，更眼經頭要中關容出上們開中養！
"""
```

When you are satisfied, commit your changes and push them to Github as usual.