Criptografía: Aspectos Teóricos y Prácticos — Laboratorio 2: Funciones auxiliares para RSA

En este laboratorio y el próximo implementarán la especificaciones oficiales de RSA:

http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf

En la primera parte se implementarán todas las funciones auxiliares enunciadas en las especificaciones. Se entregará esta primera parte el **jueves 29 de mayo**. Lo entregarán por EVA y **lo harán en equipos de por lo menos dos integrantes**.

Teoría de números. Tienen que implementar y escribir las siguientes funciones:

1. XMCD y MCD:

```
def mcd(a, b)
    '''
       Devuelve el máximo común divisor de a y b
    '''

def xmcd(a, b):
    '''
       El algoritmo de Bezout para  a y b. O sea, devuelve u, v, g
       tales que:

          g = mcd(a, b)
          a * u + b * v = g
    '''
```

2. Miller-Rabin y generación de primos:

```
def es_primo(n, rnd=default_pseudo_random, k=DEFAULT_ITERATION):
    '''
       Testea si n es (probablemente) primo

       n - el entero para testear
       rnd - el generador de números aleatorios que se usa
       k - el número de iteraciones para testear n


       Devuelve: True si n es (probablemente) primo y False
       en otro caso.
    '''

def hallar_primo(size=128, rnd=default_crypto_random,
    k=DEFAULT_ITERATION):
    '''
       Genera un número (probablemente) primo de tamaño dado

       size - tamaño del primo en bits
       rnd - el generador de números aleatorios que se usa
       k - el número de iteraciones para verificar si un
       candidato es primo
```

```
        Devuelve: un número (probablemente) primo de tamaño dado
    '''
```

Funciones de RSA

3. `os2ip` e `i2osp` que están definidas en páginas 8 y 9 de las especificaciones:

```
def os2ip(x):
    '''
       Convierte la cadena x de bytes representando un entero
       big-endian a un entero
    '''

def i2osp(x, x_len):
    '''
       Convierte un entero x a su representante big-endian de
       largo x_len
    '''
```

4. `rsaep` y `rsadp` que están definidas en páginas 10 y 11 de las especificaciones:

```
def rsaep(m, n):
        '''
           Determina si la representante del mensaje es menor que
           el módulo
        '''

def rsadp(c,m):
        '''
           Determina si la representante del texto cifrado es
           menor que el módulo
        '''
```

5. `mgf1` que está definida en página 50 de las especificaciones. Usé la biblioteca `hashlib`:

http://pymotw.com/2/hashlib/#module-hashlib

```
import hashlib

def mgf1(mgf_seed, mask_len, hash_class=hashlib.sha1):
    '''
       Mask Generation Function v1 de la PKCS#1 v2.2 estándar

       mgs_seed - la semilla, una cadena de bytes
       mask_len - lo largo del mask
       hash_class - la función hash para usar, el default es SHA1

       Return value: un mask pseudo-aleatorio, como una cadena
       de bytes
    '''
```

6. Bajar el archivo

   ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1-vec.zip

   e implementar código que se podrá usar para ver si los resultados de lo que vas a implementar en el próximo laboratorio estén bien. En particular, debería cargar los datos, convertir los números en hexadecimal a números en decimal y manejar los datos para que se pueda automatizar el chequeo de lo que implementan. Se puede suponer que las funciones de encriptar y desencriptar están dadas por

   ```
   def rsaes_oaep_encrypt((n,e), M, L)
   ```

   ```
   def rsaes_oaep_decrypt(K, C, L)
   ```

   y definidas como en páginas 18 y 21 de la especificaciones.