Project 5: Cryptography

1 The problem

The objective of this program is to make a program that encrypts a text (in lower case) `text1` using an auxilliary text `text2` and RSA encryption. The idea is to first transform `text1` into a list of pairs of numbers where each pair of numbers corresponds to one letter in `text1`. The pair is on the form `[line,pos]` and chosen so that the letter in `text1` coincides with the letter at position `pos` in line `line` of the text `text2`.

if `text1='code'` and

```
text2 = ['In Xanadu did Kubla Khan \n',
 'A stately pleasure-dome decree: \n',
 'Where Alph, the sacred river, ran \n',
 'Through caverns measureless to man \n']
```

then an example of a list would be `list=[[1,26],[1,20],[0,7],[1,6]]`.

Then every number in the list is encrypted using RSA encryption. This follows the following algorithm (from Wikipedia):

1.1 RSA Key generation

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

1. Choose two distinct prime numbers p and q. For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits' to make factoring harder. Prime integers can be efficiently found using a primality test. You can also find lists of primes on internet, google "large primes".

2. Compute $n = pq$. n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute $\varphi(n) = (p-1)(q-1) = n - (p+q-1)$, where φ is known as "Euler's totient function". This value is kept private.

4. Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$; i.e., e and $\varphi(n)$ are coprime.

5. Determine d as $d \equiv e^{-1} (\mathrm{mod} \varphi(n))$ i.e., d is the *modular multiplicative inverse* of e (modulo $\varphi(n)$). This is more clearly stated as: solve for d given
$$de \mod \varphi(n) \equiv 1.$$

e is released as the public key exponent. d is kept as the private key exponent. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\varphi(n)$ must also be kept secret because they can be used to calculate d.

1.2 Encryption

Given an integer m to encrypt compute

$$c(m) = m^e \mod n$$

c is the encrypted value. Observe that both e and n are public

1.3 Decryption

Given an encrypted value c compute

$$m(c) = c^d \mod n.$$

Here the exponent d is the private key.

2 The project

The project consists in writing three different programs, that can code using the public key, decode using the private key or (attempt) to crack the code, i.e. find the private key if the public key is provided. The three parts are as follows:

1. Write a program that reads a file `code.param` containing the parameter p, q and e as well as the names of the files containing `text1` and `text2` here is an example

   ```
   $ cat code.param
   61 53 17
   to_code.txt
   KublaKhan.txt
   ```

   Reading this file with your program should result in $p = 61$, $q = 53$, $e = 17$, `text1` read from the file `to_code.txt` and `text2`read from the file `KublaKhan.txt`. It should then transform the text `text1` to a list of pairs of integers using `text2` and the algorithm described in the introduction and encode using the public key given by $n = pq$ and e. The encoded list should be written to a file called `out.code`.

2. Another program should read a file `decode.param` of the form

   ```
   $ cat decode.param
   61 53 17
   out_code.txt
   KublaKhan.txt
   ```

   the file with the encrypted list, `out.code` as well as the file with `text2`.

   Using the private key n, d the code should then reconstruct the original message `text1`.

3. Finally write a program that given a public key n, e finds the private key d. Of course this is only possible for moderately sized public keys. You may need to read up on prime factorization, but can easily design a simple brute force algorithm yourself.

3 What to submit for the assessment

You should submit the following items in a zipped folder:

- A .pdf document (cover letter) with details on the members of the group, names and student numbers. Here you may also give some information on the project, such as if some of the extensions have been considered.
- Three well commented codes called `code.py`, `decode.py` and `crack.py` that produce the expected outputs as outlined in the discussion above.
- The file `code.out` containing the encoded list of pairs of numbers for the text, (given in the file `to_code.txt` on moodle),

  ```
  i have a degree from the university of oblivion
  and am empty-handed as the shirt on the clothesline
  ```

  using $n = 3233$, $e = 17$ and the text in the file `KublaKhan.txt'` (these are the parameters in the parameter file `code.param` on moodle).
- An execution of the program `decode.py` on the data in `code.out` obtained in the previous point, using the private key in the file `decode.param` on moodle.

- An execution of the code `crack.py` that finds the private key, for the public key: $n = 32193613398841823$ and $e = 17$. Then use this key to decode the encrypted text in `madrigal.code` on moodle, return the result.

4 Further considerations

4.1 Some code to speed things up

A naive implementation of certain steps in the algorithm above may lead to very slow coding or decoding. Should you observe this problem, you might want to import a module with the following functions (I leave it to you to figure out what these functions do):

```
def fast(x,e,m):
    X = x
    E = e
    Y = 1
    while E > 0:
        if E % 2 == 0:
            X = (X * X) % m
            E = E/2
        else:
            Y = (X * Y) % m
            E = E - 1
    return Y

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

4.2 Possible extensions

1. read up on *padding schemes* and implement one to make your code safer.

2. change your code so that it can handle both upper and lower case, as well as numbers and special characters such as ',' '.' '!' '?' etc. You have to change the text coding algorithm.

3. If you are interested in learing more about RSA encryption you can download an essay by Burt Kaliski at:
`http://www.mathaware.org/mam/06/Kaliski.pdf`