

Introduction to Programming in Python

Assignment 4 (RSA Cryptosystem) Discussion

Introduction

The RSA cryptosystem involves three integers n , e , and d that satisfy certain mathematical properties

The *public key* (n, e) is made public on the Internet, while the *private key* (n, d) is only known to Bob

If Alice wants to send Bob a message $x \in [0, n)$, she encrypts it using the function $E(x) = x^e \bmod n$, where $n = pq$ for two distinct large prime numbers p and q chosen at random, and e is a random prime number less than $m = (p - 1)(q - 1)$ such that e does not divide m

Example: suppose $p = 47$ and $q = 79$; then $n = 3713$ and $m = 3588$; further suppose $e = 7$; if Alice wants to send the message $x = 2020$ to Bob, she encrypts it as $E(2020) = 2020^7 \bmod 3713 = 516$

When Bob receives the encrypted message y , he decrypts it using the function $D(y) = y^d \bmod n$, where $d \in [1, m)$ is an integer that satisfies the equation $ed \bmod m = 1$

Continuing the example above, if $d = 2563$, then when Bob receives the encrypted message $y = 516$ from Alice, he decrypts it to recover the original message as $D(516) = 516^{2563} \bmod 3713 = 2020$

Problem 1 (RSA Library)

Implement a library called `rsa.py` that provides functions needed for developing the RSA cryptosystem and supports the following API

<code>keygen(lo, hi)</code>	generates and returns the public/private keys as a tuple (n, e, d) , picking prime numbers p and q needed to generate the keys from the interval $[lo, hi)$
<code>encrypt(x, n, e)</code>	encrypts x (int) using the public key (n, e) and returns the encrypted value
<code>decrypt(y, n, d)</code>	decrypts y (int) using the private key (n, d) and returns the decrypted value
<code>bitLength(n)</code>	returns the least number of bits needed to represent n
<code>dec2bin(n, width)</code>	returns the binary representation of n expressed in decimal, having the given width and padded with leading zeros
<code>bin2dec(n)</code>	returns the decimal representation of n expressed in binary

```
× ~/workspace/rsa_cryptosystem
```

```
$ python3 rsa.py S
encrypt(S) = 1743
decrypt(1743) = S
bitLength(83) = 7
dec2bin(83) = 1010011
bin2dec(1010011) = 83
```

Problem 1 (RSA Library)

`keygen(lo, hi)`

- Get a list of primes from the interval $[lo, hi)$
- Sample two distinct random primes p and q from that list
- Set n and m to pq and $(p-1)(q-1)$, respectively
- Get a list primes from the interval $[2, m)$
- Choose a random prime e from the list such that e does not divide m (you will need a loop for this)
- Find a $d \in [1, m)$ such that $ed \bmod m = 1$ (you will need a loop for this)
- Return the tuple¹ (n, e, d)

`encrypt(x, n, e)`

- Implement the function $E(x) = x^e \bmod n$

`decrypt(y, n, d)`

- Implement the function $D(y) = y^d \bmod n$

¹A tuple is like a list, but is immutable. You create a tuple by enclosing comma-separated values within matched parentheses, eg, `a = (1, 2, 3)`. If `a` is a tuple, `a[i]` is the i th element in it

Problem 1 (RSA Library)

`_primes(lo, hi)`

- Create an empty list
- For each $p \in [lo, hi)$, if p is a prime, add p to the list
- Return the list

`_sample(a, k)`

- Create a list b that is a copy (not an alias) of a
- Shuffle the first k elements of b
- Return a list containing the first k elements of b

`_choice(a)`

- Get a random number $r \in [0, l)$, where l is the number of elements in a
- Return the element in a at the index r

Problem 2 (Keygen Program)

Write a program called `keygen.py` that receives *lo* (int) and *hi* (int) as command-line inputs, generates public/private keys (n, e, d), and writes the keys as standard output, separated by a space

```
× ~/workspace/rsa-cryptosystem
```

```
$ python3 keygen.py 50 100  
3599 1759 2839
```

Problem 2 (Keygen Program)

Receive *lo* (int) and *hi* (int) as command-line inputs

Get public/private keys as a tuple

Write the three values in the tuple, separated by a space

Problem 3 (Encryption Program)

Write a program called `encrypt.py` that receives the public-key n (int) and e (int) as command-line inputs and a message to encrypt as standard input, encrypts each character in the message, and writes its fixed-width binary representation as standard output

```
× ~/workspace/rsa_cryptosystem
```

```
$ python3 encrypt.py 3599 1759
```

```
CS110
```

```
<ctrl-d>
```

```
000110000000010011010100001010100011001010100011001110000110010111100100
```


Problem 3 (Encryption Program)

Receive public-key n (int) and e (int) as command-line inputs

Get the number of bits per character (call it *width*) needed for encryption, ie, number of bits needed to encode n

Receive *message* to encrypt as standard input

For each character c in *message*

- Use the built-in function `ord()` to turn c into an integer x
- Encrypt x
- Write the encrypted value as a *width*-long binary string

Write a newline character

Problem 4 (Decryption Program)

Write a program called `decrypt.py` that receives the private-key n (int) and d (int) as command-line arguments and a message to decrypt (produced by `encrypt.py`) as standard input, decrypts each character (represented as a fixed-width binary sequence) in the message, and writes the decrypted character as standard output

```
× ~/workspace/rsa_cryptosystem
```

```
$ python3 decrypt.py 3599 2839
000110000000010011010100001010100011001010100011001110000110010111100100
<ctrl-d>
CS110
$ python3 encrypt.py 3599 1759 | python3 decrypt.py 3599 2839
Python is the mother of all languages.
<ctrl-d>
Python is the mother of all languages.
```

Problem 4 (Decryption Program)

Receive private-key n (int) and d (int) as command-line inputs

Get the number of bits per character (call it *width*)

Accept *message* (binary string generated by `encrypt.py`) as standard input

Assuming l is the length of *message*, for $i \in [0, l - 1)$ and in increments of *width*

- Set s to substring of message from i to $i + \text{width}$ (exclusive)
- Set y to decimal representation of the binary string s
- Decrypt y
- Write the character corresponding to the decrypted value, obtained using the built-in function `chr()`