Proj 5: Cracking a Short RSA Key (10 pts. + 40 pts. extra credit)

What you need:

- Any computer with Python 2.7. If you are using a Mac or Linux machine, Python is already installed. If you are using Windows, follow these instructions to install Python 2.7.

Purpose

To break into RSA encryption without prior knowledge of the private key. This is only possible for small RSA keys, which is why RSA keys should be long for security.

Summary

Here's a diagram from the textbook showing the RSA calculations.

RSA Key Generation

Output: public key: $k_{pub} = (n, e)$ and private key: $k_{pr} = (d)$

1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that

$$\gcd(e, \Phi(n)) = 1.$$

5. Compute the private key d such that

$$d \cdot e \equiv 1 \bmod \Phi(n)$$

Problem Statement

Meghan's public key is (10142789312725007, 5). Find her private key.

1. Factoring n

Finding the Square Root of n

n = 10142789312725007. This is the product of two prime numbers, p and q.

How large are p and q? Well, they can't both be larger than the square root of n, or they'd be larger than n when multiplied together.

Start Python in interactive mode. On a Mac or Linux box, you can do that by typing this command into a Terminal window:

```
python
```

Execute these commands:

```
import math
n = 10142789312725007
print math.sqrt(n)
```

The square root prints out, as shown below.

```
>>> import math
>>> n = 10142789312725007
>>> print math.sqrt(n)
100711416.0
>>>
```

Displaying More Decimal Places

It's not clear from that output whether the result is an integer, or just rounded off to one decimal place. To see more decimal places, we'll use the repr() function.

Execute this command:

```
print repr(math.sqrt(n))
```

Now more decimal places appear, as shown below.

```
>>> print repr(math.sqrt(n))
100711415.99999976
>>>
```

Testing 20 Candidates

So one of the prime factors must be a prime number less than 100711415. All we have to do is try dividing n by odd numbers starting at 100711413 and going down until we get an integer result. (We don't need to test 100711415 because it's divisible by 5 and therefore not a prime number.)

A good way to do this is to calculate n mod c, where c is a candidate. If c is a factor of n, the result will be zero.

We can test the first 20 candidates with a for loop.

Execute these commands:

```
c = 100711413
for i in range(c, c-40, -2):
  print i, n%i
```

Press Enter twice after the last command to terminate the loop.

The third candidate is the winner, with a remainder of zero, as shown below.

```
>>> c = 100711413
>>> for i in range(c, c-40, -2):
...   print i, n%i
...
100711413 100711373
100711411 100711387
100711409 0
100711407 32
100711405 72
100711403 120
100711401 176
100711399 240
100711397 312
100711395 392
100711393 480
100711391 576
100711389 680
100711387 792
100711385 912
100711383 1040
100711381 1176
100711379 1320
100711377 1472
100711375 1632
>>>
```

Calculating q

We now know p and we can calculate q.

Execute these commands:

```
p = 100711409
q = n / p
print p, q, n, p*q, n - p*q
```

The calculation worked, so the last value is zero, as shown below.

```
>>> p = 100711409
>>> q = n / p
>>> print p, q, n, p*q, n - p*q
100711409 100711423 10142789312725007 10142789312725007 0
>>>
```

2. Compute phin = (p-1) * (q-1)

Execute these commands:

```
phin = (p-1) * (q-1)
print p, q, n, phin
```

The parameters print out, as shown below.

```
>>> phin = (p-1) * (q-1)
>>> print p, q, n, phin
100711409 100711423 10142789312725007 10142789111302176
>>>
```

3. Installing the "gmpy" Library

This library allows Python to perform multiple-precision arithmetic. The steps to install it are different for each OS. Follow the appropriate steps below.

Installing gmpy on Mac OS

First install brew as explained here:

https://brew.sh/

Open a **new Terminal window**, not the one running Python, and execute this command to download and install a few dependencies and gmpy:

```
brew install gmp mpfr mpc
pip install gmpy
```

One student said "pip" did not work. so he did this:

```
brew install python
```

And thereafter used "python2.7" in place of "python".

Installing gmpy on Kali

In a Terminal window, execute this command:

```
apt install python-gmpy
```

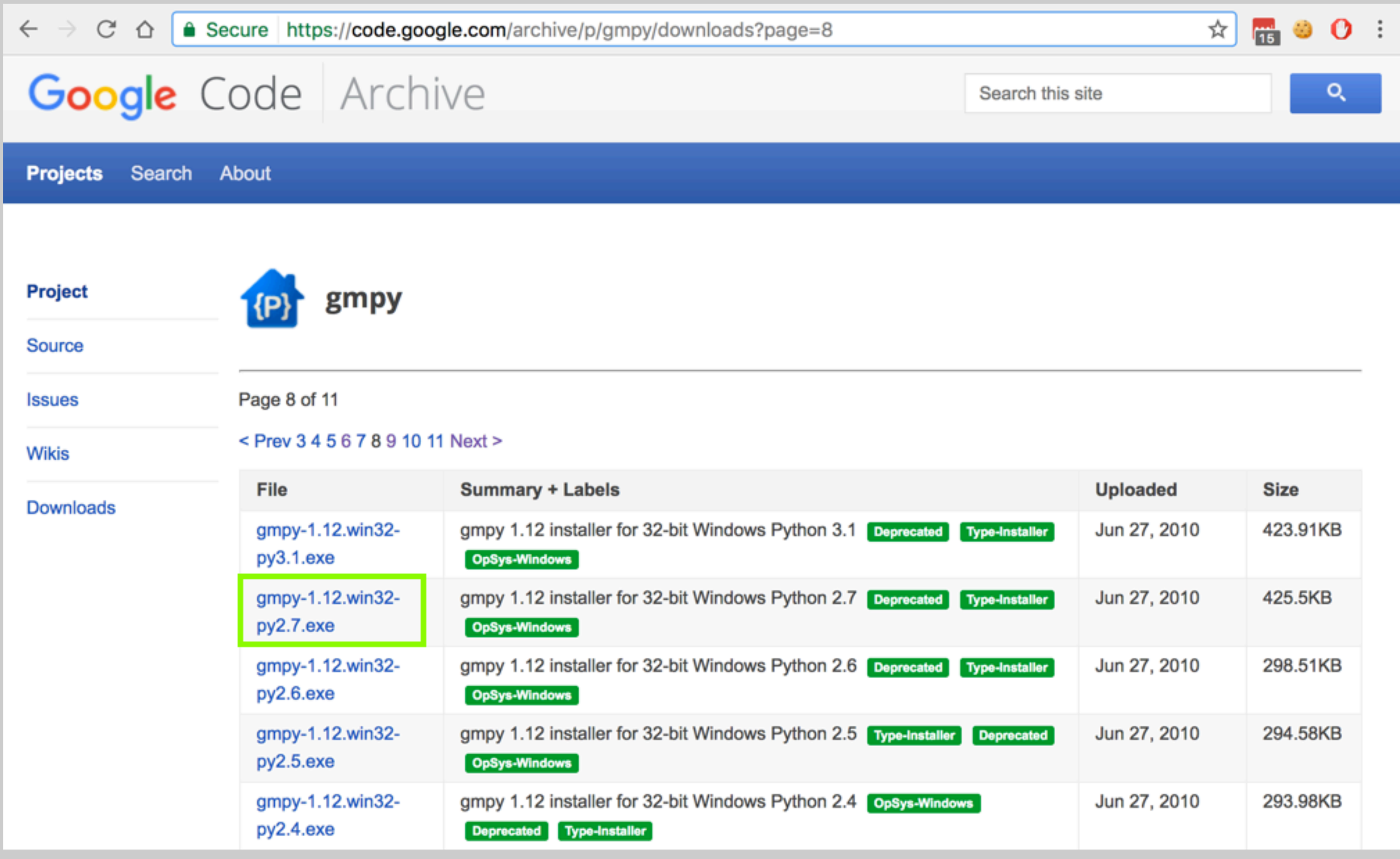
Installing gmpy on Windows

If you are using the 32-bit Windows Server 2008 system provided by your instructor, you need to use a very old version of gmpy.

In a Web browser, go to:

https://code.google.com/archive/p/gmpy/downloads?page=8

Download **gmpy-1.12.win32-py2.7.exe**, as shown below. Double-click this file and install it with the default options.



If you are using a later version of Windows, you might need a more recent version of gmpy from this page . These downloads are "wheel" files, and must be installed from a command prompt with the "pip install" command.

4. Compute Private Key d

We need to find a **d** with this property:

(d * e) mod phin = 1

d is the "multiplicative inverse" of e.

We know that e = 5 from the Problem Statement.

It's not obvious how to find d, but there's a simple way to do it in Python, using the "gmpy" library.

In the Terminal window running python, execute these commands.

```
e = 5
import gmpy
d = gmpy.invert(e, phin)
print d, e, d*e %phin
```

We get the value of d, and, to verify it, we see that d*e %phin is indeed 1, as shown below.

```
>>> e = 5
>>> import gmpy
>>> d = gmpy.invert(e, phin)
>>> print d, e, d*e %phin
81142312890417 41 5 1
>>>
```

5. Encrypting a Message

Encoding the Message as a Number

Cueball wants to send Meghan this message:

Hi!

We can only send numbers. Let's convert that message to three bytes of ASCII and then interpret it as a 24-bit binary value.

In the Terminal window running python, execute these commands.

```
x1 = ord('H')
x2 = ord('i')
x3 = ord('!')
x = x1*256*256 + x2*256 + x3
print x
```

We get the value of x, as shown below.

```
>>> x1 = ord('H')
>>> x2 = ord('i')
>>> x3 = ord('!')
>>> x = x1*256*256 + x2*256 + x3
>>> print x
4745505
>>>
```

Encrypting the Message with the Public Key

A public key contains two numbers: **n** and **e**. To encrypt a message *x*, use this formula:

$$x^e \bmod n$$

Execute these commands:

```
y = x ** e % n
print y
```

The encrypted message appears, as shown below.

```
>>> y = x ** e % n
>>> print y
247470077215935
>>>
```

6. Decrypting a Message

To decrypt a message, use this formula:

$$y^d \bmod n$$

Execute this command:

```
xx = y ** d % n
```

Python crashes, as shown below. It cannot handle such large numbers. (On Windows, you see a different error message saying "outrageous exponent". Execute the **exit()** command to exit Python.)

```
>>> xx = y ** d % n
gmp: overflow in mpz type
Abort trap: 6
Sams-MacBook-Pro:platform-tools sambowne$
```

To compute such a number, we must use the pow() function.

Execute these commands to restart python and restore all the values we calculated previously:

```
python
n = 10142789312725007
p = 100711409
q = 100711423
phin = (p-1) * (q-1)
e = 5
import gmpy
d = gmpy.invert(e, phin)
x1 = ord('H')
x2 = ord('i')
x3 = ord('!')
x = x1*256*256 + x2*256 + x3
y = x ** e % n
```

Your screen should look like the image below.

```
>>> n = 10142789312725007
>>> p = 100711409
>>> q = 100711423
>>> phin = (p-1) * (q-1)
>>> e = 5
>>> import gmpy
>>> d = gmpy.invert(e, phin)
>>> x1 = ord('H')
>>> x2 = ord('i')
>>> x3 = ord('!')
>>> x = x1*256*256 + x2*256 + x3
>>> y = x ** e % n
>>>
```

Let's try that decryption again with the pow() function. Execute these commands:

```
xx = pow(y, d, n)
print xx
```

Now it works, showing our original message in numerical form.

```
>>> xx = pow(y, d, n)
>>> print xx
4745505
>>>
```

Converting the Message to Readable Text

To convert that number back to letters, execute these commands:

```
xx1 = xx / (256*256)
xx2 = (xx - 256*256*xx1) / 256
```

```
xx3 = xx - 256*256*xx1 - 256*xx2
msg = chr(xx1) + chr(xx2) + chr(xx3)
print xx1, xx2, xx3, msg
```

Now it works, showing the original message in readable form, as shown below.

```
>>> xx1 = xx / (256*256)
>>> xx2 = (xx - 256*256*xx1) / 256
>>> xx3 = xx - 256*256*xx1 - 256*xx2
>>> msg = chr(xx1) + chr(xx2) + chr(xx3)
>>> print xx1, xx2, xx3, msg
72 105 33 Hi!
>>>
```

Challenge 5a: Encrypt "WOW" (10 pts.)

Using the same keys, encrypt this message:

```
WOW
```

Hint 1: The message, converted to a decimal number, is 7 digits long and ends in 43.

Hint 2: The encrypted message is 16 digits long and ends in 66.

Use the form below to put your name on the **WINNERS PAGE**.

Your Name (without spaces):

Encrypted message:

SUBMIT

Save a whole-screen image of the winners page showing your name with the filename "**YOUR NAME Proj 5a**".

Challenge 5b: Encrypt "OBEY!" (10 pts. extra credit)

Using the same keys, encrypt this message:

```
OBEY!
```

Hint 1: The message, converted to a decimal number, is 12 digits long and ends in 41.

Hint 2: The encrypted message is 16 digits long and ends in 81.

Use the form below to put your name on the **WINNERS PAGE**.

Your Name (without spaces):

Encrypted message:

SUBMIT

Save a whole-screen image of the winners page showing your name with the filename "**YOUR NAME Proj 5b**".

Challenge 5c: Message to Cueball (15 pts. extra credit)

Cueball's public key is:

```
(1110369753426018487552 21, 13)
```

Meghan sends this message to Cueball. Decrypt it.

```
80564890594461648564443
```

Use the form below to put your name on the **WINNERS PAGE**.

Your Name (without spaces):

Cleartext Message:

SUBMIT

Save a whole-screen image of the winners page showing your name with the filename "**YOUR NAME Proj 5c**".

Challenge 5d: Message to Rob (15 pts. extra credit)

Rob public key is:

(1234592592962967901296297037045679133590224789902207663928489902170626521926687, 5557)

Meghan sends this message to Rob. Decrypt it.

272495530567010327943798078794037733865151017104532777645776936985235709526002

Hint: make square root calculations more precise.

Use the form below to put your name on the **WINNERS PAGE**.

Your Name (without spaces):

Cleartext Message:

SUBMIT

Save a whole-screen image of the winners page showing your name with the filename "**YOUR NAME Proj 5d**".

Turning in your Project

Email the images to **cnit.141@gmail.com** with the subject line: **Proj 5 from YOUR NAME**.

Sources

Cracking short RSA keys

Prime Numbers Generator and Checker

Arbitrary precision of square roots

Posted 3-31-16 by Sam Bowne
Winners pages added 8-6-16
Attack server name updated 4-4-17
Added challenge and all renumbered 6-11-17
Added email and screen cap instructions 9-9-17
Added brew install 9-25-17
Pts fixed 10-3-17
Mac tip added 10-6-17