Day 33, RSA

One of the authors of our textbook is Ron Rivest. He and his colleagues used to spend their lunches trying to stump each other with cryptographic schemes. It turns out, they were ***very*** successful and developed the RSA scheme. Yes, the "R" in RSA is for Ron Rivest. The main idea is that it is very difficult to factor a large positive integer into its prime factors. In addition, the scheme makes use of many concepts involving modular arithmetic.

RSA is also an example of a ***Public Key*** cryptosystem. If Bob wants to send Alice a message, he would use her public key to encode the message, and then only Alice would be able to decrypt it, since only she has her private key.

In our work, you may find it helpful to factor an integer into its prime factors. There are many tools to do this, such as: https://www.calculatorsoup.com/calculators/math/prime-factors.php

It's probably helpful to work through a small example first, where Alice and Bob (standard names in cryptology for some reason), want to communicate, sending a message $\boldsymbol{M}$. For Alice, we'll call her public key P_A, and her private key S_A. Similarly for Bob, his public key is P_B, and his private key is S_B. As for all cryptographic schemes, RSA is invertible, which means that S and P are inverses:

$M = S_A(P_A(M))$ and $M = P_A(S_A(M))$, so $S_A = P_A^{-1}$ and $P_A = S_A^{-1}$

S_A is ***very*** difficult for anyone but Alice to compute.

		Bob encrypts		*Ciphertext*		*Alice decrypts*		
M	→	P_A	→	$C = P_A(M)$	→	S_A	→	M

So, what are the steps to create P and S?

1. Pick two large prime numbers, p and q, s.t. $p \neq q$
2. Compute $n = p * q$
3. Select a small odd integer, e, that is ***relatively prime*** to $(p-1)*(q-1)$, that is,
$$\boldsymbol{gcd}((p-1)*(q-1), e) = 1$$
4. Compute d as the ***multiplicative inverse of e modulo*** $((p-1)*(q-1))$
5. Publish $P = (e, n)$ as the public key
6. Keep $S = (d, n)$ as the secret key

Once we have the public and private keys, to encrypt a message, $P(M) = M^e \,\%\, n$, and to decrypt a message $S(C) = C^d \,\%\, n$.

For simplicity, we will encrypt and decrypt individual characters. And the numeric values that are encrypted/decrypted need to be smaller than n.

We've highlighted three terms above that we will discuss further.

gcd (a, b) is the greatest common divisor of a and b. The first approach is to decompose each of a and b into their prime factors, and take the product with the maximum exponent of each factor.

$$\gcd(24, 18) = \gcd(2^3 * 3^1, 2^1 * 3^2) = 2^1 * 3^1 = 6$$

But as we know, this is inefficient and computationally intensive.

The first recorded algorithm is attributed to Euclid, and it is to calculate the gcd. And interestingly, it is recursive:

```
gcd(a,b):
   if b=0:
     return a
   else:
      return gcd( b , a%b )
```

Applied to our earlier example, $\gcd(\,24\,,18\,) = \gcd(\,18\,,6\,) = \gcd(\,6\,,0\,) = 6$

When the gcd of two integers is equal to 1, the integers are said to be ***relatively prime***. For example, neither 21 or 16 are prime, but since $\gcd(\,21\,,16\,) = \gcd(\,16\,,5\,) = \gcd(\,5\,,1\,) = \gcd(\,1\,,0\,) = 1$, they are relatively prime.

a and b are congruent modulo n if they have the same remainder when divided by n. For example,
15 % 4 = 3, and 23 % 4 = 3, so 15 is congruent modulo 4 to 23, which we write as $15 \equiv 23\ \%\ 4$, or

$15 \equiv 23\ mod\ 4$.

So for an integer a, a ***multiplicative inverse of a, modulo m*** would be $a * \bar{a} \equiv 1\ \%\ n$.

What would an inverse of 3 mod 7 be, $3 * \bar{a} \equiv 1\ \%\ 7$?
If $\bar{a} = 5$,
$3 * 5 = 15$,
and, $15\ \%\ 7 = 1$, so 3 and 5 are multiplicative inverses mod 7.

What is another multiplicative inverse of 3 mod 7?
We can generate more by adding 7 to any previous $\bar{a}$, $3 * 12 \equiv 1\ \%\ 7$, 12, 19, and so on..

The multiplicative inverse $\bar{a}$ is reminiscent of $a * \frac{1}{a} = 1$.

Finally, we have the idea of a digital signature. A user could encrypt using their secret key, then anyone could decrypt using their public key to confirm the signature. This uses the fact that S and P are inverses. $S(\,P(M)\,) = P(\,S(M)\,) = M$

Original Signature → S → Signature text $= S$(Signature)

Only ***you*** can create your Signature text, since only you have your Secret key S .

Then anyone can use the Public Key P to decrypt, $P(\ S$(Signature) $\ ) =$ Original Signature

For example, we have a Digital Signature of `128, 367, 349, 115, 403` which was created with the secret key from our example, $S = (\,d\,,n\,)$, $S = (\,387\,,667\,)$. If we decrypt with the public key $P = (\,e\,,n\,)$, $P = (\,347\,,667\,)$, the resulting message is: `75, 114, 117, 115, 101`

75 114 117 115 101
K r u s e

For an inclass lab, let's step through an example using a Jupyter notebook. The following is adapted from: https://securitymouse.medium.com/learn-rsa-encryption-using-jupyter-notebooks-minus-the-jargon-d0484e2f6f8f .

0.
```
import random
import math
```

1. Pick two prime numbers, p and q, s.t. $p \neq q$
```
p=23
q=29
```

2. Compute $n = p * q$ and $phi = (p-1) * (q-1)$
```
n=p*q              # which is 667 in this example
phi=(p-1)*(q-1)    # which is 616 in this example
```

3. Select a small odd integer, e, that is ***relatively prime*** to $(p-1) * (q-1)$
```
e = random.randrange(1,phi)
while (math.gcd(e,phi) != 1):
    e = random.randrange(1,phi)
print("Computed value of e is", e)

e = 347 # Note, this value may differ due to the random
numbers used
```

4. Compute d as the ***multiplicative inverse of e modulo*** $((p-1) * (q-1))$
   First, define a function `modinverse`:
```
def modinverse(e, phi):
    e = e % phi
    for x in range(1, phi):
        if ((e * x) % phi == 1):
            return x
    return 1
```
   Then use this function to find d:
```
print(modinverse(e,phi))

d=387
```

5. The public key $P = (e, n)$ is $P = (347, 667)$

6. The secret key $S = (d, n)$ is $S = (387, 667)$

Note that RSA is relatively slow, so it is typically used to just encrypt the key for a different scheme.

Now, encrypt a short message (in ASCII) using the public key above.

h e y
104 101 121

```
message=[104,101,121]
c=[]
for num in message:
    c.append( (int(num)**e)%n )
print(c)
```

[331, 95, 564]

Next, decrypt the ciphertext using the secret key above.

```
plaintext=[]
for num in c:
    plaintext.append( str( (int(num)**d)%n ) )
outString=" ".join(plaintext)
print(outString)
```

104 101 121

So, if you intercepted a message that was created with a public key, how would you decrypt it?

You could use the public key, $P = (e, n)$, and factor n. That gives you p and q which you can use to calculate $phi = (p-1) * (q-1)$. If you know e and phi then you can calculate d (*Hint: **e** and **d** are multiplicative inverses, modulo **phi***), and then you will have the secret key, which you can use to decrypt the message.