# Entendiendo un algoritmo de criptografía asimétrica
# *Understanding RSA*

# Una operación asimétrica

- Que sea fácil de hacer pero difícil de deshacer...
  ... a no ser que se conozca una información (la clave privada)

- Elevar a una potencia es fácil    $c=m^x$ es fácil de calcular
  Verdad? Incluso aunque x sea un numero de 200 cifras?

- Encontrar el número m que cumpla que $c=m^x$ es difícil
  No parece tan difícil... seguro?
  Sabríais hacerlo si x tiene 200 cifras?

- Una dificultad mas: aritmética modulo n
  Solo existen los valores {0 ... n-1}
  n es equivalente a 0    x>n   es equivalente  x mod n  (x % n)
- Ejemplo en aritmetica modulo 7
  4+4= 8%7 =1     4*5= 20%7 =6
  2^0=1  2^2=4   2^3=8%7=1  2^4=2  2^5=5
  Las matemáticas funcionan casi igual :-)

# Un poco de matemáticas

- Elevar potencias en aritmética modulo n en cuanto el x es suficientemente grande al elevar $m^x$ desbordara varias veces n y es muy difícil de invertir
- En algunos valores de n hay ciertas propiedades que nos ayudan...
  ... cuando n=p*q p,q números primos
- Si n=p*q está demostrado que hay una curiosidad matemática
  Si elegimos un numero e que sea menor y coprimo con (p-1)*(q-1)
  [de hecho vale e cualquier numero primo menor que (p-1)(q-1)]
  En esas condiciones podemos encontrar otro numero d que cumpla que e*d=1 en aritmética modulo (p-1)(q-1)
  ...
  La curiosidad es que en esas condiciones el numero d nos ayuda a invertir la exponenciación en aritmética modulo n
  ...
  Es decir que si calculamos $m^e$ =c se cumple que $c^d$=m
- Con esto tenemos nuestro sistema de clave publica y privada

# Ejemplo

- Si p,q primos son 5 y 11 n=55
- (p-1)(q-1)=4*10=40 podemos encontrar e coprimo con 40
- Por ejemlpo e=7 hay mas cualquier primo menor que 40 vale
- Hay un d que cumple que e*d=1 mod 40
- d=23 7*23= 161%40 =1

Y podemos usar eso para invertir $m^7$

Ejemplo:

m=12 $20^7$=1280000000%55=15 $15^{23}$=1122274146401882171630859375%55=20

m=42 $42^7$=230539333248 % 55=48

$48^{23}$=46617444198218784202610668482287842099 2 % 55= 42

Esto ya se parece mucho a cifrar con una clave (7) y descifrar con otra (23)

Aunque en el fondo solo cifra numeros entre 0 y 54 en otro numero entre 0 y 54. Asi que no se tarda mucho en descifrar probando

# Eso era RSA

- Sólo que con números pequeños
- Recapitulando
  Un usuario genera una clave privada y una publica así
  Elegimos dos primos p y q   (un poco mas grandes que antes)
  Calculamos n=p*q   y  (p-1)*(q-1)=fi
  Elegimos e un numero primo menor que fi al azar
  Calculamos d que cumpla que e*d=1 modulo fi
  [Esto es fácil se hace con el algoritmo de Euclides]
  ...
  Nos olvidamos de p,q,fi y borramos todo rastro de ellos
  ...
  Guardamos n y e  y le llamamos clave publica
  Guardamos n y d  y le llamamos clave privada

# Eso era RSA

- Para cifrar algo lo dividimos en números menores que n
  [Por eso mas vale que elijamos un n grande o sea p y q grandes]
- Cifrar m
  $m^e \% n = c$ si n es grande es facil hacer la operacion pero muy difícil de deshacer solo con eso
- Descifrar c
  $c^d \% n = m$

- No parece difícil de programar las operaciones son simples (aunque no son tan simples con números grandes)
- Es fácil de programar pero no tan rápido de calcular porque son muuchas multiplicaciones
- Las dificultades están en generar los números de la clave

# Eso era RSA

- Para romperlo
  Si n es suficientemente grande
  [se usan p y q de unas 300 cifras n de unas 600 cifras o 2048 bits ]
  El atacante puede intentar adivinar d.
  d esta entre 0 y n asi que tiene que probar unos $2^{2048}$ combinaciones
  Es difícil sin saber (p-1)(q-1) que se uso para calcularlo
  Pero si los conoce puede usar el mismo algoritmo que se uso para encontrar d y entonces tendría la clave privada.
  Por eso se dice que el romper RSA es equivalente a descomponer un numero muy grande en dos factores primos

- Como curiosidad,
  Nunca se ha demostrado que no haya otra manera de romperlo.
  Se cree que es un problema NP-completo. Pero sólo es una conjetura.
  Está en el top-10 de los problemas matemáticos conocidos :-)

# Ejemplo educativo

- Un RSA con números pequeños se puede hacer fácilmente y tiene las mismas propiedades
  Salvo que será más fácil de romper :-)

- Usaremos p y q que generen un n<65535
  Así nos aseguramos que siempre el resultado cabe en 2 bytes
  Para cifrar elegiremos m de 1 byte y el resultado lo guardaremos en 2 bytes (el mensaje cifrado ocupa más que sin cifrar pero eso no es un problema)

# Funciones necesarias

- Cifrar y descifrar un numero entero

```
def cifra(m,e,n):
    c=(m**e)%n
    return c

def descifra(c,d,n):
    m=(c**d)%n
    return m
```

- Calcular d a partir de e (lo hacemos directamente por fuerza bruta)

```
def bruta_inverso(e,m):
    i=0
    while i<m :
        if (i*e)%m == 1:
            return i
        i+=1
    return False
```

# Cifrar y descifrar cadenas

```
class rsa:
    n=0
    x=0
    def __init__(self, filename):
        f=open(filename,'r')
        self.n=int(f.readline())
        self.x=int(f.readline())

    def cifrachar(self,s):
        m=ord(s)
        c=cifra(m,self.x,self.n)
        cstr='%c%c' % ( ((c&0x0000ff00) >> 8) , (c&0x000000ff) )
        return cstr

    def descifra2char(self,s):
        if len(s)>2 :
            s=s[0:2]
        if len(s)==1 :
            s=s+' '
        if len(s)==0 :
            s='  '
        c=ord(s[0])*256+ord(s[1])
        m=descifra(c,self.x,self.n)
        mstr='%c'%(m&0x00ff)
        return mstr

    def cifrastr(self,m):
        r=''
        for i in xrange(0,len(m)):
            r+=self.cifrachar(m[i])
        return r

    def descifrastr(self,c):
        r=''
        for i in xrange(0,len(c),2):
            r+=self.descifra2char(c[i:(i+2)])
        return r
```

- Ir leyendo una cadena byte a byte (enteros menores que 256)
  Y generando dos bytes por cada uno

# Generando claves

```
#!/usr/bin/env python

from rsa_tiny import *
from sys import exit

p=331
q=181
e=83
outfile='k1'

n=p*q
fi=(p-1)*(q-1)

d=bruta_inverso(e,fi)

if not d:
    print('Error no puedo calcular d')
    exit(-1)

print('p=%d'%p)
print('q=%d'%q)
print('n=%d'%n)
print('fi=%d'%fi)
print('e=%d'%e)
print('d=%d'%d)

print('guardando clave privada en '+outfile)
f=open(outfile,'w')
f.write('%d\n'%n)
f.write('%d\n'%d)

print('guardando clave privada en '+outfile+'.pub')
f=open(outfile+'.pub','w')
f.write('%d\n'%n)
f.write('%d\n'%e)
```

```
$ ./gen_rsa_tiny_key.py
p=331
q=181
n=59911
fi=59400
e=83
d=2147
guardando clave privada en k1
guardando clave privada en k1.pub
```

# Conclusiones

- Funcionamiento de RSA
  Fácil de entender y muy usado