Week 01 Lab solutions

Applied Cryptography (6G6Z0024)

Killian O'Brien

Import the Sympy library to provide the `synmpy.gcd`, `sympy.floor`, `sympy.mod_inverse` functions and others.

See accompanying videos on Moodle for explanations of these solutions.

In [1]:
```
import sympy as sp
```

Q1

In [2]:
```
sp.gcd(710,310)
```

Out[2]: 10

The version of `mygcd` below implements the tasks 1 and 2 of Q1.

In [3]:
```
def mygcd(a,b):
    a = abs(a)
    b = abs(b)
    if b==0:
        return a
    elif a==0:
        return b
    else:
        q = sp.floor(a/b)
        r = a - q*b
        #s = 'The int div with remainder is '+str(a)+'='+str(q)+'*'+str(b)+'+'+s
        #print(s)
        return mygcd(b,r)
```

In [4]:
```
mygcd(710,-310)
```

Out[4]: 10

In [5]:
```
a = 23478056238745687
b = 2345672534

sp.gcd(a,b)
```

Out[5]: 1

In [6]:
```
mygcd(a,b)
```

Out[6]: 1

Q2

In [7]:
```
sp.gcdex(710,310)
```

Out[7]: (7, -16, 10)

In [8]:
```
7*710 -16*310
```

Out[8]: 10

The version of `mygcdex` below implements the requirements of tasks 1-4 of Q2.

In [9]:
```
def mygcdex(a,b):
    if b==0:
        return (1,0,a)
    else:
        # int div a = q*b + r,  0<= r < abs(b)
        q = sp.floor(a/b)

        if b > 0:
            r = a - q*b
        elif b < 0:
            r = a - (q+1)*b

        (x,y,d) = mygcdex(b,r)
        newx = y
        if b > 0:
            newy = x - q*y
        elif b < 0:
            newy = x - y*(q+1)
        return (newx,newy,d)
```

In [10]:
```
mygcdex(710,-310)
```

Out[10]: (7, 16, 10)

In [11]:
```
7*710 +16*(-310)
```

Out[11]: 10

In [12]:
```
a = 23478056238745687
b = 2345672534

t = sp.gcdex(a,b)
print(t)
```

```
(-1073503545, 10744797595009424, 1)
```

In [13]:
```
t[0]*a + t[1]*b == t[2]
```

Out[13]: True

In [14]:
```
a = 23478056238745687
b = 2345672534

t = mygcdex(a,b)
print(t)
```

```
(-1073503545, 10744797595009424, 1)
```

In [15]:
```
t[0]*a + t[1]*b == t[2]
```

Out[15]:
```
True
```

Q3 and Q4

In [16]:
```
sp.mod_inverse(5,8)
```

Out[16]:
```
5
```

In [17]:
```
sp.mod_inverse(5,11)
```

Out[17]:
```
9
```

In [18]:
```
def mymod_inverse(x,n):
    t = mygcdex(x,n)
    if t[2] != 1:
        raise ValueError('The element '+str(x)+' is not invertible modulo '+str(
    return sp.Mod(t[0],n)
```

In [19]:
```
mymod_inverse(5,8)
```

Out[19]:
```
5
```

In [20]:
```
mymod_inverse(5,11)
```

Out[20]:
```
9
```

In [21]:
```
mymod_inverse(2,8)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[21], line 1
----> 1 mymod_inverse(2,8)

Cell In[18], line 4, in mymod_inverse(x, n)
      2 t = mygcdex(x,n)
      3 if t[2] != 1:
----> 4     raise ValueError('The element '+str(x)+' is not invertible modulo '+s
tr(n))
      5 return sp.Mod(t[0],n)

ValueError: The element 2 is not invertible modulo 8
```

In [22]:
```
Z8 = [x for x in range(8) if mygcd(x,8)==1]
Z8
```

Out[22]:
```
[1, 3, 5, 7]
```

In [23]:
```
[mymod_inverse(x,8) for x in Z8]
```

Out[23]:
```
[1, 3, 5, 7]
```

In [24]:
```
Z11 = [x for x in range(11) if mygcd(x,11)==1]
```

```
Z11
```

Out[24]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [25]:
```
[(x,mymod_inverse(x,11)) for x in Z11]
```

Out[25]:
```
[(1, 1),
 (2, 6),
 (3, 4),
 (4, 3),
 (5, 9),
 (6, 2),
 (7, 8),
 (8, 7),
 (9, 5),
 (10, 10)]
```

In [26]:
```
sp.Mod(7*8,11)
```

Out[26]:
```
1
```

In [ ]:
```

```