Exercises: discrete logarithm based cryptography

Jean-Sébastien Coron

Université du Luxembourg

For the exercises below, install the Sage library, available at `http://www.sagemath.org/`. Alternatively, you can use the Sage Cell Server at `https://sagecell.sagemath.org`. Please submit a `.ipynb` file.

1 Key generation

Using the functions random_prime and is_prime, write a function that generates a prime p of size n bits, such that $q = (p-1)/2$ is prime. Write a function that given a prime p such that $q = (p-1)/2$ is prime, outputs a generator g of the prime order sub-group G of order q of $\mathbb{Z}_p^*$.

```
def genPrime(n=100):
  pass

def findGen(p):
  pass
```

2 El-Gamal encryption

We consider a variant of El-Gamal for a message $m \in \{0,1\}^k$, using a hash function $H : \{0,1\}^* \to \{0,1\}^k$, with the ciphertext:

$$c = (g^r, H(h^r) \oplus m)$$

We can use the SHA1 hash function, with $k = 160$:

```
import hashlib

def sha1(m):
  h=hashlib.sha1()
  h.update(m.encode("utf-8"))
  return h.hexdigest()
```

To convert an integer into a string, you can use the str function. To compute the xor between two strings, you can use the function xor_string below. To pad a string to 160 bits, you can use the pad_to_160_bits function below.

```
def xor_strings(s1, s2):
    return ''.join(chr(ord(a) ^^ ord(b)) for a, b in zip(s1, s2))

def pad_to_160_bits(s):
    return s.ljust(20, '\0')
```

Implement the El-Gamal encryption scheme, with key generation, encryption and decryption.

```
def ElGamalKeyGen(n=100):
    pass

def ElGamalEncrypt(m, p, g, y):
    pass

def ElGamalDecrypt(c1, c2, x):
    pass
```

Check that decryption works.

```
def testElgamal():
    p,g,y,x=ElGamalKeyGen()
    m="Hello"
    mpad=pad_to_160_bits(m)
    c1,c2=ElGamalEncrypt(mpad,p,g,y)
    m2=ElGamalDecrypt(c1,c2,x)
    assert m2==mpad
```