# Lab 5 – Homomorphic Encryption

## Objectives

In this lab we will be implementing some partially homomorphic encryption schemes and testing them out, including the ElGamal scheme, which is multiplicatively-homomorphic, and the Paillier scheme, which is additively-homomorphic.

## Exercises

1.1. Implement the ElGamal scheme. To implement this scheme, an incomplete implementation is given and you can add the missing functions. Some notes on the implementation:

- First we need to decide on the public parameters to use, including prime group descriptor **p**, prime order **q**, and generator **g**.
  - For testing purposes, small values can be used like **q**=83, **p**=167 and **g**=4;
  - For production purposes we should use the group described in https://www.rfc-editor.org/rfc/rfc5114#section-2.3 as we did for the Secret Sharing lab class.

- A PRNG that can output secure random numbers ranging from **1** to **q-1**;

- Given the public parameters, we need to calculate the public / private keys for the scheme. Check the lecture slides for how to generate these.

- A function encrypt that, given a message **m** (**0 < m < q**) and the public key, encrypts **m** and outputs its ciphertxt **c = (c1,c2)**. Refer to the slides for the algorithm.
  - Remember that all operations must be done module **p**.

- A function decrypt that, given a ciphertxt **c = (c1,c2)** and the private key, decrypts **c** and outputs its plaintext **m**. Refer to the slides for the algorithm.
  - Remember that all operations must be done module **p**.
  - To perform the division of BigIntegers, you might have to use the modInverse instead, i.e.: **c2 / c1 mod p = (c2 . c1^-1 mod p ) mod p**

- A function eval that, given two ciphertexts **cipher1** and **cipher2**, performs the homomorphic operation of this scheme.
  - i.e., it should return ciphertext **cipher3 = ( cipher1 . cipher2 ) mod p**

1.2. The incomplete implementation includes tests for testing the encryption, decryption and homomorphic operations. Execute them after completing your implementation and check the results.

2. Implement the Paillier scheme. To implement this scheme, an incomplete implementation is also given and you can add the missing functions. Some notes on the implementation:

   - We will need to decide on parameters **p** and **q**. These should be large prime numbers of 1024 or 2048 bits.

   - Similarly to ElGamal, we will need to generate private/public keys and implement functions for encryption, decryption and homomorphic evaluation.
     - Remember that all operations must be done module **n**^2 or module **n**. Check the lecture slides for mor information.

   - Test your scheme, as done for the ElGamal scheme.

## Additional Exercises

3. If you are interested in Order Revealing Encryption, there is an implementation in C available here: [https://github.com/kevinlewi/fastore](https://github.com/kevinlewi/fastore)