Starred exercises are optional.

The techniques required to solve exercises 1a, 1b, 2, and 4∗ were covered in the lecture of December 2nd. The other techniques will be covered in the lecture of December 7th, but are already available in the slides put online.

1) a) Compute $1998^{9198} \mod 9199$.
Hint: 9199 is prime.

   b) Compute the inverse of $64 \mod 991$, and give two n such that 64 does *not* have an inverse modulo n.

   c) Show how $11^{182} \mod 14$ can be computed efficiently by hand, by giving the computation. (It should not take more than 2 lines, without using a very small font or very wide page.)

2) Consider the following four ways to compute $123456789^{987654321} \mod 1678321$:

   1) Compute $123456789^{987654321}$ by repeated multiplication. Take the result modulo 1678321;
   2) Compute $123456789^{987654321}$ by fast exponentiation. Take the result modulo 1678321;
   3) As in the previous item, but computing intermediate results modulo 1678321;
   4) As 1678321 is prime, compute $123456789 \mod 1678321$ to 939356, $987654321 \mod 1678320$ to 802161, and then $939356^{802161} \mod 1678321$, as in the previous item.

   a) Can you verify the result is (the equivalence class represented by) 1064984 in each way? How do you justify the computations?
   Hint on experimentation: Haskell works well on large integers (but has its natural limits); `expmod` as on slide could be useful.

   b) Order these methods with respect to their complexity (for these numbers).

3) a) A child has fewer than 200 Lego bricks. If she puts the pieces in rows of 4 then 2 are left. If she tries rows of 5, then still 2 are left. If she tries rows of 7, 3 pieces are left. How many bricks does she have?

   b) How can the Chinese Remainder Theorem be used to speed up RSA? Describe the more efficient version of these steps using CRT.

4*) Take an attackers point of view. You intercept the ciphertext c and you know it has been encrypted by means of RSA. You also have the public key (e, n). How would you proceed to decrypt the message? Give a naïve (in the sense, that you don't need to care about complexity) implementation in Haskell. (If you use parts of code from external sources indicate your references. In any case comment your code and explain what you do.)