# TP 4: the RSA algorithm

Jean-Sébastien Coron and David Galindo

Université du Luxembourg

## 1 RSA with artificially small parameters

The goal is to implement the RSA algorithm, but only with artificially small parameters for simplicity.

For key-generation, define a function:

```
void keygen(int *p,int *q, int *e, int *d,int length)
```

that generates two random primes $p$ and $q$ of size `length` bit (in this implementation, one can take `length`=30), and that also generates the pair $(e, d)$. A random prime is generated by repeatedly generating a random integer and then testing for primality, by trial division.

Implement the function:

```
int RSAencrypt(int m,int e,int n)
```

that takes as input a message $m$, a public exponent $e$ and a RSA modulus $n$ and outputs the corresponding ciphertext $c$. Use the square-and-multiply algorithm.

Similarly, implement the function:

```
int RSAdecrypt(int c,int d,int n)
```

that decrypts a ciphertext $c$. Check that decryption works.

In practice, RSA must be used with must larger parameters, typically the RSA modulus must be at least 1024 bits long. One must then use an efficient algorithm for prime number generation; this will be covered in the next courses.

## 2 Håstad's attack for related messages

Remember that the plain RSA public key encryption key is $(n, e)$ where $n = pq$ is the product of two primer numbers and $e$ and $(p-1)(q-1)$ are coprimes. Namely, $\text{RSA}(m) = m^e \mod n$. Assume that the public keys of Alice, Bob and Charlie are $(n_A, 3), (n_B, 3)$ and $(n_C, 3)$ respectively. Suppose that Daniel wants to send the *same* message $m$ to Alice, Bob and Charlie. That is, he sends over $y_A = m^3 \mod n_A$, $y_B = m^3 \mod n_B$ and $y_C = m^3 \mod n_C$.

1. Show that it is possible to retrieve $m$ from $y_A, y_B, y_C$ and without using the corresponding secret keys.
2. Implement the above attack by using your `keygen` and `RSAencrypt` functions.