TP 5: the RSA algorithm and Primality Testing

Jean-Sébastien Coron

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.

2 Modular exponentiation

Implement the square and multiply algorithm with large integers.

3 Fermat test

Implement the Fermat test of primality with small integers, and then with large integers.

4 Miller-Rabin

Implement the Miller-Rabin test of primality with small integers, and then with large integers.