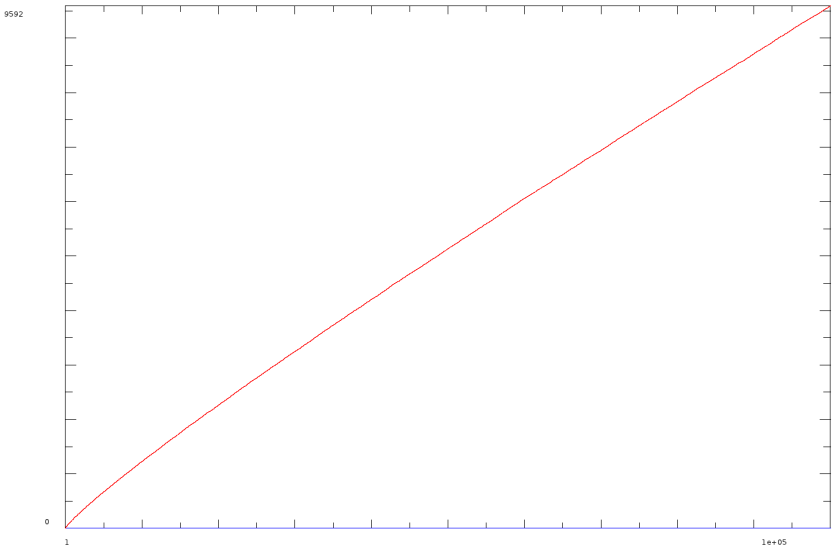
TP2, cryptographie

1. Préliminaires

Exercice 1.1. Écrire une fonction `euclide(a, b)` renvoyant le pgcd des entiers `a` et `b`.

Exercice 1.2. Écrire une fonction `bezout(a, b)` calculant une décomposition de Bézout $au + bv = \text{pgcd}(a, b)$ par l'algorithme d'Euclide étendu et changeant u si nécessaire pour que $0 \leq u < b$ (on montrera au préalable qu'il existe une telle décomposition et qu'elle est unique).

Exercice 1.3. Écrire une fonction `eratosthene(n)` renvoyant la liste de tous les nombres premiers $\leq n$ (cf. le manuel de Pari pour une description du travail avec les listes).
Facultatif : On tracera ensuite, à l'aide de la liste obtenue, le graphe de la fonction $x \mapsto \text{card}\,\{p \leq x, p \text{ premier}\}$ *qui devrait ressembler à*



Exercice 1.4 (Facultatif)**.** Construire une fonction `factorisation(n)` qui calcule une décomposition en facteur premier d'un entier n (on se servira pour cela de la fonction `eratosthene`).

2. Puissance

Nous aurons besoin dans ce TP de calculer beaucoup de puissances, et il convient donc d'utiliser un algorithme efficace ! Pour cela, nous allons comparer l'algorithme vu en cours au calcul naïf.

i) Écrire une fonction `pw` prenant en paramètres des entiers p, n, ℓ et calculant n^ℓ dans $\mathbb{Z}/p\mathbb{Z}$ à l'aide de l'algorithme décrit dans le cours.

ii) Écrire une fonction `pwn` calculant de manière naïve n^ℓ et prenant ensuite le reste de la division par p.

iii) Comparer les temps de calculs des fonctions `pw`, `pwn` et de l'opérateur `**` de `Python` à l'aide de `time.time()`.

Tous les calculs de puissance effectué dans la suite de ce TP le seront avec l'algorithme le plus rapide des trois cités ci-dessus.

3. RSA

Le but est de construire un ensemble de fonctions qui permettent l'échange et la signature de messages par l'algorithme RSA.

3.1. **Envoi de message.**

i) À l'aide de la fonction `random.uniform` de `numpy` et de la fonction `eratosthene`, créer une fonction `gen_p` renvoyant deux nombres premiers p et q distincts, aléatoires et de plus de 6 chiffres (on commencera bien sûr par tester la fonction avec des entiers plus petits !).

ii) Construire une fonction `clef_rsa` construisant, à partir des entiers renvoyés par `gen_p`, les clefs publique et privée pour l'algorithme RSA. On pourra choisir l'entier e de manière aléatoire (premier à $\varphi(n)$) et calculer d à l'aide de la fonction `bezout`.

iii) Construire une fonction `code_rsa` prenant en paramètres la clef publique et un message m, et qui code ce message.

iv) Construire une fonction `decode_rsa` prenant en paramètres la clef privée et un message M, et qui décode ce message.

v) Tester les fonctions construites sur des exemples.

3.2. **Signature.** En vous inspirant de la partie sur le codage de messages, écrire un ensemble de fonctions permettant la signature de messages par l'algorithme RSA.

4. El Gamal

Le but est de construire en ensemble de routines permettant le codage et le décodage de message avec l'algorithme El Gamal.

4.1. **Puissance et ordre.** Écrire une fonction `is_gen` prenant en paramètres un entier p premier et un entier g premier à p, et testant si g est un gérateur de $(\mathbb{Z}/p\mathbb{Z})^*$. On pourra pour cela écrire une fonction `factor` utilisant `eratosthene` afin de calculer tous les facteurs premiers de $p-1$.

4.2. **Codage et décodage.**

i) À l'aide de la fonction `random.uniform` de `Python` et de la fonction `eratosthene`, créer une fonction `gen_p` renvoyant un nombre premier p aléatoires et de plus de 6 chiffres.

ii) Construire une fonction `generateur` prenant en paramètre un nombre premier p et cherchant de manière aléatoire un entier g générateur de $(\mathbb{Z}/p\mathbb{Z})^*$ (cf. la fonction `is_gen`).

iii) Construire une fonction `clef_elgamal` construisant, à partir de `gen_p` et `generateur` les clefs publique et privée pour l'algorithme El Gamal.

iv) Construire une fonction `code_elgamal` prenant en paramètres la clef publique et un message m, et qui code ce message.

v) Construire une fonction `decode_elgamal` prenant en paramètres la clef privée et un message M, et qui décode ce message.

vi) Tester les fonctions construites sur des exemples.