Introduction à la cryptographie - TP 2

1 Primalité

Comme nous l'avons vu lors du précédent TP, générer des clefs RSA suppose pouvoir générer des (grands) nombres premiers. Tester la **primalité** d'un nombre (i.e s'assurer qu'un nombre donné est premier) a longtemps été perçu comme un problème difficile. En 2003, trois mathématiciens indiens démontraient que cette labeur millénaire était bien plus « simple » que la communauté ne le pensait auparavant (on dit que ce problème appartient à la classe **P**). Malgré cette avancée considérable, la plupart des méthodes aujourd'hui employées repose sur des algorithmes probabilistes, qui, s'ils ne garantissent pas l'exactitude de la réponse, ont le mérite d'être bien plus rapides.

1.1 Crible d'Ératosthène

La crible d'Ératosthène est un algorithme exact. Il permet de trouver tous les nombres premiers inférieurs à une borne supérieure. Imaginons souhaiter lister tous les nombres premiers inférieurs à 20. Nous allons écrire tous les nombres compris entre 1 et 20 (inclus) dans un tableau, comme montré ici :

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Nous allons entourer, dans ce tableau, tous les multiples de 2 (4, 8, etc) :

1	2	3	④	5
⑥	7	⑧	9	⑩
11	⑫	13	⑭	15
⑯	17	⑱	19	⑳

Nous lisons ensuite le tableau, ligne par ligne, à la recherche d'un nombre non entouré et non déjà testé (hormis 1 qui est un cas spécial). Nous tombons sur 3. Il faut donc maintenant entourer tous les multiples de 3.

1	2	3	④	5
⑥	7	⑧	⑨	⑩
11	⑫	13	⑭	⑮
⑯	17	⑱	19	⑳

La procédure achevée, nous recherchons un nouveau nombre, ici 5. On voit qu'il n'est plus possible d'entourer des nouveaux nombres même en poursuivant la procédure. Nous avons dès lors obtenu la liste des nombres premiers inférieurs à 20 : il s'agit des nombres qui n'ont pas été entourés dans le tableau précédent.

Q1. Écrire un programme listant les nombres premiers inférieurs à une borne donnée en s'inspirant de la crible d'Ératosthène.

Une autre façon de tester la primalité de n est de naïvement tester la divisibilité de ce dernier par tous les entiers qui lui sont inférieurs. En fait, on peut être plus « malin » et remarquer que nous n'avons pas besoin de tester l'ensemble des entiers de 2 à $n-1$ mais seulement de 2 à $\lfloor\sqrt{n}\rfloor$.

Q2. Écrire un programme indiquant la primalité **(bool)** d'un nombre donné en argument en utilisant cet algorithme naïf.

1.2 Test de Fermat

Comme nous l'avons déjà vérifié dans le TP précédent, le petit théorème de Fermat stipule que si p est un nombre premier, alors pour tout $a \in \mathbb{Z}$

$$a^{p-1} \equiv 1 \bmod p$$

Par exemple, $2^4 \equiv 1 \bmod 5$ de même que $3^4 \equiv 1 \bmod 5$ etc. Nous pouvons alors aisément vérifier que si $a^{n-1} \neq 1 \bmod n$ alors n n'est pas premier. Illustrons cela par $2^9 \equiv 2 \bmod 10$. Le nombre 10 n'est donc pas premier.
Cela dit, il existe des nombres non premiers C (Nombres de **Carmichael**, 561 en est un) tels que pour tout $a \in \mathbb{Z}$

$$a^{C-1} \equiv 1 \bmod C$$

Ces nombres, dont on sait depuis 1994 qu'il en existe une infinité, sont un véritable obstacle au test de Fermat. S'ils n'existaient pas, Fermat pourrait fournir une réponse à la question « n est-il premier ? » avec une probabilité de se "tromper" aussi petite que l'on veut.
Faisons pour le moment abstraction de ces nombres C, considérons un algorithme qui teste

$$a^{p-1} \equiv 1 \bmod p$$

sur un nombre arbitraire de a. Plus le nombre de tests est grand, plus la probabilité de se « tromper » est faible.

Q3. Coder un tel algorithme.

1.3 Test de Miller-Rabin

Le test de **Miller-Rabin** est, tout comme le test de Fermat, un test probabiliste. Il se base sur l'observation suivante (p premier) :

$$x^2 \equiv 1 \bmod p$$

est équivalent à

$$(x-1) \times (x+1) \equiv 0 \bmod p$$

Autrement dit, les racines carrées de 1 modulo p sont 1 et -1.
Si n est premier, alors $n-1$ peut s'écrire comme $2^s \times d \ldots$
L'une des propositions suivantes est toujours vraie (pour un $r < s$) :

$$x^d \equiv 1 \bmod p \tag{1}$$

ou

$$x^{2^s d} \equiv -1 \bmod p \tag{2}$$

L'algorithme de **Miller-Rabin** se base sur la contraposée des égalités précédentes.

$$x^d \neq 1 \bmod n \tag{3}$$

et

$$x^{2^r d} \neq -1 \bmod n \tag{4}$$

Cela signifie que si nous trouvons un nombre x tel que les inégalités précédentes sont vraies pour tous les $r < s$ alors n est **probablement** premier.

Q4. Coder un tel algorithme.