TP7 : RSA et factorisation d'entiers

Références :
— *Calcul mathématique avec SAGE*, multi-auteurs.
— *Algorithmique*, T. Cormen, C. Leiserson, R. Rivest [1], C. Stein.
— *A course in computational algebraic number theory*, H. Cohen.

Ce TP aborde la question de la protection de la confidentialité d'un message. Nous détaillerons l'exemple du cryptosystème RSA (Rivest, Shamir, Adleman, 1978), l'un des premiers à clé asymétrique, et qui est aujourd'hui encore largement utilisé lors des transactions en ligne. Il repose sur l'exponentiation modulaire ; et sa sécurité sur la difficulté de factoriser un entier.

1 Le protocole RSA

1.1 Clé publique, clé privée

La Terre du Milieu n'est plus sûre. Une menace grandit à l'Est, et son ombre se répand par delà les Monts Brumeux. A Imladris, Elrond tente de fédérer les peuples libres. Mais les routes sont surveillées, et des espions se glissent partout. Dans sa grande sagesse, Elrond choisit de protéger les messages qui lui sont envoyés grâce au cryptosystème RSA, dont voici le principe.

Génération des clés :

1. Elrond choisit deux entiers premiers, p et q, qu'il garde secrets. Sur un parchemin, il calcule leur produit $n = pq$. Les messages qui lui seront transmis seront des éléments de $\mathbb{Z}/n\mathbb{Z}$.
2. Il fabrique ensuite sa clé publique e, en choisissant au hasard un élément inversible de $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$ (et en prenant soin d'éviter 1 et -1).
3. Sur son parchemin, Elrond calcule sa clé privée d, telle que :
$de = 1 \mod (p-1)(q-1)$.
4. La clé publique d'Elrond (n, e) est envoyée partout en Terre du Milieu. Elrond conserve secrètement d et brûle son parchemin de calcul.

Remarque : selon l'auteur, la *clé publique* [respectivement *privée*] désigne ou bien le couple(n, e) [resp. (n, d)], ou bien la seule donnée de e [resp. d]. Dans les deux cas, ce sont toujours des éléments de $\mathbb{Z}/n\mathbb{Z}$ que l'on monte en puissance lors du cryptage et du décryptage. En particulier, les messages que l'on souhaite transmettre doivent au préalable être codés par un ou plusieurs élément(s) de cet anneau.

1. Pourquoi faut-il éviter $e = 1$ et $e = -1$?
2. Générer aléatoirement un jeu de clés (n, e, d), où les nombres premiers p et q sont tirés au hasard entre 100 et 1000.

1.2 Chiffrer et déchiffrer les messages

Protocoles de chiffrement et de déchiffrement :

— L'expéditeur chiffre son message $m \in \mathbb{Z}/n\mathbb{Z}$ par $E(m) = m^e$.
— Elrond déchiffre le message reçu $m' \in \mathbb{Z}/n\mathbb{Z}$ en posant $D(m') = (m')^d$.

1. qui a donné son "R" à RSA.

3. Montrer que pour tout $m \in \mathbb{Z}$, $m^{de} = m \bmod n$. Pourquoi a-t-on intérêt à ce que m soit premier avec n ?
4. Rappeler quel est le bon algorithme pour calculer la puissance d'un nombre ? Quelle est sa complexité ?
5. Soit x un entier entre 0 et $n-1$. Vaut-il mieux calculer x^e puis projeter dans $\mathbb{Z}/n\mathbb{Z}$, ou projeter d'abord x dans $\mathbb{Z}/n\mathbb{Z}$ avant d'en calculer la puissance ?
6. Ecrire deux fonctions `crypter(m,e)` et `decrypter(m,d)`, où m est une liste d'entiers modulo n. La première doit renvoyer la liste des nombres cryptés avec la clé publique e, la seconde, celle des nombres décryptés grâce à la clé d. Crypter puis décrypter le message $[67, 42]$.

Mais ce n'est pas l'anneau - $[\mathbb{Z}/n\mathbb{Z}]$ - qu'Elrond veut protéger, mais les messages que ses alliés pourraient lui écrire en Langage Commun [2]. Aussi, nous allons voir comment coder un texte par une suite d'entiers modulo n, en utilisant *les chaînes de caractères.* Vous pouvez commencer par lire le court paragraphe qui leur est consacré dans *Calcul mathématique avec SAGE.*

```
♣  m1='hell',  m2='o world!',  m = m1+m2,  m[3],  len(m) ♣
```

7. En désignant chaque lettre (minuscule, sans accent) par sa position dans l'alphabet, et le caractère *espace* par 0, écrire une fonction `traduire_num(t)`, qui a une chaîne de caractères t associe une liste L de même longueur, telle que son *i-ème* élément soit la position dans l'alphabet de la *i-ème* lettre de t. Ecrire aussi `traduire_texte(L)` qui effectue l'opération réciproque.
   Exemple : `'ceci est un exemple'` donne
   `[3,5,3,1,0,5,19,20,0,21,14,0,5,24,5,13,16,12,5]`.

Nous pouvons à présent coder ce message par l'entier N dont l'écriture en base 27 coïncide avec sa traduction numérique : $N = 3 + 5 * 27 + 3 * 27^2 + ... + 5 * 27^{18}$. Le problème est que ce nombre, si le message est trop long, pourrait dépasser n - rendant la projection sur $\mathbb{Z}/n\mathbb{Z}$ non injective et le message non décodable. C'est pourquoi nous allons d'abord découper le message en blocs plus petits, de longueur déterminée l, puis coder chacun de ces blocs par un élément de l'anneau $\mathbb{Z}/n\mathbb{Z}$.

8. Exprimer, en fonction de n, la plus grande longueur l de bloc possible tel que l'opération de codage reste injective. Pourquoi ne pas choisir l petit, par exemple $l = 1$?
9. Ecrire une fonction `regrouper_blocs_base27(L1,l)` qui, à une liste L_1 d'entiers inférieurs à 26, associe la liste d'entiers L_2 dont chaque élément code un bloc de L_1 de longueur l :

$$L_2[0] = L_1[0] + L_1[1] * 27 + ... + L_1[l-1] * 27^{l-1}$$

$$L_2[1] = L_1[l] + L_1[l+1] * 27 + ... + L_1[2l-1] * 27^{l-1}$$

$$...$$

   Ecrire aussi la fonction réciproque `degrouper_blocs_base27(L2,l)`. Assurez-vous qu'avec la valeur de l calculée en question 8, les entiers obtenus sont inférieurs à n.
10. Ecrire un algorithme `chiffrerRSA(m,n,e)` qui chiffre la chaîne de caractères m avec la clé publique (n, e). Vous devez obtenir en sortie une suite d'entiers inférieurs à n. Ecrire réciproquement la fonction `dechiffrerRSA(mc,n,d)`. Déchiffrer le message suivant avec la clé privée $(8680363, 6344439)$:

$$[1071173, 7618127, 5750044, 6682199, 7201675, 1205570, 3550870, 1446920,$$

$$6547720, 6867059, 1834532, 1954921, 1539949, 6534398, 6034404, 1053908, 2317075]$$

2. alphabet latin

2 Attaque par force brute et factorisation d'entiers

La sécurité d'un système cryptographique repose sur une dissymétrie entre le temps de cryptage et décryptage, lorsque l'on possède la clé, et le temps minimal requis pour "forcer" le code.

11. Expliquer comment *deviner* la clé privée lorsque l'on connaît la clé publique.
12. A l'aide de `factor`, casser le code suivant dont la clé publique est $(71968847, 9122909)$:

    $[44865280, 34012313, 46358453, 49442412, 69489959, 28150840, 35514701, 59077479, 10175926]$

2.1 Coût de l'algorithme de factorisation naïf

13. Ecrire une fonction `facto_naive(n)` qui dresse la liste des facteurs premiers de n en testant tous les diviseurs impairs inférieurs ou égaux à $\sqrt{n}$. Illustrer en affichant les factorisations des nombres de 1 à 20. Votre fonction vous permet-elle de casser la clé publique précédente ? Jusqu'à quelle taille d'entiers le temps d'exécution maximal est-il inférieur à une seconde ?
14. Expliciter la complexité de l'algorithme.
15. Sachant que les nombres effectivement utilisées pour RSA aujourd'hui font plus de 1024 bits, estimer le temps nécessaire à 10 milliards de processeurs à 10GHz, travaillant en parallèle, pour casser une clé.
16. Rappeler l'ordre de grandeur des nombres premiers (ou plutôt *pseudo-premiers*) que vous parvenez à fabriquer à partir du test de Fermat ou Solovay-Strassen ; et estimer le temps nécessaire à la factorisation d'un produit n de deux tels nombres.

2.2 Aperçu d'un meilleur algorithme : méthode du ρ de Pollard

La suite du TP n'est pas au programme, mais plutôt, dans l'esprit de l'épreuve d'option : l'algorithme n'est ni expliqué, ni même justifié, les questions ne sont là que pour vous donner des pistes d'exploration : il vous revient de fouiller et illustrer un sujet dont on ne vous donne pas toutes les clés.

La méthode du ρ de Pollard est une *heuristique* : son succès n'est pas garanti (l'algorithme peut terminer sur la factorisation triviale $1 * n$), non plus que son temps d'éxécution ; elle marche pourtant très bien en pratique.

En quelques mots, elle repose sur le fait que la suite récurrente $(x_i)_{i\in\mathbb{N}}$, où x_0 est un élément quelconque de $\mathbb{Z}/n\mathbb{Z}$, et $x_{i+1} = x_i^2 + 1$, *ressemble* à une suite d'élements de $\mathbb{Z}/n\mathbb{Z}$ choisis aléatoirement [3] ; sur le principe des tiroirs et le paradoxe des anniversaires.

17. (*Ergodicité*) Ecrire une fonction `visu_suite(n,N)` qui tire un élément au hasard dans $\mathbb{Z}/n\mathbb{Z}$, calcule les N premiers termes de la suite récurrente définie ci-dessus, puis les trace dans une fenêtre graphique. L'exécuter plusieurs fois pour $(n, N) = (233620813, 80)$, et d'autres valeurs $n = pq$ de votre choix. Qu'observez-vous ?
18. Prouver que la suite $(x_i)_{i\in\mathbb{N}}$ est en fait périodique à partir d'un certain rang.
19. (*Paradoxe des anniversaires.*) On choisit aléatoirement avec remise parmi n boules. Ecrire une fonction `anniversaire(n)` qui calcule le nombre de tirages nécessaires pour obtenir deux fois la même boule. Tracer la moyenne empirique de cette variable aléatoire en fonction de n. Montrer que son espérance varie en $O(\sqrt{n})$.
20. Implémenter et tester l'algorithme ci-dessous :

3. L'idée cachée est celle d'*ergodicité* d'un système dynamique discret.

```
def rho_pollard(n,cpt_max) :    # n produit de deux premiers distincts
    x = randint(0,n-1)
    y = x
    cpt = 0
    g = 1
    while (g==1) and (cpt<cpt_max) :
        cpt += 1
        x = (x^2 + 1)%n
        y = ((y^2+1)^2+1)%n
        g = gcd(x-y,n)
    if g==1 :
        print 'Nombre maximal de boucles autorise atteint'
    return g, n//g
```

Comparer ses performances avec `facto_naive(n)`. Jusqu'à quelle taille d'entiers pouvez-vous factoriser en moins d'une seconde ?

21. Casser le cryptogramme suivant, dont la clé publique est :

$$n = 372587796410890363404360348635537066120429257549885886119524 7$$

$$e = 150878848222754521175887439462443920378467168462528236101966 9$$

cryptogramme :

[312332839732884775851668080200224559907195149148260808908203,

366437985843701138777941945614186095694453346940336567062491 9,

229466290590827210178880033592517658650969764685448959743346 9]

Morale

Dans *Algorithmique*, R. Rivest écrit :

> La sécurité du cryptosystème RSA repose en grande partie sur la difficulté de factoriser les grands entiers. Si une personne mal intentionnée parvient à factoriser le module n d'une clé publique, elle peut en déduire la clé secrète [...]. Donc, si la factorisation des grands entiers est facile, il est facile de casser le cryptosystème RSA. La proposition inverse "la complexité de la factorisation des grands entiers implique-t-elle la casse de RSA ?" n'a pas été prouvée. Toutefois, après deux décennies de recherches, personne n'a trouvé de moyens plus simple pour casser le système RSA que de factoriser le module n. [...] A moins d'une percée fondamentale dans la conception d'algorithmes de théorie des nombres, le cryptosystème RSA fournit un haut degré de sécurité aux applications, pour peu qu'on l'implémente avec soin en suivant des normes recommandées.

22. A quelles normes pouvez-vous penser, et quelles types d'attaques pourraient-elles permettre d'éviter ?