Numérique et Sciences Informatiques
Chapitre XI - Sécurisation des communications
Travaux Dirigés 21

Classe de terminale

Première partie

Chiffrement asymétrique

On rappelle qu'un chiffrement asymétrique nécessite deux clefs :

- une clef publique qui permet de chiffrer un message
- une clef privée qui permet de déchiffrer un message

Nous avons vu précédemment (et même en classe de première) qu'un caractère peut-être représenté par un nombre entier. C'est pourquoi, nous allons chiffrer des nombres dans un premier temps puis des chaînes de caractères ensuite, dans chacun des deux exemples ci-dessous.

I. Le chiffrement RSA

Le chiffrement RSA a été inventé en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman. Il s'agit d'un chiffrement asymétrique qui utilise une clé publique (des nombres entiers) pour chiffrer et une clé privée (des nombres entiers) pour déchiffrer.

Étapes de création des clés

- Choisir deux nombres entiers premiers p et q distincts.
- Calculer $n = pq$.
- Calculer $\phi(n) = (p-1)(q-1)$. Ce nombre $\phi(n)$ s'appelle l'indicatrice d'Euler.
- Choisir un entier naturel $e > 1$ premier avec $\phi(n)$ et strictement inférieur à $\phi(n)$.
- Calculer l'entier naturel d, inverse de e modulo $\phi(n)$, c'est-à-dire l'entier naturel d tel que $ed \equiv 1 \pmod{\phi(n)}$.
- La clé publique est le couple (n, e) et la clé privée est le couple (n, d).

1. On rappelle que le PGCD de deux entiers peut-être défini par :

   - Pour tout entier naturel b, $PGCD(0, b) = b$
   - Pour tous entiers naturels a et b tels que $a < b$, $PGCD(a, b) = PGCD(a, b - a)$
   - Pour tous entiers naturels a et b, $PGCD(a, b) = PGCD(b, a)$

   Implémenter une fonction récursive `pgcd(a,b)` de paramètre `a` et `b` deux nombres entiers naturels, et qui renvoie le plus grand commun diviseur de `a` et `b`.

2. Implémenter une fonction `creation_cles(p,q)` qui prend en paramètres deux entiers premiers `p` et `q` et qui renvoie la clé publique et la clé privée, toutes les deux étant des p-uplets.
   Indication : deux entiers naturels p et qsont premiers entre eux si et seulement si $PGCD(p, q) = 1$
   On pourra utiliser le jeu de test suivant :

```
>>> creation_cles(3,11)
((33, 3), (33, 7))
>>> creation_cles(2,13)
((26, 5), (26, 5))
>>> creation_cles(5,31)
((155, 7), (155, 103))
```

3. Implémenter une fonction `chiffrement_RSA(cle_publique, nombre)` ayant deux paramètres : `cle_publique` (un p-uplet d'entiers de type (n, e)) et `nombre` (un entier inférieur à n).
Cette fonction renvoie un entier correspondant au nombre chiffré.
On pourra utiliser le jeu de test suivant :

```
>>> chiffrement((33, 3),5)
26
>>> chiffrement((26,5), 7)
11
>>> chiffrement((155, 7),8)
2
```

4. Implémenter une fonction `dechiffrement_RSA(cle_privee, nombre)` ayant deux paramètres : `cle_privee` (un p-uplet de type (n, e)) et `nombre` (un entier inférieur à n).
Cette fonction renvoie le nombre déchiffré. *On pourra utiliser le jeu de test suivant :*

```
>>> dechiffrement((33, 7),26)
5
>>> dechiffrement((26,5), 11)
7
>>> dechiffrement((155, 103),2)
8
```

5. Implémenter une fonction `chiffrement_texte_RSA(message_clair, cle_publique)`, qui prend en paramètres une chaîne de caractères `message_clair` et une clé publique `cle_publique` (un p-uplet de type (n, e)). Cette fonction renvoie une chaîne de caractères correspondant au message chiffré avec le chiffrement RSA.
On pourra tester cette fonction avec :

```
>>> chiffrement_texte_RSA("Bonjour", (155, 7))
'\x10Gi\x92G\x03h'
```

6. Implémenter une fonction `dechiffrement_texte_RSA(message_chiffre, cle_privee)`, qui prend en paramètres une chaîne de caractères `message_chiffre` correspondant à un message chiffré avec le chiffrement RSA et une clé privée `cle_privee` (un p-uplet de type (n, d)). Cette fonction renvoie une chaîne de caractères correspondant au message originel chiffré avec le chiffrement RSA.
On pourra tester cette fonction avec :

```
>>> dechiffrement_texte_RSA('\x10Gi\x92G\x03h', (155, 103))
'Bonjour'
```

II. Le chiffrement Merkle-Hellman

Le chiffrement Merkle-Hellman a besoin :

- d'une clé privée qui est un triplet composé de deux nombres et d'un tableau de nombres croissants choisis par l'utilisateur.
- d'une clé publique qui est est une liste ordonnée de nombres calculés à partir de la clé privée.

1. Génération des clés

   (a) Implémenter une fonction `sac_supercroissant(nb)`, qui prend en paramètre un entier naturel `nb` et qui renvoie un tableau de nombres entiers naturels aléatoires compris entre 1 et 1000, triés dans l'ordre croissant.

   (b) Implémenter uen fonction `generation_clefs(nb)`, qui prend en paramètre un entier naturel `nb` et qui renvoie un couple de clefs :

   - La clef privée est un triplet composé d'un entier N, d'un entier A et d'un tableau T composé de nb nombres triés dans l'ordre croissant tels que :
     → N est supérieur (strictement) à la somme des éléments du tableau T.
     → $A < N$ et $PGCD(A, N) = 1$
   - La clef publique est un tableau P tel que :
     → P et T ont le même nombre d'éléments.
     → Pour chaque indice i, $P[i]$ est égal au reste de la division euclidienne de $A \times T[i]$ par N.

   On pourra tester cette fonction avec :

```
>>> generation_clef(8)
((24120, 10171, [28, 155, 292, 360, 636, 689, 880, 980]), [19468, 8705, 3172, 19440, 4596,
13019, 1960, 6020])
```

2. Chiffrement
   Implémenter une fonction `chiffrement_MH(message_clair, clef_publique)`, qui prend en paramètres une chaîne de caractères `message_clair` et une clef publique `clef_publique`. Cette fonction renvoie une chaîne de caractères correspondant au message chiffré avec le chiffrement Merkle-Hellman.
   On pourra tester cette fonction avec :

```
>>> chiffrement_MH("A", [19468, 8705, 3172, 19440, 4596, 13019, 1960, 6020])
14725
```

3. Déchiffrement
   Implémenter une fonction `dechiffrement_MH(messagee_chiffre, clef_privee)`, qui prend en paramètres une chaîne de caractères `message_chiffre` correspondant à un message chiffré avec le chiffrement Merkle-Hellman et une clef privée `clef_privee`. Cette fonction renvoie une chaîne de caractères correspondant au message originel chiffré avec le chiffrement Merkle-Hellman.
   On pourra tester cette fonction avec :

```
dechiffrement_MH(14725, (24120, 10171, [28, 155, 292, 360, 636, 689, 880, 980]))
'A'
```