TP2 – Cryptographie symétrique

Exercice 1. *GnuPG*

On s'intéresse à quelques fonctionnalités de cryptographie asymétrique de GnuPG (« GNU Privacy Guard »). Le sujet donne les principales commandes à utiliser. Les « `...` » sont à compléter, en utilisant `man gpg` et `gpg -h` pour avoir du détail sur l'utilisation des commandes.

1. **i.** Générer des clefs avec la commande `gpg --gen-key`.
   ii. Générer un certificat de révocation pour votre clef, avec `gpg --gen-revoke ...`.
   iii. Exporter votre clef publique avec `gpg --armor --export ...`.
2. **i.** Échanger entre vous des clefs publiques par email pour la suite.
   ii. Pour la suite également, créer dans votre dossier quelques fichiers texte avec du contenu quelconque.
3. **i.** Signer un fichier pour un destinataire, avec la commande `gpg --detach-sign ...`.
   ii. Échanger des fichiers et signatures entre vous.
   iii. Vérifier la signature à la réception, avec `gpg --verify ...`
4. **i.** Chiffrer un fichier pour un destinataire, avec la commande `gpg --encrypt ...`.
   ii. Échanger des fichiers chiffrés entre vous.
   iii. Déchiffrer à la réception, avec `gpg --decrypt ...`.
5. Combiner signature et chiffrement : répéter **i.** à **iii.** des questions précédentes, avec des fichiers signés et chiffrés. On peut combiner les options `--encrypt` et `--detach-sign` de `gpg`.

Exercice 2. *Implantation du cryptosystème El Gamal*

L'objectif est d'implanter le cryptosystème El Gamal dans un *sous-groupe premier* G de $(\mathbb{Z}/p\mathbb{Z})^\times$ où p est un premier. Pour cela, on choisit p de la forme $2q+1$ où q est également premier. Alors la théorie des groupes permet d'affirmer que tout élément de $(\mathbb{Z}/p\mathbb{Z})^\times$ est d'ordre[1] 1, 2 ou q. Le groupe G est l'ensemble des éléments d'ordre 1 ou q de $(\mathbb{Z}/p\mathbb{Z})^\times$, qui est un groupe d'ordre[1] q. Tout élément $\neq 1$ de ce groupe en est un générateur.

Pour l'implantation, on se base sur la bibliothèque `sympy`, qui fournit (entre autres) les deux fonctions suivantes : `randprime(a,b)` renvoie un premier compris entre a et b, et `isprime` teste si un entier est premier. On utilise la fonction `randrange` de la bibliothèque `random`.

Dans la suite, on demande d'écrire des fonctions. Vous pouvez intégrer ces fonctions dans une classe `ElGama1`. La partie génie logiciel est laissée à votre appréciation.

1. **i.** Écrire une fonction `premier(n)` qui tire aléatoirement un premier p de n bits, de la forme $2q+1$ où q est un nombre premier. *Le principe est de tirer q aléatoirement et tester si $2q+1$ est premier, et recommencer sinon.*
   ii. Écrire une fonction `generateur(p)` qui trouve un générateur du sous-groupe d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^\times$. *Il suffit de tirer aléatoirement des éléments g de $\mathbb{Z}/p\mathbb{Z}$ jusqu'à en trouver un qui soit d'ordre q, c'est-à-dire tel que $g^2 \bmod p \neq 1$.*
2. Écrire la fonction `Gen(n)` de génération des clefs, qui effectue les tâches suivantes :
   - Choix d'un nombre premier p de n bits de la forme $2q+1$, aléatoirement ;
   - Choix d'un générateur g du sous-groupe d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^\times$;
   - Choix d'une clef secrète aléatoire $x \in \{0, \dots, q-1\}$;
   - Calculer $h = g^x$ dans $\mathbb{Z}/p\mathbb{Z}$ (*attention au temps de calcul !*) ;
   - La clef secrète est (p, x) et la clef publique est (p, g, h).
3. Écrire la fonction `Encrypt(pk, m)` qui prend en entrée la clef publique et un élément de $\mathbb{Z}/p\mathbb{Z}$ et chiffre cet élément avec la clef publique pk.
4. Écrire la fonction `Decrypt(sk, c)` qui prend en entrée la clef secrète et un chiffré et renvoie le message $m \in \mathbb{Z}/p\mathbb{Z}$ correspondant.
5. Générer des clefs de 1024 bits, chiffrer des messages et vérifier qu'ils sont correctement déchiffrés. *Le temps de génération de la clef est de l'ordre de la dizaine de secondes.*

1. Rappel, l'ordre d'un élément x est le plus petit exposant e tel que $x^e = 1$, et l'ordre d'un groupe est son nombre d'éléments.

Exercice 3. *Attaque de Wiener*

On considère le cryptosystème RSA simple. On veut implanter une attaque qui fonctionne lorsque l'exposant d de déchiffrement est petit. Cette attaque se base sur les *fractions continues*. Pour des entiers $q_0, \dots, q_m$, on note

$$[q_0, q_1, \dots, q_m] = q_0 + \cfrac{1}{q_1 + \cfrac{1}{\ddots + \cfrac{1}{q_{m-1} + \cfrac{1}{q_m}}}}$$

Étant donné deux entiers a et b, on peut calculer la fraction continue égale à $\frac{a}{b}$ en prenant la suite des quotients dans l'algorithme d'Euclide : on pose $q_0 = a \operatorname{div} b$ et $r_0 = a \bmod b$, $q_1 = b \operatorname{div} r_0$ et $r_1 = b \bmod r_0$ puis, tant que $r_{i-1} \neq 0$, $q_i = r_{i-2} \operatorname{div} r_{i-1}$ et $r_i = r_{i-2} \bmod r_{i-1}$. Les quotients obtenus vérifient $\frac{a}{b} = [q_0, \dots, q_m]$.
Pour tout i, on appelle *(fraction continue) réduite* de $\frac{a}{b}$ à l'ordre i la fraction $\frac{n_i}{d_i} = [q_0, \dots, q_i]$.

1. **i.** Écrire une fonction `fraction_continue(a,b)` qui calcule la fraction continue de $\frac{a}{b}$, sous la forme de la liste des q_i. *Vérifier que la fraction continue de* $\frac{1789}{1664}$ *est* $[1, 13, 3, 4, 1, 7]$.
   ii. Écrire une fonction `reduite(Q, i)` qui prend en entrée la fraction continue de $\frac{a}{b}$ et renvoie sa réduite d'ordre i. *Vérifier que les réduites de* $\frac{1789}{1664}$ *sont (dans l'ordre)* $\frac{1}{1}$, $\frac{14}{13}$, $\frac{43}{40}$, $\frac{186}{173}$, $\frac{229}{213}$ *et* $\frac{1789}{1664}$.

L'attaque de Wiener se base sur le résultat suivant.
Théorème de Legendre. *Soit a, b, c, $d \in \mathbb{N}$ tels que $\text{PGCD}(a, b) = \text{PGCD}(c, d) = 1$ et $\left|\frac{a}{b} - \frac{c}{d}\right| \leq \frac{1}{2d^2}$, alors $\frac{c}{d}$ est égale à l'une des réduites de $\frac{a}{b}$.*

Soit $N = p \times q$ un module RSA avec $p < q < 2p$, et e et d les exposants de chiffrement et déchiffrement. Il existe donc k tel que $ed = 1 + k\varphi(N)$ où $\varphi(N) = (p-1)(q-1)$. On admet [2] que $k < d$ et $\left|\frac{e}{N} - \frac{k}{d}\right| \leq \frac{3k}{d\sqrt{N}}$. Si d est suffisamment petit, alors le théorème de Legendre affirme que $\frac{k}{d}$ est une des réduites de $\frac{e}{N}$. Pour l'identifier, il suffit de chiffrer un message m de son choix. On obtient c et il faut trouver, parmi les dénominateurs des réduites de $\frac{e}{N}$, ceux qui déchiffrent correctement c en m.

2. **i.** Quelle borne sur d d'appliquer le théorème de Legendre ?
   ii. Écrire un algorithme `Wiener(N, e)` qui prend en entrée une clef publique RSA et tente de trouver la clef secrète e grâce à l'algorithme esquissé ci-dessus. *On peut supposer que l'algorithme ne sera utilisé que dans le cas où d est effectivement suffisamment petit.*
   iii. Appliquer l'algorithme pour déchiffrer les messages du fichier `challenges.txt`, qui sont des textes en caractère ASCII sur 7 bits, encodés de la manière suivante : un message $m = m_0 \cdots m_t$ est transformé en l'entier $\sum_{i=0}^{t} v_i 2^{7i}$ où v_i est le code ASCII du caractère m_i, puis chiffré à l'aide de la clef publique, et l'entier obtenu est retransformé en chaîne ASCII pour obtenir le chiffré c. *Note. En Python, on passe d'un caractère ASCII m_i à sa valeur v_i avec la fonction `ord`, et l'opération inverse se fait avec la fonction `chr`.*

2. Ce n'est pas très dur à démontrer, faites le !