TP4 : Programmation de fonctions cryptographiques en Java (introduction)

L'objectif de ce travail est de se familiariser avec la programmation en Java de quelques fonctions cryptographiques. Les API Java contiennent un certain nombre de classes qui permettent de représenter aisément des concepts cryptographiques. Voir par exemple la classe `Biginteger`

`http://download.oracle.com/javase/1.4.2/docs/api/java/math/BigInteger.html`

et plus généralement l'extension `Java cryptography extension`

`http://download.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html`

À la page

`http://www.pps.jussieu.fr/~amadio/Ens/Securite/ExCodeJava.tar.gz`

vous disposez d'un code Java produit par vos prédécesseurs qui contient :

1. Un générateur de nombres pseudo-aléatoires.
2. Un testeur de primalité type Miller-Rabin.
3. Une méthode de factorisation en nombre premiers.
4. Les fonctions de chiffrement et de déchiffrement RSA.

Étudiez, modifiez et testez ces programmes. Utilisez-les ensuite pour :

1. Déterminer le plus petit nombre premier sur 512 bits.
2. Factoriser : 831802500, 18533588383 et 562952088322483.
3. Chiffrer et déchiffrer avec RSA.

TP5 : fonctions cryptographiques en Java (problèmes)

L'objectif de ce travail est de programmer en Java deux fonctions de hachage et de les évaluer de façon expérimentale du point de vue de l'efficacité et de la sécurité.

Réseaux de substitutions et de permutations

Afin de définir la première fonction de hachage, nous allons d'abord programmer un chiffrement à bloc :

$$E_k : 2^{16} \to 2^{16}$$

basé sur un réseau de substitutions et de permutations (RSP). Il s'agit d'un chiffrement en quatre tours, chacun composant un XOR avec la clé puis une substitution puis une permutation, comme décrit ci-dessous.

Substitution La substitution correspond à la troisième ligne de la première boîte de DES qui est la fonction $S : 2^4 \to 2^4$ spécifiée par le tableau suivant :

Entrée :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sortie :	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0

Permutation Comme permutation on utilise la fonction $P : \{1;:::;16\} \to \{1;:::;16\}$ spécifiée par le tableau suivant :

Entrée :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sortie :	7	3	2	4	8	10	13	12	14	5	1	11	16	15	6	9

Clef La clé K de départ est aussi sur 16 bits. La clé K_i du tour i pour $i = 1;2;3;4$ est obtenue en effectuant $4*(i-1)$ décalages circulaires vers la droite de K. Ainsi si $K = x_1 \cdot x_2 \cdot x_3 \cdot x_4$ et $x_i \in 2^4$ pour $i = 1;2;3;4$, on aura :

$$\begin{array}{lcl} K_1 & = & x_1 \cdot x_2 \cdot x_3 \cdot x_4 \\ K_2 & = & x_4 \cdot x_1 \cdot x_2 \cdot x_3 \\ K_3 & = & x_3 \cdot x_4 \cdot x_1 \cdot x_2 \\ K_4 & = & x_2 \cdot x_3 \cdot x_4 \cdot x_1 \end{array}$$

Tour Le chiffrement se décompose en 4 tours. Le tour i, pour $i = 1;2;3;4$ consiste à transformer un bloc $x = (x_1 \cdot x_2 \cdot x_3 \cdot x_4)$, où $x_i \in 2^4$ comme suit :

$$\begin{array}{lcll} (y_1 \cdot y_2 \cdot y_3 \cdot y_4) & = & (x \oplus K_i) & \text{(XOR avec la clé)} \\ (w_1 \cdot w_2 \cdot w_3 \cdot w_4) & = & (S(y_1) \cdot S(y_2) \cdot S(y_3) \cdot S(y_4)) & \text{(Substitution)} \\ z & = & P(w_1 \cdot w_2 \cdot w_3 \cdot w_4) & \text{(Permutation)} \end{array}$$

- Programmez les fonctions de chiffrement et de déchiffrement.
- Pour hacher un mot

  $$x = x_1 \cdots x_n \quad x_i \in 2^{16} \text{ pour } i = 1;:::;n$$

  on adapte le mode de *transmission CBC* :

  $$\begin{array}{ll} y_0 & = IV \\ y_{i+1} & = E_k(y_i \oplus x_i) \quad i = 0;:::;n-1 \\ h(x) & = y_n \end{array}$$

  Notez que cette fonction de hachage dépend de la clef et du vecteur d'initialisation IV. Programmez cette fonction de hachage en utilisant comme fonction E_k la fonction RSP ci-dessus.

La seconde fonction de hachage est basée sur la fonction de compression de Chaum *et al*.

Recherche d'un générateur

Programmez un algorithme probabiliste qui étant donné un nombre naturel n trouve $p; a$ tels que p est représenté sur n bits, p et $(p-1)/2$ sont premiers et a est un générateur pour $(\mathbb{Z}_p)^*$. Dans la recherche d'un générateur, vous pouvez utiliser le résultat suivant.

Théorème Si $p > 2$ est premier et $a \in (\mathbb{Z}_p)^*$ alors a est un générateur du groupe multiplicatif si et seulement si $(a^{(p-1)/r} \not\equiv 1) \bmod p$ pour tout r premier tel que $r \mid (p-1)$.

Fonction de Chaum

Programmez la fonction de Chaum *et al* :

$$c(x_1; x_2) = (a^{x_1} b^{x_2}) \bmod p$$

où $p; (p-1)/2$ sont premiers, a est un générateur de $(\mathbb{Z}_p)^*$ et $b \in (\mathbb{Z}_p)^*$ quelconque. Dérivez de cette fonction une fonction de compression qui travaille sur des séquences de bits et qui opère avec un nombre p premier qui peut être représenté avec (environ) 16 bits.

Construction de Merkle et Damgard

Programmez la construction de Merkle et Damgard considérée dans le cours qui permet de construire une fonction de hachage à partir d'une fonction de compression. Dérivez une fonction de hachage en utilisant comme fonction de compression celle de Chaum et al. sur 16 bits (voir ci-dessus). Vous pouvez supposer que la longueur du texte à hacher est limitée à 2^{16} bits.

Comparaison

Comparez de façon expérimentale les deux fonctions de hachage obtenues en terme de :

Efficacité Vitesse de la fonction de hachage.

Image inverse Étant donnée une valeur hachée y (16 bits), déterminez le nombre d'essais nécessaires (en moyenne) pour trouver une image inverse x d'une taille inférieure ou égale à 10^3 bits.

Vous veillerez à échantillonner vos tests sur tous les paramètres des fonctions en question.

Vous êtes encouragés à utiliser autant que possible les fonctionnalités de Java, c'est-à-dire à minimiser votre effort de programmation ! Vous pouvez aussi utiliser le code Java disponible pour le TP4. Vous devez produire un code bien documenté, un rapport de deux pages maximum et préparer une démonstration de 10 minutes de vos programmes.