Université Paris 8 J. Lavauzelle

Master 1 Mathématiques et applications — année 2022–23

Cryptographie à clé publique – Devoir 1

03/02/2023

Consignes :

1. à rendre par email avant le vendredi 17/02/2023 ;
2. le code doit être commenté ;
3. pour la question 6, une réponse argumentée est attendue ;
4. il est conseillé d'utiliser `python` et sa bibliothèque externe `cryptodome` et ses fonctions `getPrime, isPrime`, mais tout autre langage (standard) est accepté. Pour l'installation de `cryptodome`, voir :
`https://pycryptodome.readthedocs.io/en/latest/src/installation.html`

Exercice 1. Implantation de RSA brut avec déchiffrement accéléré.

Dans cet exercice, on souhaite implanter la version brute du chiffrement RSA, dans deux versions de déchiffrement : l'une standard et l'autre accélérée.

Pour cela, on considère $n = pq$ un module RSA, et (e, d) les exposants public/privé. La clé publique est $\texttt{pk} = (n, e)$ et, en supposant qu'on ajoute le module RSA à la clé privée, la clé privée est $\texttt{sk} = (n, d)$.

Question 1.– [Génération de clés RSA]
À l'aide de bibliothèques appropriées, implanter une fonction `keygen_RSA(t)` de génération de clés RSA `pk` et `sk`, telles que les nombres premiers p et q sont de taille `t` bits.

Question 2.– [Chiffrement et déchiffrement standard]
Implanter la fonction de chiffrement `encrypt_RSA(m, pk)` et la fonction de déchiffrement `decrypt_RSA(c, sk)`, où `m` est un message à chiffrer (donc un entier entre 0 et $n-1$), et où `c` est un chiffré RSA. On prendra garde à effectuer une exponentiation modulaire rapide.

Question 3.– [Test]
Tester la validité des premières fonctions avec $t = 256$ (la génération de clés devrait être quasi-immédiate).

Attention ! Pour rappel, en pratique $t = 256$ est largement insuffisant pour la sécurité de RSA.

On note maintenant $d_p := d \mod (p-1)$ et $d_q := d \mod (q-1)$, ainsi que u et v les coefficients de Bezout associés à p et q, de sorte que $up + vq = 1$.

Question 4.– [Précalcul d'entiers auxiliaires]
Implanter une fonction `compute_key_elements(p, q, e)` qui retourne les valeurs de n, u, v, d_p et d_q à partir de p, q et e.

On suppose maintenant que la privée `sk_aux` contient les entiers d_p, d_q, u, v, p, q et n.

Question 5.– [Déchiffrement accéléré]
Implanter la fonction de déchiffrement rapide `decrypt_RSA_fast(c, sk_aux)`, qui utilise le théorème des restes chinois (voir cours).

Question 6.– [Vitesse de déchiffrement]
Comparer expérimentalement la vitesse des deux versions de l'algorithme de déchiffrement.

Indication : pour cela, on pourra prendre $t = 256$ *pour générer les clés, choisir un message aléatoire, calculer un chiffré* `c`*, puis comparer la vitesse d'exécution de* `decrypt_RSA_fast(c, sk_aux)` *et de* `decrypt_RSA(c, sk)`*.*

Pour comparer le temps d'exécution d'une fonction, on peut utiliser la fonction `time.time()`*, voir par exemple dans la ressource suivante :* `https://www.ukonline.be/cours/python/opti/chapitre3-1`*. On peut aussi avoir intérêt à répéter et prendre la moyenne/médiane de plusieurs mesures pour éviter des effets de bords.*