TP6 - Chiffrement asymétrique

Axel REMACK - Stacy VARLOTEAUX

17 novembre 2021

1 Génération des clés RSA avec GMP

Le chiffrement RSA est souvent utilisé pour communiquer une clé de chiffrement symétrique qui permet de poursuivre un échange de façon confidentielle. Le chiffrement RSA repose sur un utilisateur qui créé son couple clé privée/clé publique (bi-clé) en utilisant une certaine procédure dont le code va être expliqué ci-après. La taille des clés étant conséquentes, il faut utiliser une bibliothèque particulière pour gérer les grands entiers (GMP).

La première étape de création des clés consiste à générer aléatoirement 2 nombres premiers distincts **p** et **q**. Tout d'abord, il faut initialiser le générateur pseudo-aléatoire avec la fonction *gmp_randseed_ui()* de GMP permettant d'initialiser le seed. Ensuite, on génère deux nombre aléatoires via la fonction *mpz_urandomb()* avec une limite de taille (entre 0 et $2^{primesize}$). A partir de ces derniers, on va chercher le prochain nombre premier grâce à la fonction *mpz_nextprime()*. A la sortie, **p** et **q** sont des nombres premiers générés aléatoirement grâce aux fonctions de GMP.

La deuxième étape est celle du calcul du module de chiffrement **n = p * q**, ainsi que la valeur de l'indicatrice d'Euler en n nommé **x = (p - 1)*(q - 1)**. Ces calculs se résument à une multiplication effectuée avec GMP par la fonction *mpz_mul()*, ainsi que l'utilisation de *mpz_sub_ui()* pour les soustractions.

La troisième étape est le calcul aléatoire de l'exposant de chiffrement **e** qui rempli certains critère : ce doit être un entier naturel premier avec l'indicatrice d'Euler et doit lui être strictement inférieur. Une fois de plus la fonction *mpz_urandomb()* est utilisée pour générer un nombre aléatoire. Seulement, comme il est aléatoire, on continue de le tirer jusqu'à ce qu'il remplisse les critères, vérifiés via la fonction *mpz_gcd()*. Elle permet de calculer le plus grand diviseur commun à l'exposant de chiffrement et à l'indicatrice d'Euler pour voir si l'exposant est bien premier avec l'indicatrice.

La dernière étape est celle du calcul de l'exposant de déchiffrement **d** qui se trouve être l'inverse de l'exposant de chiffrement modulo l'indicatrice d'Euler. L'inverse se calcul grâce à la fonction *mpz_invert()* de GMP.

La clé publique est **(e,n)** et la clé privée est **(d,n)**.

2 Remplacement des fonctions GMP

2.1 Remplacement de *mpz_powm()* par l'algorithme d'exponentiation rapide

L'algorithme d'exponentiation rapide est utilisé pour le calcul des puissances de grands nombres entiers positifs d'un nombre. Il permet de renvoyer un message chiffré **m** à partir d'un message de base (**g**), d'un exposant de chiffrement (**k**) et d'un module de chiffrement (p). En reprenant l'algorithme décrit dans l'énoncé, on voit qu'il y a trois traitements à faire en fonction de l'exposant **k**.

Le premier traitement s'opère si k < 0 ce qui se vérifie par la fonction *mpz_cmp_si(k, 0)*. Si la condition est vérifiée, alors on utilise la méthode *mpz_fdiv_q()* pour attribuer à **g** la valeur de 1/g, et *mpz_mul_ui()* pour multiplier k par -1. Si k = 0 alors on renvoi une valeur de 1. Sinon, on va vérifier si k est pair ou non.

Pour cela, on utilise la méthode *mpz_even_p(k)* puis on applique le traitement demandé par l'algorithme. En sortie, on a m = g^k mod p.

2.2 Remplacement de *mpz_nextprime()* par le test de primalité de Rabin-Miller

Pour déterminer si un nombre est premier ou non on va utiliser le test de primalité de Rabin-Miller. C'est un test de primalité probabiliste : étant donné un nombre entier, il conclut soit de façon certaine que celui-ci est composé, soit qu'il est probablement premier.

Le principe est de tester z fois (dans notre cas 10000 fois) la primalité du nombre choisi. On va donc prendre un entier impair i > 2 et tester dans un premier temps si notre nombre potentiellement premier choisi est bien premier avec lui. Si c'est le cas, on va répéter l'opération z fois ou jusqu'à ce qu'un entier i prouve qu'il est composé. A la fin, on possède soit la preuve que notre nombre n'est pas premier, soit qu'il possède une forte probabilité d'être premier (la probabilité étant plus forte avec un z très grand).

2.3 Remplacement de *mpz_invert()* par l'algorithme d'Euclide étendu

L'algorithme d'Euclide étendu permet de calculer efficacement le plus grand diviseur commun de deux nombres (PGCD). Il sert, dans notre cas, à calculer l'exposant de déchiffrement **d** pour la création de la clé privée.

L'idée principale de l'algorithme est d'effectuer les mêmes étapes que pour l'algorithme d'Euclide classique, mais en exprimant à chaque itération le reste comme une combinaison linéaire des deux entiers fournis, a et b. Puisque le dernier reste est le PGCD, celui-ci sera alors exprimé comme une combinaison linéaire de a et b.