Fiche 208 Niveau : 2 version 2 (1/05/2010) **Orange**

Algorithme de Montgomery pour la multiplication modulaire

1 Présentation du problème, notations

Étant donnés trois nombres entiers a, b, n, le problème est de calculer le plus rapidement possible $a \cdot b \mod n$.

Disons tout de suite que si on a une seule multiplication à faire, la transformation de Montgomery ainsi que l'algorithme qui en découle, que nous présentons ici, n'ont pas d'intérêt. En revanche si des multiplications s'enchaînent comme dans le cas de l'algorithme « square and multiply » décrit dans la « fichecrypto_004_V2 » alors cet algorithme prend tout son sens.

En conclusion il faut considérer l'algorithme de Montgomery comme s'insérant dans l'algorithme « square and multiply » du calcul de $a^n \mod n$.

2 La transformation de Montgomery

2.1 Définition de la transformation - lien avec le produit

Soit n le modulo intervenant dans l'opération. **On supposera désormais que n est un nombre impair**. Le nombre de bits de n est l'entier k tel que :

$$2^{k-1} \leq n < 2^k.$$

On appellera r le nombre 2^k. Comme n est impair r est premier avec n et donc inversible modulo n. On notera r^{-1} l'inverse de r modulo n.
Soit ϕ (transformation de Montgomery) l'application de $I_n = \{0, 1 \cdots, n-1\}$ dans lui-même définie par :

$$\phi(a) = a \cdot r \mod n.$$

Cette application ϕ (multiplication par r modulo n) est une bijection de I_n dans lui-même puisque r est inversible modulo n et on peut écrire :

$$a = \phi(a) \cdot r^{-1} \mod n.$$

Le théorème qui suit nous indique comment s'exprime le transformé d'un produit de deux termes en fonction des transformés de chacun de ces deux termes.

Théorème 2.1 *Soit $c = a \cdot b \mod n$. Alors :*

$$\phi(c) = \phi(a) \cdot \phi(b) \cdot r^{-1} \mod n.$$

Preuve. Calculons le second membre de l'égalité à prouver :

$$\phi(a) \cdot \phi(b) \cdot r^{-1} \mod n = a \cdot r \cdot b \cdot r \cdot r^{-1} \mod n,$$

$$\phi(a) \cdot \phi(b) \cdot r^{-1} \mod n = a \cdot b \cdot r \mod n = c \cdot r \mod n = \phi(c).$$

□

Ceci nous conduit pour calculer $c = a \cdot b \mod n$ à calculer $\phi(a)$ et $\phi(b)$, à en déduire $\phi(c)$ par la formule précédente et enfin à trouver c par la transformation de Montgomery inverse.

2.2 Comment se calcule le transformé d'un produit

On calcule une fois pour toutes r^{-1} par l'algorithme d'Euclide étendue, et pendant qu'on y est on calcule aussi l'autre coefficient de Bézout, c'est-à-dire qu'on calcule r^{-1} et v tels que :

$$r \cdot r^{-1} - v \cdot n = 1,$$

où $0 < r^{-1} < n$ et $0 < v < r$.

Notons $\otimes$ l'opération sur $\{0, 1 \cdots, n-1\}$ définie par :

$$A \otimes B = A \cdot B \cdot r^{-1} \mod n.$$

Alors nous avons montré que :

$$\phi(a \cdot b \mod n) = \phi(a) \otimes \phi(b).$$

Le calcul de $A \otimes B$ peut s'effectuer de la manière suivante :

Algorithme pour calculer $A \otimes B$
1) $s = A \cdot B$,
2) $t = (s \cdot v) \mod r$,
3) $m = (s + t \cdot n)$,
3) $u = m/r$,
4) si $u \geq n$ **alors** $A \otimes B = u - n$ **sinon** $A \otimes B = u$.

En effet, il existe un entier k tel que :

$$t = s \cdot v + k \cdot r.$$

On calcule alors :

$$m = (s + t \cdot n) = s + s \cdot v \cdot n + k \cdot r \cdot n,$$

$$m = (s + t \cdot n) = s + s \cdot (r \cdot r^{-1} - 1) + k \cdot r \cdot n,$$

$$m = (s + t \cdot n) = s \cdot r \cdot r^{-1} + k \cdot r \cdot n,$$

$$u = s \cdot r^{-1} + k \cdot n.$$

Donc :

$$u \equiv A \otimes B \quad (n).$$

Mais on voit que

$$0 \leq s < n^2$$

et

$$0 \leq t < r,$$

donc :

$$0 \leq m < n \cdot (r+n) < 2 \cdot r \cdot n,$$

et

$$0 \leq u < 2 \cdot n.$$

Ceci prouve qu'il n'y a que deux possibilité : soit $A \otimes B = u$ soit $A \otimes B = u - n$ suivant que $0 \leq u < n$ ou $n \leq u < 2n$.

Dans cet algorithme, on ne fait pas de division par n, mais par r, ce qui change tout puisque r est une puissance de 2. Bien entendu, pour une seule opération, ceci n'est pas intéressant puisqu'il y a plusieurs calculs préparatoires à faire. Mais si on enchaîne un grand nombre d'opérations alors on devient gagnant.

3 Enchaînons les opérations

Reprenons l'algorithme square and multiply que nous avons développé dans la « fichecrypto_002_V2 » et dans la « fichecrypto_004_V2 ». On veut calculer $a^k \mod n$. On calcule alors :

1) $A = \phi(a)$
2) $P = A^{\otimes k}$ par l'algorithme square and multiply intégrant la transformation de Montgomery comme indiqué dans la « fichecrypto_002_V2 »paragraphe 3 ;
3) ayant ainsi $P = A^{\otimes k} = \phi(a^k)$ on revient à a^k par la formule $a^k = P \cdot r^{-1} \mod n$.

Pour calculer $P = A^{\otimes k}$, il suffit de constater que l'opération $\otimes$ est associative et que son élément neutre est $r \mod n = r - n = \phi(1)$. On applique donc l'algorithme du paragraphe 3 de la « fichecrypto_002_V2 ».

Square and multiply
entrées : un tableau N de $K+1$ bits
représentant k en binaire
sortie : le nombre $P = A^{\otimes k}$

début
  $P \leftarrow r - n$;
  $i \leftarrow K$;
  tant que $i \geq 0$ **faire**
    $P \leftarrow P \otimes P$;
    si $N[i] == 1$
      alors $P \leftarrow P \otimes A$;
    finsi
    $i \leftarrow i - 1$;
  fintq
  retourner P ;
fin

Le coût des opérations auxiliaires du début et de la fin du programme (transformation directe de a et inverse de $A^{\otimes k}$ ainsi que les calculs de r^{-1} et v) sont amortis par l'enchaînement des opérations de multiplication et d'élévation au carré.