Introduction à la cryptographie - TP 1

1 RSA

1.1 Exponentiation

Comment calculer A^m efficacement ?
Comme nous l'avons vu au cour précédent, l'algorithme *Square-and-Multiply* est un algorithme d'exponentiation rapide relativement efficace. Son utilisation est générique et ne tient pas compte de la spécificité de l'exposant m. Je vous ai présenté très rapidement l'algorithme d'Itoh-Tsujii qui permet d'évaluer cette quantité quand $m = 2^k - 1$ (k est entier).
Voici une séquence qui a pour but de calculer $A^{2^{232}-1}$ (contenu dans U_{10}) :

U_0	1	$\beta_1 = A$
$U_1 = U_0 + U_0$	2	$\beta_2 = \beta_1^2 * \beta_1$
$U_2 = U_1 + U_0$	3	$\beta_3 = \beta_2^2 * \beta_1$
$U_3 = U_2 + U_2$	6	$\beta_6 = \beta_3^{2^3} * \beta_3$
$U_4 = U_3 + U_0$	7	$\beta_7 = \beta_6^2 * \beta_1$
$U_5 = U_4 + U_4$	14	$\beta_{14} = \beta_7^{2^7} * \beta_7$
$U_6 = U_5 + U_5$	28	$\beta_{28} = \beta_{14}^{2^{14}} * \beta_{14}$
$U_7 = U_6 + U_0$	29	$\beta_{29} = \beta_{28}^2 * \beta_1$
$U_8 = U_7 + U_7$	58	$\beta_{58} = \beta_{29}^{2^{29}} * \beta_{29}$
$U_9 = U_8 + U_8$	116	$\beta_{116} = \beta_{58}^{2^{58}} * \beta_{58}$
$U_{10} = U_9 + U_9$	232	$\beta_{232} = \beta_{116}^{2^{116}} * \beta_{116}$

TABLE 1 – Itoh Tsuji utilisé pour $m = 2^{232} - 1$

Ici, je ne tiens pas à expliquer le fonctionnement de l'algorithme. Pour faire « simple », il se base sur des *chaines d'addition*.
Dans la séquence précédente, il y a 231 mises au carré et 10 multiplications. Si nous avions utilisé l'algorithme *Square-and-Multiply* , la complexité aurait été de 231 mises au carré et 231 multiplications.
Bref, il faut garder à l'idée qu'un algorithme générique est généralement très loin d'être le plus efficace dans des cas d'utilisation bien spécifiques.

- En utilisant le langage de votre choix (C ou $C++$ de préférence), implantez la méthode *Square-and-Multiply*.
- Vérifiez que tout naturel $A < 9859$ vérifie l'égalité

$$A^{9858} \equiv 1 \bmod 9859$$

1.2 Chiffrement RSA

En 1977, l'un premier algorithme de chiffrement à clefs publiques vit le jour. L'algorithme RSA (rien à voir avec le revenu de solidarité active) du nom de ses trois auteurs *Ronard* ***R****ivest*, *Adi* ***S****hamir* et *Leonard* ***A****dleman* fut breveté quelques années plus tard en 1983 par le MIT. L'algorithme se base sur de l'exponentiation modulaire.

Un petit exemple : Imaginons qu'Alice souhaite transférer la lettre C (encodée par 2) à son ami Bob. Bob dispose d'une clef publique $(33, 7)$.

- Alice calcule la quantité $2^7 \bmod 33 \ = 29$
- Alice envoie à Bob la quantité calculée, c'est à dire, 29.
- Bob reçoit la valeur 29 et calcule $29^3 \bmod 33 = 2$. Bob a donc reçu C. $(33, 3)$ est appelée clef privée de Bob.

Exercices

- En utilisant le langage de votre choix (C ou $C++$ de préférence), implantez l'algorithme d'inversion modulaire, c'est à dire étant donnés deux entiers premiers entre eux m et a, écrire un programme capable de trouver b tel que $a \times b = 1 \bmod m$.
- Écrire une fonction GenerateKeys qui génère un couple de clefs publiques et privées.