TD 3 de Cryptographie

INSA CVL

28 septembre 2020

Récupérer sur Célène l'archive correspondant au TD 3. Elle contient un modèle du code source à réaliser durant ce TD.

Exercice 1. *(Un peu d'arithmétique)*. Dans cet exercice, nous allons prouver quelques propriétés qui seront utiles pour comprendre le chiffrement RSA. Soit p et q deux nombres premiers. On notera $N = pq$ leur produit. Soit deux entiers e et d tels que $ed \equiv 1 \mod \phi(N)$. Enfin, soit $M \in \{0, 1, \cdots, N-1\}$ un entier.

a. Montrer que e et $\phi(N)$ sont premiers entre eux.

b. Montrer que $M^{ed} \equiv M \mod p$ et $M^{ed} \equiv M \mod q$.

c. En déduire que $M^{ed} - M$ est un multiple de N.

d. Conclure que $M^{ed} \equiv M \mod N$.

e. Appliquer ces résultats à RSA en montrant que l'algorithme de déchiffrement renvoie le message qui à été chiffré par l'algorithme de chiffrement.

f. Le chiffrement (et le déchiffrement) de RSA consiste en un calcul d'exponentiation. Proposer un algorithme simple qui calcule une exponentiation et donner sa complexité. Votre algorithme est-il utilisable en pratique pour RSA ?

g. Soit M et e deux entiers positifs. On notera $b_l b_{l-A} \cdots b_1 b_0$ l'écriture binaire de e. On aura donc :

$$e = \sum_{i=0}^{l} b_i 2^i.$$

On pose :

$$\begin{cases} M_0 = M \\ \forall i > 0, M_{i+1} = M_i^2. \end{cases}$$

Montrer que :

$$M^e = \prod_{i=0}^{l} M_i^{b_i}.$$

h. En déduire un algorithme qui calcule l'exponentiation M^e en $\log_2(e)$ calculs.

i. À l'aide de l'algorithme d'Euclide, calculer $\mathsf{pgcd}(96, 76)$, $\mathsf{pgcd}(306, 758)$, $\mathsf{pgcd}(50, 33)$ et $\mathsf{pgcd}(456, 43)$.

j. À l'aide de l'algorithme d'Euclide étendu, donner les coefficients de Bézout pour chaque couple (a, b) de l'exercice précédent qui vérifie $\mathsf{pgcd}(a, b) = 1$.

Exercice 2. *(Implémentation de RSA)*. Nous allons maintenant implémenter le système de chiffrement RSA.

a. Pour manipuler des nombres entiers, nous allons définir un nouveau type :
`typedef unsigned long long int huge;`
Pourquoi est-ce nécessaire ? Quel est la taille d'un `huge` ?

b. Implémenter une fonction qui calcule une exponentiation modulaire $a^b \mod n$ efficacement. Le prototype de la fonction est le suivant :
`static huge modexp(huge a, huge b, huge N);`

Attention ! Nous allons manipuler de très grands nombres, prenez soin de réduire vos variables modulo N le plus souvent possible dans vos algorithmes, afin d'éviter de dépasser la taille maximale supportée par les variables de type `huge`.

c. Implémenter les fonctions de chiffrement et le déchiffrement dont les prototypes sont :
`static huge RSAcrypt(huge m, huge e, huge N);`
`static huge RSAdecrypt(huge c, huge d, huge N);`
Tester que tout fonctionne bien en exécutant la fonction main.

d. Implémenter l'algorithme d'Euclide en utilisant le prototype de fonction suivant :
`static huge pgcd(huge a, huge b);`

e. En utilisant l'algorithme d'Euclide étendu, implémenter une fonction qui prend en paramètre trois entiers positifs a, b et N et qui calcule deux entiers positifs u et v tels que $au + bv \equiv 1 \mod N$. Le prototype de cette fonction doit être :
`void bezout(huge a, huge b, huge N, huge* u, huge* v);`

Attention ! les variables de type `huge` ne sont pas signées. Prenez garde à ne pas réaliser de soustraction dont le résultat soit négatif. Pour rappel, pour tout $a < N$, $-a \equiv N - a \mod N$.

f. Implémenter une fonction qui prend en paramètre deux nombres premiers et qui génère des clés RSA publique et privée. On utilisera le prototype suivant :
`void keyGen(huge p, huge q, huge * N, huge * e, huge * d);`

g. Tester votre implémentation de RSA avec les nombres premiers $p = 51109$ et $q = 51131$, en générant des clés publique et privée, et en chiffrant et déchiffrant un message.