MI44
Securité Informatique
Travaux Pratique 1: Cryptage par Bloc

ALLIOT Renaud

2 mai 2015

1 Introduction

Le but de ce travaux pratique seras de mettre en place un cryptage RSA. Pour cela on devra programmer un générateur de clé puis programmer le cryptage/décryptage puis la simulation d'un protocole.
Ayant choisis de coder en C++ ce projet, j'ai créer une classe "Msg" qui correspond au bloc sur lesquels nous travaillerons. Les précisions sur cette dernières seront présentant dans l'annexe A.

2 Génération des clés

Pour générer les clé nous avons besoin de plusieurs élément :
— Générer deux nombres aléatoire premier.
— Le choix de l'exposant de cryptage.
— Le calcul de d'un inverse modulaire

Nombre aléatoire premier

Pour générer un nombre aléatoire premier on uttilise la fonction `rand()` pour générer un nombre aléatoirement, puis l'on effectue un test de primalité et tant qu'il n'est pas vérifié on génère a nouveau le nombre.

Pour ce qui est du test, tout d'abord on exclu les cas ou le nombre est inférieur à 2, ensuite on l'arrête s'il est égal à deux. Ensuite on test si il est pair, puis on effectue une boucle de 3 à $\sqrt{p}$ avec un pas de deux pour esquiver les nombres pairs précédement testé et on test s'il est divisible avec l'un de ces nombres.

La fonction crée est `bool test_primalite(int p)` .

Choix de l'exposant

Le cryptage RSA demande que si e est l'exposant alors $pgcd(e, \phi(n)) = 1$ et $0 < e < \phi(n)$ donc on effectue une boucle qui génère e aléatoirement tant que $pgcd(e, \phi(n)) \neq 1.pgcd(e, \phi(n)) = 1$

Calcul de l'inverse modulaire

L'exposant d est l'inverse de e modulo $\phi(n)$. Pour cela on uttilise l'algorithmes d'euclide étendu qui calcul $d^{-1} \equiv e\ [\phi(n)]$.

On utilise donc la fonction suivante `int algo_euclide_gen(int inverse, int modulo)` qui renvoie l'inverse modulaire.

Implémentation

On effectue les étape dans cette ordre :

1. On génère p et q deux nombre premier aléatoirement.
2. On calcul $n = p \times q$ et $\phi(n) = (p-1)(q-1)$.
3. On calcul e.
4. On calcul d l'inverse modulaire de e modulo $\phi(n)$.
5. On renvoie le couple [(n,e),(n,d)].

La fonction de génération de clé est `couple gen_cle()`

3 Cryptage/Décryptage

Pour cela on code un algorithmes d'exponentiation modulaire afin de calculer $C \equiv M^e\ [n]$ ou $M \equiv C^d\ [n]$ avec M le message non crypté, C le message crypté, e l'exposant de chiffrement, d l'exposant de déchiffrement et n le module.

L'algorithmes pour $C \equiv M^e\ [n]$ uttilise le procédés suivant :
— On écrit e sous la forme $b_n b_{n-1} \cdots b_1 b_0$ où $b_i \in 0, 1$.
— A chaque itération allant de 1 à n, si $b_i = 1$ alors on calul $C = C \times M\ [n]$.
— Dans tout les cas on calcul $M^2[n]$.

Données : $M \in \mathbb{N},\ e \in \mathbb{N},\ n \in \mathbb{N}$
Résultat : $C \in \mathbb{N}$
début
  Calculer $e = (e_k, e_{k-1}, ..., e_1, e_0)_2$ où $e_i \in 0, 1$;
  $1 \leftarrow C$;
  $M \leftarrow base$;
  pour $i \leftarrow 0$ *à* k **faire**
    si $e_i = 1$ **alors**
      $C \times base\ [n] \leftarrow C$;
    fin
    $base \times base\ [n] \leftarrow base$;
  fin
fin

Algorithme 1 : Calcul de $C \equiv M^e\ [n]$

La fonction qui en resulte est : `double expo_modul(int message, int eOuD, int n)` .

Pour crypté j'ai donc décider d'uttiliser le tableau de conversion du TP2. Ainsi je convertit chaque caractère en son entier corespondant, que je passe dans l'algorithmes avec la clé pulique pour crypté. Il en ressort un chiffre in traductible (Car en général très grand). Pour le decryptage récupère le nombre, le passe dans la fonction de décryptage et il en ressort le code ascii du caractère décrypté. On obtiens alors les fonction suivante : `double encrypt_rsa(double message, int e, int n)` et `double decrypt_rsa(double message, int d, int n)`.

4 Le protocole

5 Conclusion

Le programme fonctionne correctement, cependant il faudrait voir pour prendre en charge les minuscule et/ou le tableau ACII (hormis les caractère de 0 à 31 et 127 qui sont des caractère purement informatique et donc illisible). On peut aussi implémenter d'autre mode de cryptage, comme le CBC, permettre à l'uttilisateur de modifier le nombre de tour, d'entrer une clé de tailler quelconque.

A Classe "Msg" : Attribut et méthodes

La classe Msg correspond à un bloc de deux caractère en binaire. Elle est uttilisé lors de Feistel.

Attribut

`vector<int> firstChar` qui est le bloc des bits de poids faible (caractère de droite).
`vector<int> secondChar>` qui est le bloc des bits de poids fort (caractère de gauche).

Méthode

`msg()` Constructeur par défaut qui met chaque caractère égal à 'A'.
`msg(const vector<int>& a, const vector<int>& b)` Constructeur surchargé qui affecte 'a' à 'firstChar' et 'b' à 'secondChar'.
`msg(char a, char b)` Constructeur surchargé qui met a dans firstChar et b dans secondChar en les convertissant.
`msg(const msg& copy)` Constructeur par copie.

`msg& operator=(const msg& c)` Surcharge de l'opérateur d'affectation.
`bool isEqual(const msg& b) const` Methode itermédiaire pour le test d'égalité et d'inégalité.
`msg& operator^=(const msg& b)` Surcharge de l'opérateur XOR raccourcis

`vector<int> getFirstChar() const` Accesseur de firstChar
`void setFirstChar(vector<int> value)` Modificateur de firstChar avec un vector<int>.
`void setFirstChar(char c)` Modificateur de firstChar en convertissant un caractère c en binaire.
`void setFirstChar(int pos, int value)` Modificateur du bit "pos" de firstChar par "value".
`vector<int> getSecondChar() const` Accesseur de secondChar. `void setSecondChar(vector<int> value)` Modificateur de secondChar avec un vector<int>.
`void setSecondChar(char c)` Modificateur de secondChar avec un caractère c convertis en binaire.
`void setSecondChar(int pos, int value)` Modificateur du bit "pos" de firstChar par "value".

`msg shiftLeftMsg(int shiftValue) const` Renvoie *this décaler de "shiftValue" vers la gauche
`msg shiftRightMsg(int shiftValue) const` Renvoie *this décaler de "shiftValue" vers la droite.
`msg function(const msg& key) const` Renvoie function(*this, key).

`void encrypt_binaire(char a, char b)` Convertit les caractère 'a' et 'b' en mot binaire qu'il met à la place de secondChar et firstChar (dans cette ordre).
`void decrypt_binaire(char& a, char& b) const` Convertit firstChar et secondChar en caractère dans 'a' et 'b'.

`void displayMsg(ostream& stream)` Ecris firstChar et secondChar dans stream.

Fonction externe

`bool operator==(const msg& a, const msg& b)` Surcharge du test d'égalité pour les msg.
`bool operator!=(const msg& a, const msg& b)` Surcharge du test d'inégalité pour les msg.
`msg operator^(const msg& a, const msg& b)` Surcharge de l'opérator XOR pour les msg.
`ostream& operator<<(ostream& stream, msg& a)` Surcharge de l'opérateur flux de sortis pour msg.