Projet Scilab
Cryptographie
Echange de clés de Diffie - Hellman
Procédé de signature MD5

Laïla Barbach
Priscilla Imbert
LM206

Sommaire

Introduction à la cryptographie

Qu’est ce que la cryptographie ?

Voici une définition obtenue dans le glossaire de Futura sciences.

Cryptographie :
Conception de mécanismes cryptologiques destinés à garantir les notions de sécurité à des fins de confidentialité, d’authenticité et d'intégrité de l'information, mais aussi pour d'autres notions comme l'anonymat.

Il existe deux types de cryptographie :

Cryptographie symétrique :

Les algorithmes sont publics, mais une information secrète commune à l'émetteur et au destinataire permet leur usage entre plusieurs personnes. On dit « symétrique » car émetteur et destinataire ont le même niveau d’information.

Cryptographie asymétrique :

Les algorithmes sont publics, mais chaque individu possède un couple de clés : l’une secrète lui permettant d’effectuer les opérations que lui seul est sensé être en mesure de faire (signature ou déchiffrement), tandis que sa clé publique est diffusée afin de permettre à ses interlocuteurs de mettre en oeuvre les opérations réciproques (vérification de signature ou chiffrement de message). Les deux clés ont un rôle «asymétrique», d’où la terminologie.

La cryptographie est l’art de chiffrer/coder les messages. C’est une science dans laquelle mathématiques et informatiques (parfois même physique) jouent un rôle primordial. Elle est le fruit d’un besoin continu pour des entités telles que des populations de garder certaines choses secrètes, que ce soit pour éviter des guerres, protéger un peuple. Elle a longtemps été réservée à une élite (militaires et sociétés possédant de gros moyens financiers) et s’est au fil du temps ouverte au grand public.
Quant à la cryptologie (science du secret), elle regroupe la cryptographie qui permet de coder les messages et la cryptanalyse, qui permet de les décoder.

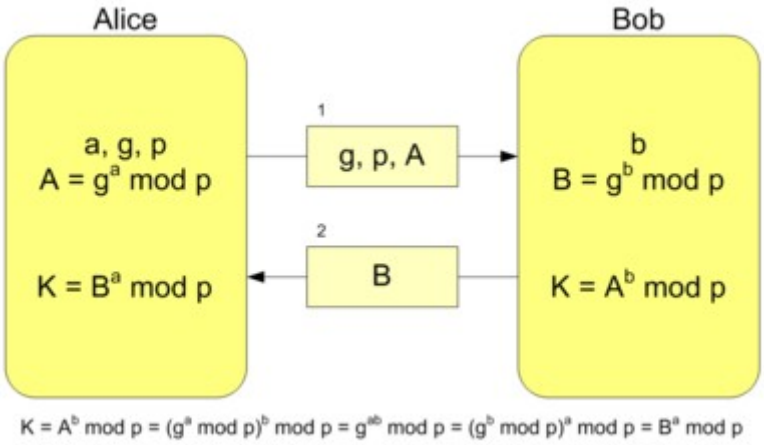
Les individus qui échangent des messages dans les différents procédés cryptographiques sont familièrement appelés Alice et Bob (les sujets A et B en réalité). Alice étant malade, Arwenn la remplacera tout au long de notre dossier.

A. Echange de clés de Diffie – Hellman

1) Un peu d'histoire…

C'est en 1976 que Whitfield Diffie et Martin Hellman, de l'université de Stanford, proposent un principe de chiffrement entièrement nouveau : la cryptographie à clé publique, ou asymétrique.

2) Principe et exemple



Principe de fonctionnement

- Arwenn et Bob choisissent p premier et une base g
- Arwenn choisit a, un entier gardé secret et élève g à la puissance a puis le transmet, modulo p, à Bob
- Bob fait de même après avoir choisi un entier b
- Arwenn élève le nombre reçu de Bob à la puissance a modulo p : elle obtient g^ba modulo p
- Bob fait de même et obtient g^ab modulo p (il obtient la même chose qu' Arwenn)
- Les deux interlocuteurs sont donc en possession de la même clé sans l'avoir explicitement échangée.

Principe mathématique

On utilise ici la notion de groupe multiplicatif (abordé en arithmétique ce semestre) : on se place dans Z/pZ. Les opérations effectuées font intervenir la notion de modulo (dans notre cas : modulo p). D'après les propriétés connues des groupes on a bien (g^b)^a = (g^a)^b.

Sécurité et Limites

La sécurité d'un tel échange réside dans le problème du logarithme discret. En effet seuls peuvent être interceptés les éléments : g^a modulo p et g^b modulo p. Il faut donc à l'intercepteur, soit l'élément a, soit l'élément b pour obtenir la clé secrète. Or

pour les obtenir à partir des éléments interceptés, il faut résoudre le problème du logarithme discret (un problème calculatoire trop complexe lorsque les éléments a, b, g et p sont correctement choisis).

Par la recherche de résolution du problème du logarithme discret, il a été établi qu'avec les méthodes de résolution actuels, un nombre premier p de l'ordre de 300 chiffres, ainsi que a et b de l'ordre de 100 chiffres, il était impossible de « casser » la clé de Diffie – Hellman. Le progrès peut donc à tout moment provoquer la chute d'un tel principe.

Il existe tout de même une méthode d'usurpation baptisée ; « l'attaque de l'homme du milieu ». Cette attaque repose sur l'interception des nombres g^a et g^b : l'homme du milieu peut, après les avoir interceptés, envoyer une autre clé g^a' à Bob et de même g^b' à Arwenn. Ainsi, il peut communiquer avec Arwenn en se faisant passer pour Bob et réciproquement sans qu'aucun des deux ne se doute qu'il a en fait échangé une clé secrète avec un imposteur.
Pour parer cette éventuelle menace, il peut être intéressant d'utiliser un procédé de signature permettant d'identifier le destinataire, comme par exemple le procédé de signature MD5 abordé par la suite. Arwenn pourra ainsi être sûre que la clé qu'elle reçoit provient bien de Bob et réciproquement.

Exemple

L'exemple suivant sera utilisé sur Scilab pour prouver que la fonction utilisée fonctionne réellement.

On suppose p = 23 et g = 3 (on prend ici de faibles nombres pour rendre les calculs plus simples, par ailleurs, on a remarqué en l'expérimentant qu'avec de trop grands nombres, Scilab traitait les calculs avec des valeurs approchées, ce qui donnait au final un résultat erroné).

Soient a = 6 et b = 15.
Alice envoie à Bob 3^6 modulo 23, c'est-à-dire 16, tandis que Bob envoie à Arwenn 3^15 modulo 23, soit 12.
Arwenn calcule ensuite la clé secrète : 12^6 modulo 23, c'est-à-dire 9.
Bob obtient quant à lui 16^15 modulo 23, soit 9. Les deux clés sont à priori les mêmes. Nous verrons par la suite grâce à Scilab que c'est bien le cas.

3) Outils utilisés

Le projet de créer deux clés de Diffie – Hellman et de vérifier leur égalité a fait intervenir un élément nouveau dans notre initiation au logiciel Scilab : la fonction modulo.

Celle-ci s'utilise de la manière suivante :si l'on veut le résultat de m, modulo n, il suffit d'écrire : modulo (m,n).
Par exemple, on sait que 24 modulo 5 vaut 4, vérifions le sur Scilab :

```
-->modulo(24,5)
 ans  =

    4.
```

4) Implémentation sur Scilab

But :

- choisir p et g
- créer une fonction qui à partir de a et b crée les deux clés (gardées secrètes) et vérifie qu'elles sont bien les mêmes
- faire choisir à l'ordinateur a et b aléatoirement (gardés secrets)
- appliquer la fonction à a et à b

Implémentation sur Scilab :

```
 _____________________________________________________
                 scilab-4.0

          Copyright (c) 1989-2006
        Consortium Scilab (INRIA, ENPC)
 _____________________________________________________

Startup execution:
  loading initial environment

-->function[z]=envoirecep(u,v)
-->x=modulo((modulo(g^u,p))^v,p);
-->y=modulo((modulo(g^v,p))^u,p);
-->z=(x==y)
-->endfunction

-->g=3
 g  =

    3.

-->p=23
 p  =

    23.

-->a=int(10*rand(1)+1);

-->b=int(10*rand(1)+1);
```

```
-->[z]=envoirecep(a,b)
 z  =

  T
```

On obtient T, qui signifie True. L'ordinateur nous garantie ici que les deux clés sont bien les mêmes (sans même nous les afficher à l'écran puisqu'elles sont gardées secrètes).

B. Procédé de signature MD5

1) Un peu d'histoire…

L'algorithme MD5 (Message Digest 5) est une fonction de hachage cryptographique très populaire, mais qui n'est plus considéré comme un algorithme sûr. Une fonction de hachage est une fonction qui a pour but d'associer à un grand ensemble de données, un ensemble plus petit (de l'ordre de quelques centaines de bits) qui dépend de paramètres de l'ensemble de départ. Le MD5 a été inventé par Ronald Rivest, en 1991, la précédente version MD4 étant devenue trop vulnérable.

2) Principe

Principe de fonctionnement

Arwenn voudrait envoyer un message à Bob, la signature MD5 va permettre à Bob d'être sûr que ce message provient bien d' Arwenn et de personne d'autre.
Arwenn écrit un message et l'accompagne de sa signature qui dépend du message et de quatre mots de 32 bits (A, B, C et D) gardés secrets et connus uniquement d' Arwenn et Bob.
Bob en recevant le message accompagné de la signature vérifie que celle-ci est la bonne en recalculant la signature sachant que Bob possède A, B, C, D et le texte du message.

Principe mathématique

Nous ne nous sommes pas attachées à comprendre le principe de signature MD5, mais seulement à l'appliquer, le niveau de ce procédé cryptographique étant plus élevé que celui précédemment étudié.

Les fonctions mathématiques utilisées dans le procédé de signature MD5 sont les suivantes :

F(B,C,D) = min((B x C)+((1-B) x D),1)
G(B,C,D) = min((B x D)+(C x (1-D)),1)
H(B,C,D) = (B+C)+D (ces « + » ont ici valeur en logique de ou exclusifs)
I(B,C,D) = C + (B+(1-D)) (le premier « + » a ici valeur en logique de ou exclusif)

Sécurité et Limites

La sécurité d'un tel procédé réside dans le fait que même si un message et sa signature son interceptées, ils ne communiquent pas à l'imposteur les éléments de base de création de la signature qui sont A, B, C, D et des constantes. Et si l'imposteur décide de se resservir de cette signature, cela ne peut lui permettre que de renvoyer un message identique, la signature interceptée ne correspondant qu'à un seul message : le message intercepté.

En 1996, une faille a été découverte dans MD5 et les scientifiques ont suggéré l'utilisation de fonctions plus robustes comme SHA1. En 2004, une équipe chinoise a découvert des collisions complètes : MD5 n'est donc plus considéré comme fiable. De nos jours, il existe des pages web au contenu différent mais ayant le même MD5, ce qui peut entraîner l'infiltration de virus informatiques non détectés. MD5 reste encore utilisé comme outil de vérification lors de téléchargements bien que SHA1 le remplace de plus en plus.

3) Outils utilisés

Le projet d'implémenter le procédé de signature MD5 à fait intervenir deux éléments nouveaux dans notre initiation au logiciel Scilab : la fonction for et l'opération de décalage.

L'opération de décalage devait permettre, à partir d'un tableau de 64 lignes et 32 colonnes de décaler pour chaque ligne, les cinq premiers éléments à la fin de la ligne.
Soit E le bloc initial, pour obtenir le bloc transformé à l'aide de l'opération de décalage précédemment définie il faut réaliser dans Scilab, l'opération suivante :

E=[E(1 :$,6 :$),E(1 :$,1 :5)]

Explication : E devient l'apposition du bloc allant de sa 6ème à sa dernière colonne (pour toutes les lignes : 1 :$) et du bloc allant de la 1ère à la 5ème colonne (également pour toutes les lignes).

Quant à la fonction for que nous n'avions encore jamais rencontrée, elle s'utilise de la manière suivante :
Pour une implémentation de la ligne m à n, il suffit d'écrire : for i = m : n
On inscrit ensuite le sujet de l'implémentation, autrement dit l'opération que l'on souhaite réaliser pour chaque ligne suivi de l'inscription « end » afin d'indiquer à la machine que l'instruction est terminée.

4) Implémentation sur Scilab

But :

- faire choisir aléatoirement à l'ordinateur A, B, C et D, quatre lignes de 32 bits, qui correspondent en réalité aux quatre clés choisies et connues uniquement d' Arwenn et Bob, sans que celui-ci les affiche (ces clés sont secrètes)
- simuler un texte M d'environ 64 x 4 caractères traduits en binaire en faisant choisir aléatoirement à l'ordinateur un tableau de valeurs binaires de 64 lignes par 32 colonnes
- établir un tableau de constantes aléatoires, comprises entre 0 et 99, de la même taille (ces constantes sont normalement définies par des formules trigonométriques mais il importe peu de les choisir aléatoires)
- définir les quatre fonctions dans un langage informatique connu de Scilab
- réaliser le procédé de signature MD5 en réalisant 4 implémentations sur chacun des 4 blocs de 16 lignes avec pour chacun l'ordre suivant :

A←D
D←C
C←B
B←(A+F(B,C,D)+Mi+Ki) <<5

Les quatre blocs font successivement intervenir les fonctions F, G, H et I.
Mi et Ki sont les ièmes lignes de leurs tableaux respectifs.
<< 5 correspond au décalage explicité ci-dessus

Problème : la quantité A+F(B,C,D)+Mi+Ki doit être binaire, l'opération pouvant convertir cette quantité en binaire n'a pas été réalisé lors de l'expérience sur Scilab, ce qui a donné lieu à de petites erreurs. Ce point sera évoqué dans la partie réservée aux problèmes rencontrés et une méthode de solution sera abordée.

- la signature sera le dernier bloc A, B, C, D obtenu après tous ces calculs.

Implémentation sur Scilab :

```
 _____________________________________________
                 scilab-4.0

          Copyright (c) 1989-2006
        Consortium Scilab (INRIA, ENPC)
 _____________________________________________

Startup execution:
  loading initial environment

-->A=int(2*rand(1,32));

-->B=int(2*rand(1,32));

-->C=int(2*rand(1,32));

-->D=int(2*rand(1,32));

-->M=int(2*rand(64,32));

-->K=int(2*rand(64,32));

-->function[x]=F(B,C,D)
-->x=min((B.*C)+((1-B).*D),1)
-->endfunction

-->function[x]=G(B,C,D)
-->x=min((B.*D)+(C.*(1-D)),1)
-->endfunction

-->function[x]=H(B,C,D)
-->x=modulo((modulo(B+C,2))+D,2)
-->endfunction

-->function[x]=I(B,C,D)
-->x=modulo(C+(B+(1-D)),2)
-->endfunction

-->for i =1:16
-->E=A+F(B,C,D)+M(i,:)+K(i,:)
```

```
-->F=[E(1:$,6:$),E(1:$,1:5)]
-->A=D
-->D=C
-->C=B
-->B=F
-->end
```

On obtient pour la première boucle le résultat suivant :

```
E  =


        column  1 to 23

   3.   3.   3.   0.   3.   2.   2.   2.   4.   1.   4.   2.   2.   1.   4.   1.   1.   1.   3.
2.
   2.   3.   2.

        column 24 to 32

   1.   1.   2.   3.   2.   1.   3.   1.   1.
Warning :redefining function: F

 F  =


        column  1 to 23

   2.   2.   2.   4.   1.   4.   2.   2.   1.   4.   1.   1.   1.   3.   2.   2.   3.   2.   1.
1.
   2.   3.   2.

        column 24 to 32

   1.   3.   1.   1.   3.   3.   3.   0.   3.
 A  =


        column  1 to 23

   1.   0.   1.   1.   1.   1.   1.   0.   1.   0.   1.   0.   1.   1.   1.   0.   1.   0.   0.
1.
   0.   1.   1.

        column 24 to 32

   0.   0.   1.   0.   0.   1.   0.   0.   0.
 D  =


        column  1 to 23
```

```
   0.   1.   0.   0.   0.   1.   1.   0.   0.   1.   1.   0.   1.   0.   0.   1.   1.   0.   0.
1.
   0.   0.   1.

       column 24 to 32

   1.   1.   1.   1.   1.   0.   1.   0.   1.
 C  =


       column  1 to 23

   0.   1.   0.   1.   0.   0.   0.   0.   0.   0.   1.   0.   0.   1.   0.   1.   1.   0.   0.
1.
   0.   0.   1.

       column 24 to 32

   1.   1.   1.   1.   0.   0.   1.   1.   0.
 B  =


       column  1 to 23

   2.   2.   2.   4.   1.   4.   2.   2.   1.   4.   1.   1.   1.   3.   2.   2.   3.   2.   1.
1.
   2.   3.   2.

       column 24 to 32

   1.   3.   1.   1.   3.   3.   3.   0.   3.
 !--error 21
invalid index
at line      13 of function %s_e called by :
E=A+F(B,C,D)+M(i,:)+K(i,:)
```

Comme “prévu”, il y a de petites erreurs lors de l’exécution...
Voici, les informations à donner à l’ordinateur pour achever le procédé de signature MD5. Ces opérations étant similaires à celles du premier bloc, il est inutile d’afficher les résultats obtenus. Nous rappelons que la signature obtenue est le dernier bloc A, B, C, D calculé.

```
-->for i =17:32
-->E=A+G(B,C,D)+M(i,:)+K(i,:)
-->F=[E(1:$,6:$),E(1:$,1:5)]
-->A=D
-->D=C
-->C=B
-->B=F
-->end
```

```
-->for i =33:48
-->E=A+H(B,C,D)+M(i,:)+K(i,:)
-->F=[E(1:$,6:$),E(1:$,1:5)]
-->A=D
-->D=C
-->C=B
-->B=F
-->end

-->for i =49:64
-->E=A+I(B,C,D)+M(i,:)+K(i,:)
-->F=[E(1:$,6:$),E(1:$,1:5)]
-->A=D
-->D=C
-->C=B
-->B=F
-->end
```

Conclusion : avantages et inconvénients de Scilab

1) Brève comparaison avec C ++

Comparons brièvement le langage C++ et le langage Scilab.

Tout comme Scilab, C++ fait intervenir un ensemble de fonctions préenregistrées dont certaines sont même similaires (comme par exemple la fonction for).
Cependant, l'utilisation de C++ est à notre sens plus interactive et plus pratique que celle de Scilab même si Scilab est d'un niveau de compréhension moins élevée.

Par exemple, on trouve sur C++ des fonctions d'affichage telles que printf et scanf qui permettent un contrôle de l'affichage des données plus précis qu'avec Scilab.

Celui-ci nous donne les résultats à la fin des calculs ou nous les omet grâce à l'utilisation du point virgule mais il ne nous permet pas comme avec scanf de provoquer une demande à l'utilisateur d'insertion de données, au cours même de l'exécution. Cette interactivité de C++ donne un aspect plus ludique à ce langage tandis que Scilab reste assez froid et donne une faible lisibilité du travail réalisé.

Il y a sur C++ une variété dans le type des variables utilisées et traitées. Certes, il peut paraître contraignant avec ce langage de devoir sans cesse définir la nature des variables mises en jeu mais il offre ainsi plus de possibilités et surtout une meilleure traçabilité lorsque des erreurs sont commises. Ainsi, il est plus facile de prendre conscience de son erreur dans le choix d'un type de variable sur C++ (surtout quand la machine le signale explicitement) que sur Scilab où l'erreur apparaît lors de la demande du résultat mais est plus difficile à repérer et à corriger. En effet, il peut être fastidieux de changer une expression mathématiques complexe entière pour corriger une erreur sur Scilab que de remplacer la donnée de nature de variable par une autre (préalablement communiquées à la machine avant les opérations à effectuer avec ces variables) avec C++.

Nous émettons toutefois une réserve quant à cette comparaison, notre connaissance de Scilab n'étant pas complète, elle ne nous permet pas d'être sûr de toutes les possibilités offertes par Scilab.

2) Principaux problèmes rencontrés

Un problème qui a sûrement été récurrent parmi tous les groupes réalisant ce projet a été le manque de documentation et son inaccessibilité. En effet, il a été dur de trouver les informations concernant les différentes fonctions que nous voulions utiliser, le système nous interdisant leur accès presque systématiquement. Difficile donc de ne pas perdre de temps dans ces conditions.

Un second problème, cette fois-ci mathématique, abordé plus haut a été une erreur rencontrée lors de l'exécution de notre programme de signature MD5. En effet, les résultats que nous obtenions qui normalement devaient être binaires se sont révélés faux. L'erreur provenait en fait de la non traduction en binaire de la valeur suivante :

(A+F(B,C,D)+Mi+Ki)

Par manque de temps et de connaissances, cette erreur a été laissée de côté. Nous allons maintenant y réfléchir et proposer une méthode de résolution approchée.

Problématique : Soit X, une quantité non binaire, nous allons créer un algorithme de transformation de cette quantité en quantité binaire.

Soit i, la quantité non binaire à convertir, l'algorithme serait :

Si i < 2
    fin
Si i > ou = 2
    i ← i – 2

Jusqu'à ce que i < 2

(La traduction en langage Scilab de cet algorithme ne sera pas abordée du fait du manque de documentation concernant les fonctions « if » et « until ».)

Par conséquent, il faudra prendre la somme S1 = « A + F(B,C,D) » et la transformer en quantité binaire grâce à l'algorithme ci-dessus. Ensuite, on considérera S2 = « S1 + Mi » et on effectuera la même opération. Et de même avec « S2 + Ki ».

Nous émettons une réserve quant à la validité de ce processus, nous ne faisons ici qu'une hypothèse de résolution du problème.

Bonnes vacances !