CRYPTOGRAPHIE
TD N°2

Objectif du TD: Avoir quelques notions sur les algorithmes employés
Il ne s'agit pas d'un TD mais d'un cours/td...

ENONCE :

Chiffrement symmétrique :
Les algos de chiffrement symmétrique reposent (a ce que j'en sais) toutes sur la notion de XOR, comme le one time pad. La clef est de longueur donnée

=> adapter la taille de la clef et la taille du message.

Chiffrement asymétrique :

Les méthodes de chiffrement asymétrique (à ce que j'en sais) consistent à transformer le message en un nombre puis à effectuer un calcul compliqué à l'aide de la clef publique (pour chiffrer). La clef privée permettra de faire un calcul secondaire conduisant au message d'origine.

Exemple : RSA : la clef publique est un doublet (e,n). Si Alice veut chiffrer son message, elle le transforme en un nombre entier M. Ce nombre doit etre plus petit que n.
Le message chiffré est tout simplement $C=M^e \%n$
Pour déchiffrer, Alice dispose d'une clef privée d. Le déchiffrement est fait ainsi :
$D=C^d \%n$

Evidemment, on désire obtenir D=M...Cette propriété est assurée par la méthode par laquelle sont calculés e, d et n (et qui les lie).

Dans le cas de RSA, la fonction de chiffrement est identique à la fonction de déchiffrement. Ce n'est pas nécessairement le cas. Quoiqu'il en soit, on imagine bien qu'il est plus long de faire de ces calculs qu'un XOR !

Ici encore, il faut trouver un moyen d'adapter la taille de la clef (ici n) aux messages trop gros !

Méthodes générales : quel que soit le type de chiffrement (symétrique ou asymétrique) et l'algorithme de chiffrement (DSA, DES....), le problème est d'appliquer un algo travaillant avec une clef de taille fixe sur un message de taille variable.

Il existe deux méthodes : on découpe le message en blocs de taille fixe adéquate (chiffrement par blocs : block cypher) ou on génère une chaine de la taille voulue a partir de la clef (chiffrement en série ou chiffrement par flux : stream cypher)

Chiffrement en série : (stream cypher)
le principe : construire une chaine de taille N qui soit « aléatoire » sur la base de la clef. Les algos qui utilisent cela font ensuite un XOR des deux chaines... Pas de standards (a ma connaissance)

Chiffrement par blocs. (block-cypher)

le principe : séparer en mots de taille fixe n et appliquer l'algo de chiffrement pour chaque bloc.
- ECB (Electronic Code Book).

T[n] : nième bloc de texte clair.
C[n] : nième bloc de texte chiffré.
E(m) : fonction de chiffrement du bloc m.
D(m) : fonction de déchiffrement du bloc m.

Chiffrement : C[n] = E(T[n])
Déchiffrement : T[n] = D(C[n])

problèmes : Risque de faire conserver les redondances dans le texte chiffré => mélanger les blocs.

Solutions : prise en compte des blocs précédents dans le chiffrement. (pour simplifier : bloc XOR clef XOR bloc précédent (clair ou chiffré).

- CBC (Cipher Block Chaining) :
On rajoute :
VI = vecteur d'initialisation (connu des deux)
^ = OU Exclusif

Chiffrement : C[0] = E(T[0] ^ VI)
C[n] = E(T[n] ^ C[n-1]) , si (n > 0)
Déchiffrement : T[0] = D(C[0] ^ VI)
T[n] = D(C[n] ^ C[n-1]) , si (n > 0)

- CFB (Cypher FeedBack)
Chiffrement : C[0] = T[0] ^ E(VI)
C[n] = T[n] ^ E(T[n-1]) , si (n > 0)
Déchiffrement : T[0] = C[0] ^ E(VI)
T[n] = C[n] ^ E(C[n-1]) , si (n > 0)

Déchiffrement et chiffrement se font de la même manière !

- OFB (Output FeedBack) : je le garde pour le partiel....

Remarque sur les algos de chiffrement symétrique par blocs :
En général, pour un bloc a coder, un algorithme de chiffrement symétrique va mélanger les données a chiffrer de façon a atténuer encore les redondances du message. Ces données mélangées sont alors chiffrées avec la clef (par XOR). Le résultat est ensuite mixé avec les blocs précédents en fonction du mode opératoire....

Importance des tailles de clefs :
Induisent la difficulté d'une attaque en force Brute.
Imaginez une machine capable de tester une clef toutes les t s.
- En combien de temps aura elle fait le tour des possibilités de clefs pour une clef de n bits ?
- Combien de temps en moyenne lui faudra-t-il pour craquer une clef de n bits ?

Faites l'application numérique du temps moyen pour t=10^-9s et n = 64, 128, 256 et 512 bits ?

Faisabilité d'une analyse en facteurs premiers :
En fait, pour savoir si un nombre est premier, on ne teste pas ses diviseurs ! On utilise d'autres propriétés. En revanche, pour connaître les facteurs premiers d'un nombre, disons pour simplifier qu'on est pour le moment limités a tester ses diviseurs.

Soit un nombre N a 200 décimales... il est plus grand que 10^199. Combien de tests faudrait il faire pour connaître ses diviseurs ? Imaginez une machine capable de faire un test en 10^-10 secondes, combien lui faudra-t-il de temps pour calculer ceci ?

Du coup, en prenant deux nombres premiers de 100 décimales et en les multipliant, on est a peu prêt assuré que personne ne les retrouve à partir de leur produit !

Hachage et Signature :
Ici, nous allons voir les méthodes de calculs de Digest pour des messages de taille quelconques. Notre méthode de hachage (sommaire) consistera a sommer les valeurs de caractères trouvés dans le message, le tout modulo 256. Notre digest tiendra donc sur un octet.
Vous allez personnifier Mallory :

1. Prenez un message d'Alice et son digest.
2. Calculez le digest du message que vous voulez envoyer a Bob.
3. Modifier votre message pour que les deux digests soient identiques (rajoutez des caractères bien choisis)
    - Ceci fonctionne-t-il si Alice signe le digest ?
    - Quel est le problème ici ?

Il vous faut des fonctions de hachage perfectionnées !