MATH EN INFO

FICHE EXPLICATION Nº 18

Déchiffrer un morceau

1 Descriptif

L'objectif de ce défi est d'implémenter une méthode permettant de déchiffrer un morceau du message en utilisant la clé publique, la clé privée et l'algorithme de RSA. On rappelle que la clé publique de RSA est composée de deux parties N et e que nous identifierons respectivement par "cleRSA_N" et "cleRSA_e" et que la clé privée est composée d'une seule partie d que nous identifierons par "cleRSA_d". On rappelle aussi qu'un morceau M du message est déchiffré en faisant simplement M^d modulo N.

Attention, votre méthode doit renvoyer un mot binaire dont la taille doit impérativement être fixée à celle d'un morceau standard. On utilisera la méthode statique "getTailleMorceau" de la classe "ParametresRSA" pour récupérer cette valeur.

2 Protocole

1. Une fois la connexion établie, le serveur commence par envoyer un premier message annonçant le début du défi :

   -- Debut du defi : Dechiffrer un morceau --

   Ce message n'attend pas de réponse.
2. Le serveur envoie ensuite une série de triplets (M,N,d) où M est un mot binaire de 128 bits et N et d des nombres binaires.
3. Pour chaque triplet (M,N,d), le serveur doit recevoir en retour M déchiffré avec la clé (N,d) (sous forme binaire).
4. Après chaque réponse, le serveur enverra un message commençant par "OK" ou "NOK" suivant si la réponse est correcte ou non.
5. A la fin du défi, le serveur enverra un message indiquant "Defi valide" ou "Defi echoue !". Aucune réponse n'est attendue.
6. Le serveur terminera la communication par le message "FIN", votre client devra alors fermer la socket. Aucune réponse n'est attendue.

3 Exemple de communication

Voici un exemple (incomplet) d'une communication pour ce défi. Dans cet exemple les "<" et ">" indiquent le sens de transfert de chaque message et ne doivent pas être présents dans la communication.

< -- Debut du defi : Dechiffrer un morceau --
< 00000000000000000000000000000000000000010100101...
< 10000011011110100011110011101111000110011101010...
< 01011101100000100110000011110011000000011101011...
> 10100001101111110010110010101010111001101001111...
< OK
< 00000000000000000000000000000000000000000000000...
< 10010101001000000100000100101001010011100011001...
< 10000101101111011001000101000010000010010100001...
> 01111110000100010110111111111101010010010110111...
< OK
< 00000000000000000000000000000000000000000000000...
< 01101001011110100111101111110110011110111110001...
< 01001100100011011000001000111111110010001011011...
> 01110101001111100011111111001000011000011111100...
< OK