TP : RSA

Avertissement :

L'objectif du TP consiste à consolider les notions vues en cours et en TD. Il ne s'agit nullement d'un TP destiné à l'industrialisation ! Il existe des outils conçus pour réaliser des connexions sécurisées.

Il y a ainsi quelques faiblesses dans le TP. Voici les plus importantes :

- Taille des clés très faible
- Il n'y a ni le bourrage optimal, ni le réseau de Feistel asymétrique (Optimal asymmetric encryption padding : OAEP)
- Le HMAC s'appuie seulement sur une fonction de hachage pour la signature et cette fonction n'est pas sécurisée (md5). Bien que le remplacement de cette fonction soit demandé, l'échange n'est pas sécurisé.
- Le TP utilise la conversion en décimal (mauvaise manipulation de l'encodage) pour un objectif pédagogique.

En outre, l'ensemble des fonctions commencent par home pour rappeler qu'il s'agit bien d'une approche simplifiée faite maison.

Sujet :

Le TP vise à illustrer le déploiement d'un ensemble de mécanismes cryptographiques pour sécuriser l'échange entre deux parties : Alice et Bob. Le TP considère un message envoyé de Bob vers Alice, où chacun dispose d'une paire de clés (publique, privée) pour le chiffrement RSA. Bob chiffre le message avec la clé publique d'Alice. Il procède aussi à la signature de l'empreinte numérique du message avec sa clé privée. Alice reçoit le message le déchiffre et vérifie la signature de Bob.
Pour ce faire, le TP est accompagné d'un fichier RSA_B_A.py contenant les éléments donnés dans le tableau suivant :

Variables	Alice	Bob
Les deux nombres premiers	x1a et x2a	x1b et x2b
La fonction d'Euler et n	phia, na	phib , nb
Exposants publique, privé	ea, da	eb, db
Le message en clair	dechif (en string)	secret (en string), num_sec (en nombre décimal)
Le message chiffré	chif (message chiffré en nombre décimal)	chif (message chiffré en nombre décimal)
Le hash	Ahachis3 (en nombre décimal)	Bhachis3 (en nombre décimal)
Vérification de la signature	designe (en nombre décimal, déchiffré avec la publique de Bob)	signe (en nombre décimal, chiffré avec la clé privée de Bob)

Il lui manque des fonctions pour fonctionner correctement. Vous devez ainsi compléter les fonctions suivantes :

- home_mod_expnoent(x,y,n): La fonction qui permet de réaliser l'exponentiation modulaire x^y%n. Vous utilisez la fonction donnée en cours (il faut l'optimiser). Pour vérifier le résultat, vous pouvez utiliser la fonction pow(x,y,z) de python.
- home_ext_euclide(y,b) : La fonction permettant d'obtenir la clé secrète. Pour ce faire, il faut utiliser l'algorithme d'Euclide étendu.

Il faut aussi l'améliorer de la manière suivante :

- Améliorer le processus de vérification de la signature de Bob (MD5 est une fonction faible)
- Avec le changement de la fonction MD5, il est possible d'écrire des messages plus longs. Définir la nouvelle limite du message (sur la version initiale, la limite de 10 caractères).
- Modifier l'algorithme de RSA afin qu'il soit plus léger grâce au théorème du reste chinois (Voir le document CRT_RSA.pdf).

Enfin, vous remarquez que la taille du message échangé entre Alice et Bob est limitée par la taille de la clé. Afin d'étendre la taille du message et de permettre de chiffrer correctement les messages de valeurs faibles, il est possible de procéder au découpage du message par blocs et au bourrage. En suivant l'hypothèse selon laquelle Bob envoie le message m à Alice, il faut procéder de la manière suivante :

- Définir une taille limite de chaque bloc que nous notons k
- Bob : chaque bloc ne doit pas contenir plus de 50% du message que nous notons m_i et dont la taille en octets est j. Ainsi, $m = m_0 \| m_1 \| m_2 \| \dots \| m_i \| \dots \| m_n$ et $j \leq k/2$.
- Bob : Générer $k - j - 3$ octets non nuls. Bob utilise alea%255+1 pour chaque octet. Le nombre issu de cette génération est noté x.
- Bob : Constituer le bloc de la forme suivante : $00 \| 02 \| x \| 00 \| m_i \|$, avec 00, 02 sont des octets valant 0 et 2 en hexadécimal.
- Bob : Chiffrer le bloc avec RSA.
- Alice : Déchiffrer le bloc avec RSA
- Alice : Eliminer $00 \| 02 \| x \| 00$ de chaque bloc pour obtenir m_i
- Alice : Concaténer l'ensemble de m_i pour constituer le message initial.

Cette approche est inspirée de PKCS#1v1.5. Dans PKCS#1v1.5 la taille minimale de x est de 8 octets. Il est important de souligner que cette approche est vulnérable aux attaques avec un serveur qui nous informe lorsqu'il s'agit d'un mauvais bourrage (Un exemple d'attaque similaire sera vu en TD).

Format pour rendre le devoir

Le rapport de TP et le code sont à rendre individuellement. Vous déposez le rapport et le code dans la section devoir dédiée à ce TP. Seuls les codes en Pythons sont acceptés et les rapports en format pdf. Donc vous avez deux/trois fichiers à télécharger :

1. NOM_Prénom.pdf : Il s'agit du rapport
2. Nom_Prénom_main.py : Il s'agit du code en python
3. Nom_Prénom_util.py : Seulement en cas de création de classes et fonctions dans un fichier séparé

Bon courage