TP Mon RSA

Développer une application, qui permet de chiffrer & déchiffrer des messages en RSA[1]

Contexte

Vous allez coder, entièrement (c.a.d depuis une page blanche) un programme en ligne de commande qui permet de communiquer en RSA.

Pour des raisons de puissance de calcul & d'optimisation de programme, cette application se limitera à des longueurs de clé trop faible pour être réellement utile.

… mais c'est fun à coder
et surtout c'est obligatoire !

Je recommande Python, mais vous pouvez utiliser le langage de votre choix (oui, même Java)

Attention : Autant vous pouvez utiliser des bibliothèques & des fonctions pour vous aider (calcul de nombres 1er , calcul de modulo, …) autant vous ne pouvez pas utiliser des fonctions pour calculer directement des clé RSA : Elles seraient incompatible avec ce TP

Vous pourriez (au mieux) chiffrer / déchiffrer vos propres textes, mais pas ceux chiffrés par les programmes des autres

Livrable

Livrable : Un dépôt git

Si le dépôt est public : Me donner l'URL par Discord / Mail (hugues.levasseur@arrobe.fr)

Si le dépôt est privé : M'inviter avec des droits suffisants pour que je voie le code

- Gitlab (de préférence) : https://gitlab.com/arrobe
- Github (boo ! Gafam!) : https://github.com/ArrobeHugues

Le README.md doit :

- Donner le(s) nom en clair (pas les pseudos) du ou des auteurs
- Expliquer comment installer le logiciel & les éventuelles dépendances

Attention : En cas de binôme, les deux doivent obligatoirement avoir poussé des commits dans des proportions comparables (Fini les binômes « tu fait tout, et moi je met juste mon nom »)

1 … dans une version simplifiée du RSA

Rappel du fonctionnement du RSA

Génération des clé

On utilise 6 nombres entiers p, q, n, n', d et e qui respectent les règles suivantes :

- p & q sont deux nombres premiers différents
- n = pq
- n' = (p - 1) (q – 1)
- e est un nombre premier différent de d
- ed = 1 modulo n'

Concrètement :

- Générer 2 nombres 1er différents de 10 chiffres de long[2] : p & q
- Calculer n et n' (facile)
- Faire une boucle[3] qui teste des valeurs pour e et d jusqu'à ce que :
  - e soit premier
  - e soit différent de d
  - ed % n' = 1 (ou, dit autrement, ed = 1 modulo n')

Bravo, vous avez une clé privée : n et d et une clé publique : n et e

Chiffrement

Pour chiffrer une chaîne de caractères avec la clé publique n et e, il faut :

- Transformer chaque caractère en entrée en un chiffre en utilisant le code ASCII
- Assembler & redécouper cette chaîne en blocs :
  - On prends des blocs de longueur (taille de n) – 1 de long
  - On part de la droite et on complète, éventuellement le dernier avec des 0 non significatifs
  - Chaque bloc en clair B est chiffré en un bloc C par la formule $C = B^e$ modulo n
- Assembler les blocs chiffrés C en une suite de chiffres
- Transformer cette suite de chiffres en texte affichable grâce un double encodage ASCII puis Base64

2 La limite à 10 chiffres de long est, bien sur, pour ce TP pas pour le 'vrai' RSA
3 Oui, toute la difficulté est là …

Déchiffrement

Pour déchiffrer une chaîne de caractères avec la clé privée n et d, il faut :

- « défaire » les 2 encodages de la dernière étape de chiffrement : Base64 décode puis ASCII encode
- Découper la chaîne en blocs C de longueur (taille de n) – 1 de long
- Déchiffrer chaque bloc C en un bloc B avec la formule $B = C^d$ modulo n
- Découper cette nouvelle chaîne en blocs de 3 chiffes et procéder à un encodage ASCII
- Tadam !

Le programme, en détail

Les paramètres

Le programme se lancera en ligne de commande

Si le programme est lancé sans paramètres (ou avec `help` comme paramètre), il affiche un manuel

ex :

```
Script monRSA par Nug
Syntaxe :
	monRSA <commande> [<clé>] [<texte>] [switchs]
Commande :
	keygen : Génère une paire de clé
	crytp : Chiffre <texte> pour le clé publique <clé>
	decrytp: Déchiffre <texte> pour le clé privée <clé>
	help : Affiche ce manuel
Clé :
	Un fichier qui contient une clé publique monRSA ("crypt") ou une clé privée ("decrypt")
Texte :
	Une phrase en clair ("crypt") ou une phrase chiffrée ("decrypt")
Switchs :
	-f <file> permet de choisir le nom des clé générés, monRSA.pub et monRSA.priv par défaut
	...
```

Contrôlez que les paramétrés sont corrects.

Règles :

- Commande est obligatoire et vaut "keygen", "crytp", "decrypt" ou "help"
- Si "crytp" paramètre 2 (clé) et 3 (texte) obligatoire
- Si "decrytp" paramètre 2 (clé) et 3 (texte) obligatoire
- Tout les switchs son facultatifs

Keygen

Si le module keygen est appelé, vous devez générer 2 fichiers (dans le même dossier) :

- `monRSA.priv` et `monRSA.pub`

Une fois calculé les nombres `n`, `d` et `e` (voir « Génération des clé ») vous devez créer 2 fichiers texte avec le contenu suivant

Attention : Respecter exactement ce formalisme, sinon vos programmes seront incompatibles entre eux.

Pour la clé privée, la 1er ligne est « en dur » :

```
---begin monRSA private key---
```

La seconde ligne c'est :

base64_encode(decimal_vers_hexa(`n`), retour chariot, decimal_vers_hexa(`d`))

Adaptez les fonction à votre langage

la 3eme ligne est « en dur » :

```
---end monRSA key---
```

Exemple :

```
nug@xig:/var/python/mon-rsa$ cat monRSA.pub
---begin monRSA private key---
MHgxYTQxMDA2ODcyMWQ0M2Y3ZgoweDEyNDM3YWI3ZDlkMTJhMDA=
---end monRSA key---
```

Pour la clé publique :

```
---begin monRSA public key---
```

base64_encode(decimal_vers_hexa(`n`), retour chariot, decimal_vers_hexa(`e`))

```
---end monRSA key---
```

Crypt

- Lisez le fichier fourni en 2eme paramétré
- Vérifiez qu'il commence par '---begin monRSA public key ---'
- Lisez la ligne 2 et extrayez-en les valeurs `n` et `e` :
  - Base64_decode de l'ensemble

    - hexa_vers_décimal de la partie avant le retour chariot => `n`
    - hexa_vers_décimal de la partie après le retour chariot => `e`
- Traitez le 3eme paramètre selon l'algorithme expliqué au chapitre « Chiffrement »
- Affichez le cryptogramme

Decrypt

- Lisez le fichier fourni en 2eme paramétré
- Vérifiez qu'il commence par '---begin monRSA private key ---'
- Lisez la ligne 2 et extrayez-en les valeurs `n` et `d` :
    - Base64_decode de l'ensemble
    - hexa_vers_décimal de la partie avant le retour chariot => `n`
    - hexa_vers_décimal de la partie après le retour chariot => `d`
- Traitez le 3eme paramètre selon l'algorithme expliqué au chapitre « Déchiffrement »
- Affichez le texte en clair

Options (facultatif)

Vous pouvez modifier votre programme pour qu'il accepte les switchs suivants :

Filename

Ajouter un switch -f <filename> qui précise à keygen le nom des fichiers à générer (défaut monRSA)

Size

Ajouter un switch -s <size> qui précise à keygen la taille de la clé à générer (défaut 10)

Input

Crypt & decrypt acceptent un fichier texte à la place d'une chaîne (utiliser un switch -i)

Output

Crypt & decrypt acceptent un switch -o qui donne le nom d'un ficher de sortie (plutôt que d'afficher)

???

Vous pouvez, bien sur, ajouter d'autres switchs a condition qu'ils respectent la compatibilité ascendante du programme