TP MonEcc

Développer une application, qui permet de chiffrer & déchiffrer des messages en ECC

1 Bla bla

1.1 Contexte

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

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 pour les parties Hash & AES, vous pourrez utiliser des bibliothèques & des includes. Autant pour la partie ECC, vous devez les coder vous même !

1.2 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 »)

2 Rappel du fonctionnement de ECC

Pour ce TP on utilisera la courbe suivante : **$Y^2 = X^3 + 35X + 3$ (modulo 101)**

et le point de départ **P** (2, 9)

2.1 Génération des clé

- Tirer un nombre **k** aléatoire entre 1 et 1000

- Calculer **Q = kP**

Si par exemple vous avez k = 25 vous calculez Q = P + P + P + P …. + P (25 fois)

Voir méthode Double & add dans le cours)

Bravo, vous avez une clé privée : **k** et une clé publique : **Q**

2.2 Chiffrement et déchiffrement

En utilisant votre clé privée **k** et la clé publique **Q_b** de votre 'cible' : Calculez le secret partagé **S**

$S = k\ Q_b$

Hachez ce secret avec l'algorithme SHA256

Ex en Python

```
import hashlib
secret_partage = [123456, 654321] # Exemple fictif de secret S
secret_partage = hashlib.sha256(secret_partage[0])
secret_partage = hashlib.sha256(secret_partage[1])
print(f"le secret a utilser en AES : {secret_partage.hexdigest()}")
```

Ensuite chiffrez le message en AES/CBC avec le secret hashé comme clé
(on utiliser les 16 1er caractères du secret comme vecteur d'initialisation)

Ex en Python

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import padding

texte_en_clair = "tralala tsoin tsoin"
iv =  secret_partage[:16] # 16 premiers chars dans IV
cle =  secret_partage[-16:] # Le reste comme clé
padder = padding.PKCS7(128).padder()
padded_data = padder.update(texte_en_clair.encode('utf-8'))
padded_data += padder.finalize()
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
encryptor = cipher.encryptor()
ciphertext = encryptor.update(padded_data) + encryptor.finalize()
print(f"Le texte chiffré est {ciphertext}")
```

3 Le programme, en détail

3.1 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 monECC par Hugues
Syntaxe :
      monECC <commande> [<clé>] [<texte>] [switchs]
Commande :
      keygen : Génère une paire de clé
      crytp : Chiffre <texte> pour la clé publique <clé>
      decrytp: Déchiffre <texte> pour la clé privée <clé>
      help : Affiche ce manuel
Clé :
      Un fichier qui contient une clé publique monECC ("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, monECC.pub et
      monECC.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

3.2 Keygen

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

- monECC.priv
- monECC.pub

Une fois calculé le nombre k et les coordonnées de Q (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.

3.2.1 Clé privée

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

```
---begin monECC private key---
```

La seconde ligne c’est :
base64_encode(k)

la 3eme ligne est « en dur » :

```
---end monECC key---
```

Exemple :

```
nug@xig:/var/python/mon-ecc$ cat monECC.pub
---begin monECC private key---
MTAwMA==
---end monECC key---
```

3.2.2 Clé publique

La 1er ligne est « en dur » :

```
---begin monECC public key---
```

La seconde ligne c'est :
base64_encode(Qx + ";" + Qy)

on met les 2 coordonnées du point Q séparés par un ;

la 3eme ligne est « en dur » :

```
---end monECC key---
```

3.3 Crypt

- Lisez le fichier fourni en 2eme paramètre
- Vérifiez qu'il commence par '---begin monECC public key ---'
- Lisez la ligne 2 et extrayez-en les valeur Q_x et Q_y
- Traitez le 3eme paramètre selon l'algorithme expliqué au chapitre « Chiffrement »
- Affichez le cryptogramme

3.4 Crypt

- Lisez le fichier fourni en 2eme paramètre
- Vérifiez qu'il commence par '---begin monECC private key ---'
- Lisez la ligne 2 et extrayez-en la valeur k
- Traitez le 3eme paramètre selon l'algorithme expliqué au chapitre « Chiffrement »
- Affichez le texte en clair

4 Options (facultatif)

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

4.1 Filename

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

4.2 Size

Ajouter un switch -s <size> qui précise à keygen la plage d'aléa de la la clé à générer (par défaut on est entre 1 et 1.000)

4.3 Input

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

4.4 Output

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

4.5 ???

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