TD 6 Programmation et utilisation de DH

27 mai 2024

Introduction

Si ce n'est pas encore fait, installez le paquet `cryptography` de `Python` qui utilise la librairie `OpenSSL`.

Le but de ce TP est de réaliser une mise en accord par DH pour obtenir une clé maître, nommée `MasterKey`. La clé maître sera ensuite dérivée avec `pbkdf2`pour obtenir la clé de session `SessionKey` ainsi qu'une valeur initiale `IV`. La clé de session et la valeur initiale seront ensuite utilisées pour réaliser un chiffrement `AES` par blocs en mode `CBC` sur le texte clair compressé au préalable par la librairie `zlib` qui fournit un compressé au format `bytestream`. Le TP est à réaliser préférablement en binôme et les messages seront échangés sur une mssagerie instantannée entre les deux participants du binôme (Alice et Bob).

1 Mise en accord par Diffie Hellman

On s'inspire largement de la documentation. Prenez tout d'abord connaissance du code ci-dessous :

[ ]:
```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.backends import default_backend
```

[ ]:
```
# Generate some parameters. These can be reused.
parameters = dh.generate_parameters(generator=2, key_size=1024,
                                    backend=default_backend())

# Generate a private key for use in the exchange.
private_key = parameters.generate_private_key()

# In a real handshake the peer_public_key will be received from the
# other party. For this example we'll generate another private key and
# get a public key from that. Note that in a DH handshake both peers
# must agree on a common set of parameters.
peer_public_key = parameters.generate_private_key().public_key()
shared_key = private_key.exchange(peer_public_key)
```

Exercice 1 Dans le monde réel, les paramètres DH sont à récupérer et à transmettre à l'autre partie. On suppose pour la suite qu'Alice est l'initiatrice de l'échange. La méthode `dh.generate_parameters` construit l'objet `parameters` qui regroupe le grand entier premier p et g le générateur du groupe. Retrouvez ces deux entiers au moyen de leurs accesseurs `p` et `g` et de la méthode d'instance `parameter_numbers()`. Pour tester, rangez-les dans deux variables, `p` et `g`.

[ ]:

Les entiers p et g peuvent être transmis en clair sur le canal (ici dans un message `Discord`) mais doivent ensuite être regroupés dans un objet `parametres` comme décrit dans le code exemple de la documentation et recopié ci-dessous

[ ]:
```
# EXEMPLE DE RECONSTRUCTION DES PARAMETRES
p=parameters.parameter_numbers().p
g=parameters.parameter_numbers().g
pn=dh.DHParameterNumbers(p, g)
parametres = pn.parameters()
```

Exercice 2 Reconstruisez les paramètres DH de Bob et affichez-les.

[ ]:

Une fois les paramètres p et g partagés, Alice doit construire sa clé privée et sa clé publique (selon la terminologie utilisée avec le partage DH). Elle partage ensuite sa clé publique sur le canal en utilisant la méthode d'instance `public_numbers()` et l'accesseur `y`.

Exercice 3 Ecrivez une fonction `UserGen` qui prend en entrée p, g, et qui engendre la clé privée de l'utilisateur et la valeur y qui devra être transmise.

[ ]:
```
def UserGen(p,g):
```

Exercice 4 Utilisez la fonction `UserGen` pour engendrer la clé privée d'Alice et la valeur y qu'elle doit transmettre à Bob.

[ ]:

Exercice 5 Utilisez la fonction `UserGen` pour engendrer la clé privée de Bob et la valeur y qu'il doit transmettre à Alice.

[ ]:

Exercice 6 retrouvez la clé partagée par Alice et Bob en faisant les calculs du côté d'Alice et du côté de Bob. Ecrivez une fonction `CalculMaster` qui prend en entrée p, g, le message reçu y et la clé privée. Appliquez cette fonction pour le calcul de la clé maître du côté d'Alice puis du côté de Bob et comparez les résultats obtenus. **Attention !** Il faut reconstruire un objet `dh.DHPublicNumbers` à partir de la valeur y reçue et de l'objet paramètres DH reconstruit à partir de p et g puis d'appliquer la méthode `public_key()`.

[ ]:
```
def CalculMaster(p,g,y,sk):
```

Exercice 6.1 Alice calcule la clé maître `MasterKey` à partir du message reçu de Bob et de sa clé privée :

[ ]:

Exercice 6.2 Bob calcule la clé maître `MasterKey` à partir du message reçu d'Alice et de sa clé privée :

[ ]:

2 Dérivation de clé

Alice et Bob partagent maintenant `MasterKey`. Il utilisent ensuite un algorithme de dérivation de clé pour construire : - la clé AES de 256 bit (ou 32 octets) ; - une valeur initiale de 128 bits (ou 16 octets).

Exercice 7 On s'inspire de la documentation pour écrire une fonction `derive` qui prend en entrée un secret initial de 256 bits, 128 bits de sel, la taille de la clé dérivée (en bits) et qui retourne une clé dérivée du secret initial de la taille souhaitée. On extraira de la clé maître les 256 premiers bits pour obtenir le secret initial, les 128 bits suivants pour le sel.

[ ]:
```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

def derive(mdp,sel,bits):
```

Exercice 8.1 Dérivez `MasterKey` du côté d'Alice et du côté de Bob pour obtenir le clé de session `SessionKey` de 256 bits.

[ ]:
```

```

Exercice 8.2 Dérivez `MasterKey` du côté d'Alice et de Bob pour obtenir la valeur initiale `IV` de 128 bits.

[ ]:
```

```

On peut vérifier qu'il y a bien égalité entre les valeurs dérivées :

A présent, Alice et Bob disposent des même paramètres et peuvent enfin échanger des messages chiffrés par `AES-256` en mode `CBC` en utilisant `SessionKey`et `IV`.

3 Compression des clairs

Exercice 9. En utilisant la librairie `zlib` écrivez les fonctions `compresse` et `decompresse` pour compresser et décompresser une chaîne de caractères `utf-8` convertie en `bytestram` et avec comme type de sortie des `bytestream`.

[33]:
```
import zlib

def compresse(texte):

def decompresse(comprime):
```

4 Chiffrement des messages

Exercice 10 En vous inspirant du TD 2, écrivez les fonctions `chiffre()` qui prend en entrée le texte clair (au format standard), la clé de session et l'IV et qui et qui va effectuer la compression du texte, le chiffrer par `AES-256-CBC` et retourner le chiffré au format base 64. La fonction `dechiffre()` prend en entrée le chiffré au format base 64, une clé et l'IV et retourne le texte clair (au format standard).

Classe pour utiliser AES256 avec une clé et une iv

[ ]:
```
import base64

def chiffre(texte,cle,iv):

def dechiffre(cryptogramme,cle,iv):
```

A présent vous êtes en mesure d'échanger entre vous des messages chiffrés sur `Discord`