Cryptographie - Authentifiction - 150 points

Kévin DUVERGER

Table des matières

1 Résolution Authentifiction :

La première chose à faire est d'intéragir avec le serveur pour voir ce qu'il demande :

```
Welcome to the NSA's authentication service.
Please choose what you want to do :
        [1] Observe an authentication
        [2] Authenticate
Your choice :
```

Comme vous pouvez le voir on peut soit observer un utilisateur qui s'authentifie ou s'authentifier nous-mêmes.

Testons donc ce qu'il se passe dans le premier cas :

```
Step 1 : the server sends a 16-byte random challenge :
        challenge = 686b2a2a3e84ce7a63d282853fcaacd7

Step 2 : the client sends its authentication as a tuple (IV, C) :
        IV : a random 16 bytes value, here IV = dabfa6cdc5ea21eb3d493012c9323781
        C  : a ciphertext AES-CBC-Encrypt(K, IV, challenge | "admin" | 000000000000 | "<END>"), here
     C = 1f8d77e2241815cd230cc137e4882068de2a250193985734f5e67e24efb5a2b9

Step 3 : the server decrypts with AES-CBC-Decrypt(K, C) and does some checks on the plaintext :
        The first 16 bytes are equal to the challenge sent
        The last 5 bytes are "<END>"
```

Nous pouvons donc voir que pour authentifier un utilisateur, le serveur comment par envoyer un challenge complètement aléatoire, puis la personne qui s'authentifie doit créer un chiffré contenant de challenge et en utilisant une clé K partagée au préalable avec le serveur, et pour finir le serveur vérifie que le challenge est bon et que les 5 derniers octets forment le mot `<END>`.

Et dans le second cas, le serveur nous envoie un challenge en attendant une réponse :

```
Here is your challenge : 917be91060b0ee7acffc937ff3f3f766
Now, go fast because you have only 2 seconds to give the IV and then the ciphertext !
Now please give your IV     :
```

C'est ici qu'il faudra que nous attaquions pour arriver à nous authentifier. Dans toute la suite, le challenge pour lequel on a pu observer la communication s'appellera ch_1 et celui pour lequel on doit répondre ch_2.

Nous n'avons aucun espoir de trouver la clé K, il n'y a pas de collisions sur les valeurs des challenges, donc on pourrait croire qu'il nous sera impossible de renvoyer le bon chiffré pour le nouveau challenge ch_2. On peut quand même tenter quelque chose : **modifier le chiffré que l'on a vu pour une authentification pour qu'il se déchiffre sur le nouveau challenge reçu**, et l'idée qui rend cela possible est que le schéma de chiffrement utilise le mode CBC.
Expliquons tout d'abord comment fonctionne le mode CBC pour le message que l'on reçoit et concentrons nous sur le premier bloc qui correspond au challenge. Le chiffré de ce premier bloc est obtenu en calculant $C_1 = \mathsf{Enc}(K, ch_1 \oplus IV)$, et le déchiffrement s'obtient en calculant $\mathsf{Dec}(K, C_1') \oplus IV'$: la question est maintenant de savoir ce que l'on peut mettre pour C_1' et IV' pour que le déchiffrement donne ch_2.
La seule valeur dont on connaît le déchiffrement est C_1, c'est donc la seule valeur que l'on peut prendre pour C_1' (car toute autre valeur que l'on prendrait ferait que $\mathsf{Dec}(K, C)$ soit inconnu). La première étape du déchiffrement CBC va donc donner $ch_1 \oplus IV \oplus IV'$ dont on voudrait pour rappel que ce soit égal à ch_2. Nous pouvons maintenant finir en prenant $IV' = ch_1 \oplus IV \oplus ch_2$, ce qui combiné avec $C_1' = C_1$ nous donnera bien le déchiffrement sur ch_2 !

Nous n'avons résolu qu'une partie du problème, il faut maintenant s'intéresser à la seconde partie du chiffré qui contient le nom de l'utilisateur, une suite de 0 ainsi qu'un marqueur de fin. Notez que nous n'avons pas modifié le premier bloc de chiffré : nous pouvons tout simplement prendre $C_2' = C_2$ ce qui se déchiffrera bien sur la même chose qu'avant !

Voici donc le code de cette solution :

```python
import random
import hashlib
import os, sys, socket

def xorBytes(bytes1, bytes2) :
    result = b""
    for i in range(0, len(bytes1)) :
        result += (bytes1[i] ^ bytes2[i]).to_bytes(1, byteorder="big")
    return result

server = "146.59.227.136"
port = 23403
```

```python

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# Getting the start screen
ligne = ma_socket.recv(4096).decode("utf-8")
print(ligne)
ma_socket.sendall(b"1\n")

# Getting the data
data = ma_socket.recv(4096).decode("utf-8").split("\n")
challenge = bytes.fromhex(data[1][13:])
IV = bytes.fromhex(data[4][41:])
ciphertext = bytes.fromhex(data[5][98:])
print(challenge.hex(), IV.hex(), ciphertext.hex())

ma_socket.close()



ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# Getting the start screen
ligne = ma_socket.recv(4096).decode("utf-8")
print(ligne)
ma_socket.sendall(b"2\n")

# Sending the IV
data = ma_socket.recv(4096).decode("utf-8").split("\n")
finalChallenge = bytes.fromhex(data[0][25:])
print(finalChallenge.hex())
finalIV = xorBytes(xorBytes(challenge, finalChallenge), IV)
ma_socket.sendall((finalIV.hex() + "\n").encode("utf-8"))
# Then the ciphertext
data = ma_socket.recv(4096).decode("utf-8")
finalCT = ciphertext[0:16] + ciphertext[16:32]
ma_socket.sendall((finalCT.hex() + "\n").encode("utf-8"))
# And the flag
data = ma_socket.recv(4096).decode("utf-8")
print("Flag : " + data)

ma_socket.close()
```