TP4 — Pratiques

Ex1. Un air de déjà vu

Nous avons intercepté l'échange suivant. Visiblement il y a du CTR dans l'air, et on fonctionne avec un mini-AES et des blocs de 16 bits. On sait de source sûre que le premier message de Yoh est : "Mat Mat!". On sait aussi que le second message clair de Yoh et le premier message clair de Mathieu sont identiques à 100%.

(a) Détaillez précisément la méthodologie qui vous permettra de déchiffrer l'intégralité de cette discussion.

(b) Déchiffrez avec Python l'intégralité de la discussion. Ne soyez pas surpris par une incohérence apparente des messages déchiffrés.

(a) `Yoh: 24ae 4d8a 64cd b5f0 bcb1`

(b) `Yoh: 24ae 31d9 68dc cbae`

(c) `Mat: bf24 3e86 f781 dc8a`

(d) `Mat: bf24 3294 bb82`

(e) `Yoh: 24ae 4d8e 628e 91b0`

(c) Normalement, vous avez désormais toutes les billes en main pour envoyer un message à Yoh, en vous faisant passer pour Mat. Avec la configuration CTR de Mat et avec Python, chiffrez les 4 octets du message suivant : "Top!"

Ex2. Man-in-the-middle – El-Gamal L'exercice avec lequel vous allez vous amuser (au premier degré) est un défi proposé aux étudiants il y a deux ans. Grâce au jeu (archive elgamal.zip dans laquelle il faudra ouvrir elgamal.html) vous allez devoir mener une attaque dite *man-in-the-middle*. Une attaque connue contre Diffie-Hellman est décrite ci-dessous. Le contexte est le suivant : A et B sont deux agents honnêtes alors que Y pas du tout (mais A et B ne le savent pas). A veut communiquer avec B mais tous les messages passent sur le réseau. Y, tout puissant qu'il est, peut intercepter les messages et en envoyer d'autres. Les paramètres g et n sont publiques.

1. $A \rightarrow Y(B) : A,\ B,\ g^{na} \bmod n$ où $Y(B)$ signifie que Y intercepte le message à destination de B
2. $Y \rightarrow B : A, B, g^{ny} \bmod n$
3. $B \rightarrow Y(A) : B,\ A,\ g^{nb} \bmod n$
4. $Y \rightarrow A : B,\ A,\ g^{ny} \bmod n$

En résumé, la clé calculée par A est $(g^{ny})^{na} \bmod n$ et il pense la partager avec B. La clé calculée par B est $(g^{ny})^{nb} \bmod n$ et il pense la partager avec A. Et enfin, Y peut calculer les deux clés car il connaît :

1. $g^{na} \bmod n$ intercepté à l'étape 1 ;
2. $g^{nb} \bmod n$ intercepté à l'étape 3 ;
3. ny le nombre aléatoire qu'Y a généré.

En s'inspirant de cette approche, essayez de découvrir le message secret que Bob va envoyer à Alice dans ce jeu en sachant que le chiffrement utilisé sera **El-Gamal**.
Histoire de ne pas partir de rien, voici une fonction très utile pour calculer l'inverse modulaire d'un entier pour une base donnée.

```python
def euclide(a,b):
  assert a>b
  u,v,uu,vv=1,0,0,1
  while b>0:
    (q,r)=divmod(a,b)
    a,u,v,b,uu,vv=b,uu,vv,r,u-q*uu,v-q*vv
  return (a,u,v)

def invmod(x,n):
  assert x<n
  return euclide(n,x)[2]%n

```

Attention : pensez à lancer la console de développement dans le navigateur, il est probable que des choses intéressantes puissent apparaître.

Ex3. Diffie-Hellman

Cet exercice est issu d'un sprint des années précédentes. Le code Python ci-dessous n'est pas exécutable, et en aucun cas il vous sera demandé de l'exécuter. Cependant, il décrit de manière très précise le protocole de communication qu'utilisent Alice et Bob pour communiquer. Visiblement, ils utilisent Diffie-Hellman pour établir un secret partagé. A partir de ce secret partagé, ils dérivent deux clés secrètes : une (kab) lorsque les communications vont d'Alice vers Bob et l'autre (kba) quand les communications vont de Bob vers Alice.

```python
from Crypto.Cipher import AES
from Crypto.Hash import HMAC, SHA256
from Crypto.Random.random import randint

g = 2
n = 999959
kab = None
kba = None

def envoie(s, qui, quoi):
    s.send("{}: {}\n".format(qui, quoi.hex()).encode())

def recoit(s):
    data = s.recv()
    st = data.index(b" ")
    return bytes.fromhex(data[st + 1 : -1].decode())


def start_alice(s):
    "invoque avant toute operation chez alice"
    global kab, kba
    # DH
    a = randint(n // 2, n)
    x = pow(g, a, n)
    envoie(s, "alice", x.to_bytes(4, "big"))
    y = int.from_bytes(recoit(s), "big")
    # secret partage
```

```
28      k = pow(y, a, n).to_bytes(4, "big")
29      # derivation de cles
30      h = HMAC.new(k, digestmod=SHA256)
31      h.update(b"alice->bob")
32      kab = h.digest()
33      h = HMAC.new(k, digestmod=SHA256)
34      h.update(b"bob->alice")
35      kba = h.digest()
36
37    def send_alice(s, data):
38      "invoque pour envoyer un message cote alice"
39      cipher = AES.new(kab, AES.MODE_GCM, mac_len=16)
40      assert len(cipher.nonce) == 16
41      c, h = cipher.encrypt_and_digest(data)
42      msg = cipher.nonce + c + h
43      envoie(s, "alice", msg)
44
45
46  def recv_alice(s):
47      "invoque pour recevoir un message cote alice"
48      msg = recoit(s)
49      nonce = msg[:16]
50      c = msg[16:-16]
51      h = msg[-16:]
52      cipher = AES.new(kba, AES.MODE_GCM, mac_len=16, nonce=nonce)
53      data = cipher.decrypt_and_verify(c, h)
54      return data
55
56
57  def start_bob(s):
58      "invoque avant toute operation chez bob"
59      global kab, kba
60      # DH
61      b = randint(n // 2, n)
62      y = pow(g, b, n)
63      envoie(s, "bob", y.to_bytes(4, "big"))
64      x = int.from_bytes(recoit(s), "big")
65      # secret partagé
66      k = pow(x, b, n).to_bytes(4, "big")
67      # dérivation de clés
68      h = HMAC.new(k, digestmod=SHA256)
69      h.update(b"alice->bob")
70      kab = h.digest()
71      h = HMAC.new(k, digestmod=SHA256)
72      h.update(b"bob->alice")
73      kba = h.digest()
74
75
76  def send_bob(s, data):
77      "invoque pour envoyer un message cote bob"
78      cipher = AES.new(kba, AES.MODE_GCM, mac_len=16)
79      assert len(cipher.nonce) == 16
```

```
    c, h = cipher.encrypt_and_digest(data)
    msg = cipher.nonce + c + h
    envoie(s, "bob", msg)


def recv_bob(s):
    "invoque pour recevoir un message cote bob"
    msg = recoit(s)
    nonce = msg[:16]
    c = msg[16:-16]
    h = msg[-16:]
    cipher = AES.new(kab, AES.MODE_GCM, mac_len=16, nonce=nonce)
    data = cipher.decrypt_and_verify(c, h)
    return data
```

Vous retrouverez dans le fichier dicussionalicebob.txt les échanges chiffrés en ayant respecté le protocole décrit ci-dessus.
A partir de l'étape 3 (le second message envoyé par Alice), nous sommes sur le chiffrement symétrique avec les clés dérivées de l'étape Diffie-Hellman. En mode GCM, pour chaque envoi, les 16 premiers octets correspondent au nombre aléatoire (nonce) utilisé par le mode GCM (comme CTR). les 16 derniers octets correspondent au MAC (pour vérifier que le message n'est pas altéré). Tous les octets compris entre les deux correspondent au message à déchiffrer.
Questions

1. Identifiez la faiblesse de la configuration utilisée pour Diffie-Hellman.
2. Calculez l'entier k à partir duquel les deux clés sont dérivées.
3. Calculez les deux clés dérivées.
4. Déchiffrez la discussion ci-dessus.