Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Réseaux avancés II

Projet (crypto)

Réalisation d'une attaque « bruteforce *» pour l'accès à un réseau WiFi protégé en WPA-PSK*

Le but de ce projet est de réaliser un outil permettant de trouver par une méthode « brute force » le mot de passe utilisé dans la protection d'un réseau WiFi protégé par WPA-PSK.

Avertissement

Ce projet est à but pédagogique et ne saurait être utilisé sur un réseau WiFi dont vous n'auriez pas l'autorisation préalable d'accès. Les informations que vous trouverez de manière automatique, sur la configuration du réseau, devront rester confidentielles.

Présentation de la méthode de protection WPA-PSK

Le WPA,« *WiFi Protected Access* », est, comme son nom l'indique, un mécanisme de protection de l'accès aux réseaux WiFi destiné à remplacer le WEP qui ne présente pas un niveau de sécurité suffisant (le « cassage » de la protection WEP peut être réalisé en quelques minutes sur un ordinateur personnel).

Il existe deux versions :

- **WPA** : version transitoire vers la future norme 802.11i, conçue pour être utilisable sur tout matériel WiFi. Cette version supporte :
  - TKIP, « *Temporal Key Integrity Protocol* », qui est un protocole visant à corriger les failles découvertes dans WEP, basé sur l'utilisation de la méthode du « rekeying », c-à-d la génération d'une nouvelle clé pour le chiffrement de chaque paquet :
    - afin d'être compatible avec les matériels existants, il utilise toujours l'algorithme RC4 comme méthode de chiffrement ;
    - il utilise une fonction de combinaison de la clé secrète principale, « *root key* », avec le vecteur d'initialisation comme entrée du protocole de chiffrement RC4, contrairement au WEP qui utilise une simple concaténation de cette clé et du vecteur d'initialisation (faiblesse utilisée dans la plupart des attaques sur le WEP visant RC4) ;
    - il utilise un compteur de séquence permettant d'éviter les attaques par rejeu ;
    - enfin, il utilise un code de vérification d'intégrité sur 8 octets, 64 bits, appelé MIC ou « *Message Integrity Code* » pour éviter que des paquets « forgés », c-à-d construit par un attaquant, ne puissent être acceptés (ce qui était possible avec le WEP sur des paquets chiffrés mais dont le contenu était connu, par exemple pour le protocole ARP).

    Des failles ont été trouvées dans TKIP.

- **WPA2** : version compatible avec la norme 802.11i, mais intégrée uniquement sur du matériel récent :
  - elle utilise l'algorithme de chiffrement CCMP, « *Counter Mode with Cipher Block Chaining Message Authentication Code Protocol* », basé sur AES.
  - elle est disponible en deux versions :
    - la version « personal » qui utilise un mécanisme d'authentification basée sur le partage d'une clé secréte, PSK, « *Pre-Shared Key* », pour l'authentification mutuelle de la station, ordinateur voulant participer au réseau WiFi, et le point d'accès mettant en place ce réseau ;
    - la version « enterprise » : utilisant EAP, « *Extensible Authentication Protocol* », pour réaliser cette authentification mutuelle en plus d'une authentification de l'utilisateur de la station avec un serveur externe d'authentification(un serveur RADIUS par exemple).
      Cette authentification externe peut utiliser différents mécanismes comme EAP-TLS, EAP-TTLS, PEAP-TLS, *etc* avec la norme de transport 802.1X.

Présentation de la norme 802.1X

802.1X définie la méthode d'encapsulation du protocole EAP, dans un réseau basé sur 802 (« 802.3 » pour Ethernet, « 802.11 » pour WiFi *etc*), c'est pourquoi on l'appelle également EAPOL, « *EAP Over LAN* ».

Un paquet EPAOL est :
- encapsulé dans une trame Ethernet (qui peut être, elle-même, encapsulée dans une trame WiFi, 802.11) ;
- identifié dans la trame Ethernet qui le contient par le type, ou « ethertype », `888E` (le paquet EAPOL ne repose donc pas sur IP de type `0800`) ;
- chargé de contenir les paramètres (clés, nonces, *etc*) de l'authentification.

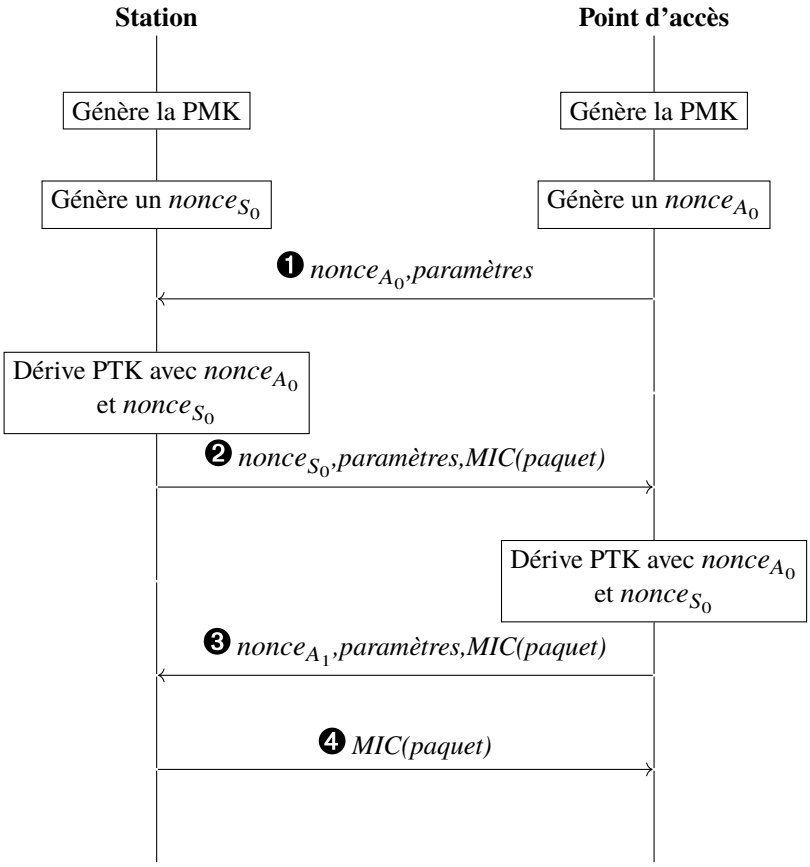
Les paquets EAPOL sont manipulables à l'aide de la bibliothèque Scapy.

Déroulement des échanges WPA-PSK

L'authentification mutuelle entre la station, S, et le point d'accès, A, se fait avec l'utilisation de la clé PMK, « *Pairwise Master Key* », avec la formule suivante $PMK = PBKDF2(PSK, SSID, 4096)$ exploitant des données connues : la PSK et l'identifiant du point d'accès (créé une clé de 256bits).

Pour ensuite assurer la sécurité des communications, il faut dériver la PTK, « *Pairwise Transient Key* », à l'aide de deux valeurs utilisées une seule fois, *nonce*, « *Number used ONCE* », créées respectivement par la station et le point d'accès.
Ces nonces vont être échangés en clair dans deux paquets EAPOL différents.

Ces échanges, on parle de « handshake » en 4 étapes, sont les suivants :



On remarquera que :
- *un MIC est échangé pour différents paquets ;*
- *ce MIC est calculé avec les valeurs des deux $nonce_{A_i}$ et $nonce_{S_j}$ qui le précèdent.*

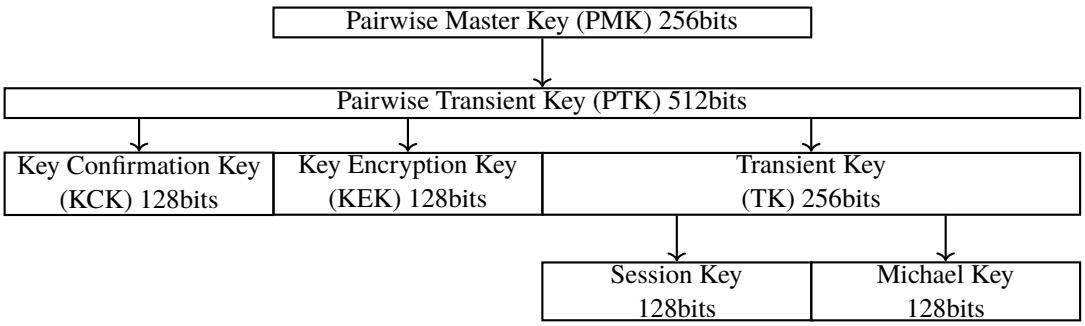
Dérivation des différentes clés

À partir de la PMK, on dérive la PTK à l'aide de la fonction suivante :
$PTK = PRF{-}512(PMK, \text{"Pairwise key expansion"}, LowerMAC \| HigherMAC \| LowerNonce \| HigherNonce)$
où ‖ désigne la concaténation.

Une PRF, « *Pseudo Random Function* », est une fonction utilisée pour agrandir une clé et une graine, « *seed* », en une séquence pseudo-aléatoire de taille variable (128, 256, 384 ou 512 bits).

Pour pouvoir sélectionner le plus petit des nonces et la plus petite des adresses MAC, il est nécessaire de les convertir en valeur numérique afin de les comparer.
La chaîne « Pairwise key expansion » est une chaîne spécifique à l'application comme référencée plus loin dans la partie sur les fonctions cryptographiques.

Une fois la PTK déterminée, on obtient différentes clés :



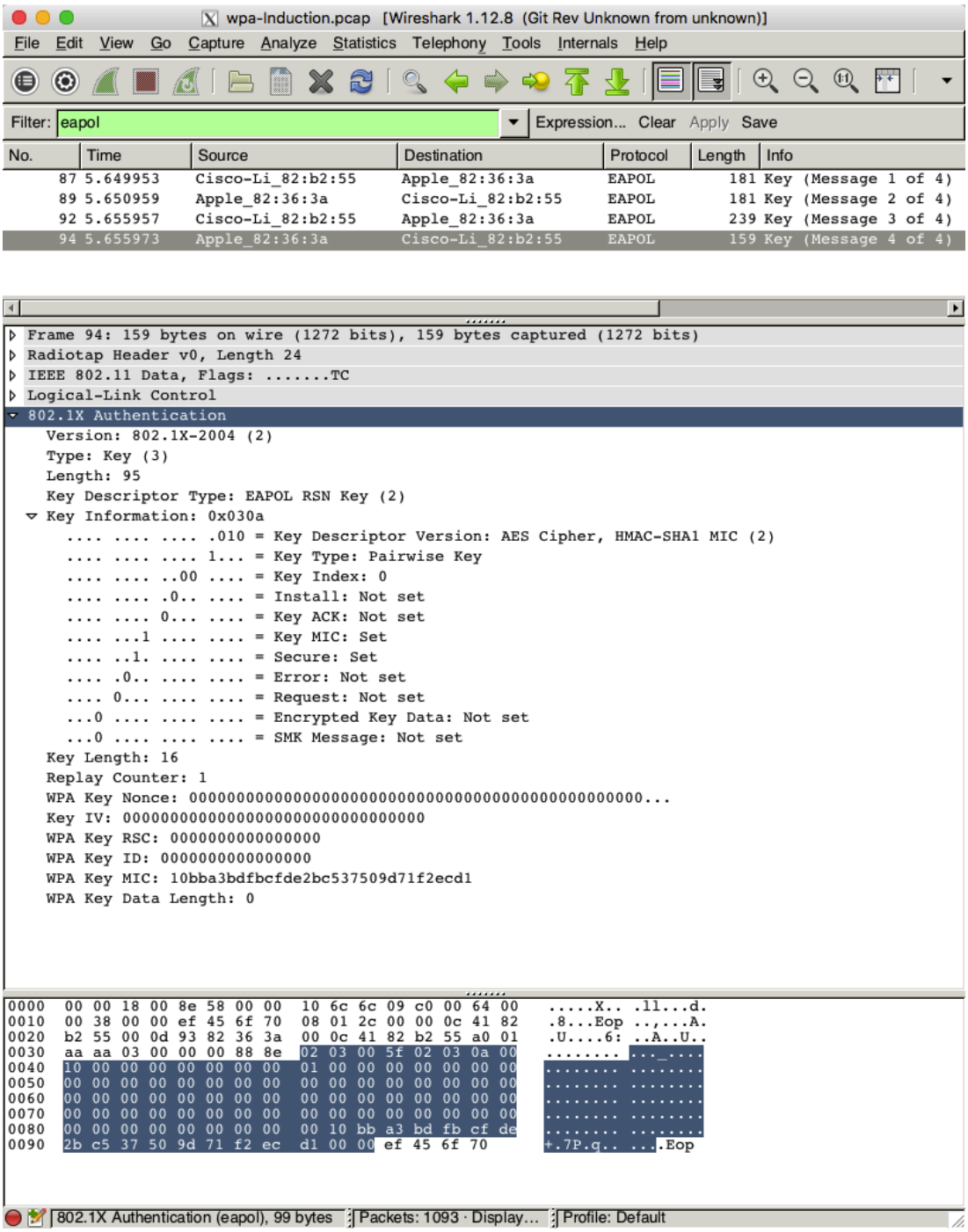
- la KEK est utilisée pour chiffrer la GTK, « *Group Transient Key* », lors de son envoi à la station. Cette GTK est générée par le point d'accès pour chiffrer les communications globales à plusieurs stations ;
- la TK est utilisée comme clé de chiffrement des communications entre le point d'accès et la station ;
- la KCK est utilisée pour calculer un MIC, « *Message Integrity Code* », qui est utilisé pour vérifier que le contenu du paquet EAP n'a pas été modifié.

Les 4 paquets échangés contiennent différentes informations :

❶	de $A \rightarrow S$	les paramètres indiquent la version de WPA (1 ou 2), le $nonce_A$ transmis permet à la station de calculer la PTK.
❷	de $S \rightarrow A$	les paramètres indiquent les capacités de la station (support chiffrement et authentification), le $nonce_S$ permet au point d'accès de calculer la PTK. Le paquet EAP est protégé des modifications par un MIC, calculé à partir de la KCK déduite de la PTK.
❸	de $A \rightarrow S$	les paramètres contiennent la GTK chiffrée avec la KEK, un MIC protège le paquet EAP.
❹	de $S \rightarrow A$	un paquet EAP presque vide confirme la bonne réception de la part de la station, ce paquet est protégé par un MIC.

Le quatrième paquet semble être le plus intéressant à traiter, il faut juste faire attention à utiliser les $nonce_A$ et $nonce_S$ qui le précèdent, puisque ceux-ci peuvent être changés et servent à déterminer la KCK.

Le quatrième paquet du « handshake » :



On remarque que :

- la clé PMK est déterminée en fonction de la PSK et du SSID (connu) ;
- la clé PTK est déterminée à partir de deux *nonces* échangés en clair, des adresses MAC de la station et du point d'accès (connues) ;
- dans le quatrième paquet échangé, on dispose d'un MIC calculé sur un paquet de contenu connu à l'aide de la fonction $HMAC_{SHA1}$ pour AES et $HMAC_{MD5}$ pour TKIP.

Si on récupère les paquets EAP, 1, 2 et 4, il est possible de mener une attaque « *brute-force* » sur le PSK :

a. on génère une combinaison de caractères pour la PSK ;
b. on dérive la PTK et la KCK ;
c. on calcule le MIC sur un paquet identique au paquet 4 (mais ne le contenant pas) ;
d. on compare si le MIC qu'on a calculé est identique au MIC contenu dans le paquet 4, si c'est le cas, alors on a découvert la PSK !
e. sinon on recommence avec une autre combinaison pour la PSK.

Travail à réaliser

Vous programmerez un outil d'attaque « *brute force* », générant différentes combinaisons de caractères (comme vu dans l'UE Sécurité des Usages TIC) et vous essaierez de trouver la PSK d'un réseau protégé par WPA-PSK dont une capture d'un handshake est disponible sur la page de l'UE Réseaux avancés II sur `http://p-fb.net/`.

Quelques indications :

- ▷ lors de la configuration du point d'accès, vous savez que l'administrateur a tapé 4 caractères « a », suivis de 4 que vous n'avez pas vu, soient 8 caractères en tout ;
- ▷ vous soupçonnez que l'administrateur n'a utilisé que des caractères alphabétiques en minuscule.

Trois jeux de tests

```
La passphrase :   radiustest
Le SSID :         linksys54gh

MAC station :
         000c 41d2 94fb
MAC AP :
         000d 3a26 10fb

La PMK :
   9e99 88bd e2cb a743 95c0 289f fda0 7bc4
   1ffa 889a 3309 237a 2240 c934 bcdc 7ddb

Le nonce AP :
   893e e551 2145 57ff f3c0 76ac 9779 15a2
   0607 2703 8e9b ea9b 6619 a5ba b40f 89c1

Le nonce station :
   dabd c104 d457 411a ee33 8c00 fa8a 1f32
   abfc 6cfb 7943 60ad ce3a fb5d 159a 51f6

La PTK:
   ccbf 97a8 2b5c 51a4 4325 a77e 9bc5 7050
   daec 5438 430f 00eb 893d 84d8 b4b4 b5e8
   19f4 dce0 cc5f 2166 e94f db3e af68 eb76
   80f4 e264 6e6d 9e36 260d 89ff bf24 ee7e

La trame EAPOL numéro 4 sans le MIC :
   0103 005f fe01 0900 0000 0000 0000 0000
   1400 0000 0000 0000 0000 0000 0000 0000
   0000 0000 0000 0000 0000 0000 0000 0000
   0000 0000 0000 0000 0000 0000 0000 0000
   0000 0000 0000 0000 0000 0000 0000 0000
   0000 0000 0000 0000 0000 0000 0000 0000
   0000 00

Le MIC :
   d0ca 4f2a 783c 4345 b0c0 0a12 ecc1 5f77
```

```
ESSID is: soho-psk
AA is:
         0020 a64f 31e4

SPA is:
         000c 41da f2e7

snonce is:
         ed12 afbd a8c5 8305 0032 e5b5 2953 82d2
         7956 fd58 4a63 43ba fe49 135f 2695 2a0f


anonce is:
         477b a8dc 6d7e 80d0 1a30 9d35 891d 868e
         b82b cc3b 5d52 b5a9 a42c 4cb7 fd34 3a64


keymic is:
         f3a0 f691 4e28 a2df 1030 61a4 1ee8 3878


eapolframe is:
         0103 005f fe01 0900 0000 0000 0000 0000
         0200 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000
         00f3 a0f6 914e 28a2 df10 3061 a41e e838
         7800 00

Starting dictionary attack.  Please be patient.
Testing passphrase: secretsecret
Calculating PMK for "secretsecret".
Calculating PTK with collected data and PMK.
Calculating hmac-MD5 Key MIC for this frame.

The PSK is "secretsecret".
```

Le paquet indiqué à la fin peut être généré par le code Python suivant à partir de la séquence d'octets :

```
mon_paquet = EAPOL(bytes.fromhex('0103005ffe01090000000000000000000200000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000f3a0f6914e28a2df103061a41ee838780000'))
mon_paquet.key_ACK = 0
mon_paquet.wpa_key_mic = ''
```

Vous pouvez récupérer le fichier `wpa-induction.pcap`, sur le Wiki de WireShark, `http://wiki.wireshark.org/`, contenant un « handshake » WPA :

```
xterm
$ wget -O wpa-Induction.pcap "http://wiki.wireshark.org/SampleCaptures?
  action=AttachFile&do=get&target=wpa-Induction.pcap"
```

Le SSID est « `Coherer` » et la PSK est « `Induction` ».

Disposer des fonctions cryptographiques nécessaires

Pour pouvoir utiliser la fonction PBKDF2, « Password-Based Key Derivation Function 2 » définie dans le Public-Key Cryptography Standards (PKCS) #5 v2.0, il vous faut récupérer la bibliothèque de Dwayne C. Litzenberger, disponible à l'URL suivante : `https://raw.githubusercontent.com/dlitz/python-pbkdf2/master/pbkdf2.py`.

xterm
```
$ wget --no-check-certificate
  "https://raw.githubusercontent.com/dlitz/python-pbkdf2/master/pbkdf2.py"
```

Pour l'utiliser ensuite dans un programme Python :

```
1 from pbkdf2 import PBKDF2
2
3 f=PBKDF2(passphrase,essid,4096)
4 pmk=f.read(32)
```

Pour implémenter la fonction $PRF_n(K, A, B)$, « *Pseudo-Random Function* », où n peut valoir 128, 256, 384 ou 512 :

1. initialiser un compteur, i, sur un octet à 0 ;
2. créer un bloc de données en concaténant les données suivantes :
   - A une chaîne de caractères spécifique à l'application ;
   - 0 un octet valant zéro ;
   - B les données à traiter ;
   - le compteur i ;
   - ce qui s'écrit : $A|0|B|i$
3. calculer $r = HMAC_{SHA1}(K, A|0|B|i)$;
4. mémoriser r dans une chaine R ;
5. recommencer la génération (étape 3) autant de fois que nécessaire pour générer le nombre n de bits aléatoires demandés.
   Avant chaque itération, il est nécessaire d'incrémenter le compteur i et après chaque itération, il est nécessaire de concaténer la nouvelle valeur de r dans la chaîne R.
6. dans le cas où l'on produit plus de bits que nécessaires en ne conserve que ceux nécessaires.
   Par exemple, en utilisant SHA-1 qui donne en sortie 20 octets, il suffit de faire 4 itérations pour disposer de 80 octets et de n'en conserver que 64 pour disposer des 512 bits voulus.

Pour utiliser la fonction $HMAC_{SHA1}$ sous Python :

```
1 import hmac, hashlib
2
3 mon_hashmac = hmac.new(cle,digestmod=hashlib.sha1)
4 mon_hashmac.update(chaine)
5 val_hmac = mon_hashmac.digest()
```

Pour utiliser la fonction $HMAC_{MD5}$ sous Python :

```
1 import hmac, hashlib
2
3 mon_hashmac = hmac.new(cle,digestmod=hashlib.md5)
4 mon_hashmac.update(chaine)
5 val_hmac = mon_hashmac.digest()
```

Intégration du format d'échange WPA-PSK dans Scapy

La bibliothèque Scapy peut interpréter les paquets échangés lors du « handshake » d'authentification entre la station et le point d'accès au format EAPOL.
Par contre, Scapy n'intègre pas la manipulation des données nécessaires à WPA-PSK. Il est nécessaire d'ajouter le support de ces données dans Scapy à l'aide du code suivant :

```
#! /usr/bin/python3

from scapy.all import *

class WPA_key(Packet):
  name = "WPA_key"
  fields_desc = [ ByteField("descriptor_type", 1),
       BitField("SMK_message",0,3),
       BitField("encrypted_key_data",0,1),
       BitField("request",0,1),
       BitField("error",0,1),
       BitField("secure",0,1),
       BitEnumField("key_MIC",0,1,{0:'not present',1:'present'}),
       BitField("key_ACK",1,1),
       BitField("install",0,1),
       BitField("key_index",0,2),
       BitEnumField("key_type",1,1,{0:'Group Key',1:'Pairwise Key'}),
       BitEnumField("key_descriptor_Version",2,3,{1:'HMAC-MD5 MIC',
                2:'HMAC-SHA1 MIC'}),
       LenField("len", None, "H"),
       StrFixedLenField("replay_counter", "", 8),
       StrFixedLenField("nonce", "", 32),
       StrFixedLenField("key_iv", "", 16),
       StrFixedLenField("wpa_key_rsc", "", 8),
       StrFixedLenField("wpa_key_id", "", 8),
       StrFixedLenField("wpa_key_mic", "", 16),
       LenField("wpa_key_length", None, "H"),
       StrLenField("wpa_key", "", length_from=lambda pkt:pkt.wpa_key_length)]
  def extract_padding(self, s):
      l = self.len
      return s[:l],s[l:]
  def hashret(self):
      return chr(self.type)+self.payload.hashret()
  def answers(self, other):
      if isinstance(other,WPA_key):
          return 1
      return 0

bind_layers( EAPOL, WPA_key, type=3)

if __name__ == "__main__":
  import socket, sys, struct

  interact(mydict=globals(), mybanner="EAPOL")
```

Le source est également disponible sur la page de l'UE `http://p-fb.net/master-1/reseaux-ii.html`.

Remise du travail

Le travail devra être remis sous forme d'une archive sur « Community Science ».

Cette archive contiendra vos sources (programme Python, modules *etc.*) ainsi que le rapport au format PDF.

Quelques fonctions utiles

Une fonction de conversion d'une chaîne d'octet en sa valeur entière :

```
def strtoint(chaine):
   return int(chaine.encode('utf8').hex(),16)
```

Une fonction permettant de passer d'une chaîne exprimée en notation hexadécimal aux octets :

'006EFF' ⟶ '\x00\x6E\xFF'

```
def cs(a):
   return bytes.fromhex(a)
```

Pour supprimer le « *padding* » qui peut être présent dans la capture fournie par la carte réseau sans-fil :

xterm

```
>>> p
<EAPOL  version=1 type=KEY len=95 |<WPA_key  descriptor_type=254 SMK_message=0L
encrypted_key_data=0L request=0L error=0L secure=0L key_MIC=present key_ACK=0L
install=0L key_index=0L key_type=Pairwise Key key_descriptor_Version=HMAC-MD5 MIC
len=0 replay_counter='\x00\x00\x00\x00\x00\x00\x00\x02' nonce='' key_iv=''
wpa_key_rsc='' wpa_key_id=''
wpa_key_mic='\x00\x00\xf3\xa0\xf6\x91N(\xa2\xdf\x100a\xa4\x1e\xe8'
wpa_key_length=14456 |<Padding  load='\x00\x00' |>>>

>>> bytes(p)
b'\x01\x03\x00_\xfe\x01\t\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\xf3\xa0\xf6\x91N(\xa2\xdf\x100a\xa4\x1e\xe88x\x00\x00'

>>> bytes(p)[:-len(bytes(p[Padding]))]
b'\x01\x03\x00_\xfe\x01\t\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\xf3\xa0\xf6\x91N(\xa2\xdf\x100a\xa4\x1e\xe88x'
```

Le padding est du à l'implémentation du mode « moniteur *» de la carte sans-fil. Ce mode permet de récupérer les trames de gestion de la liaison sans-fil qui ne sont pas fournies dans le mode «* promiscuous *».*