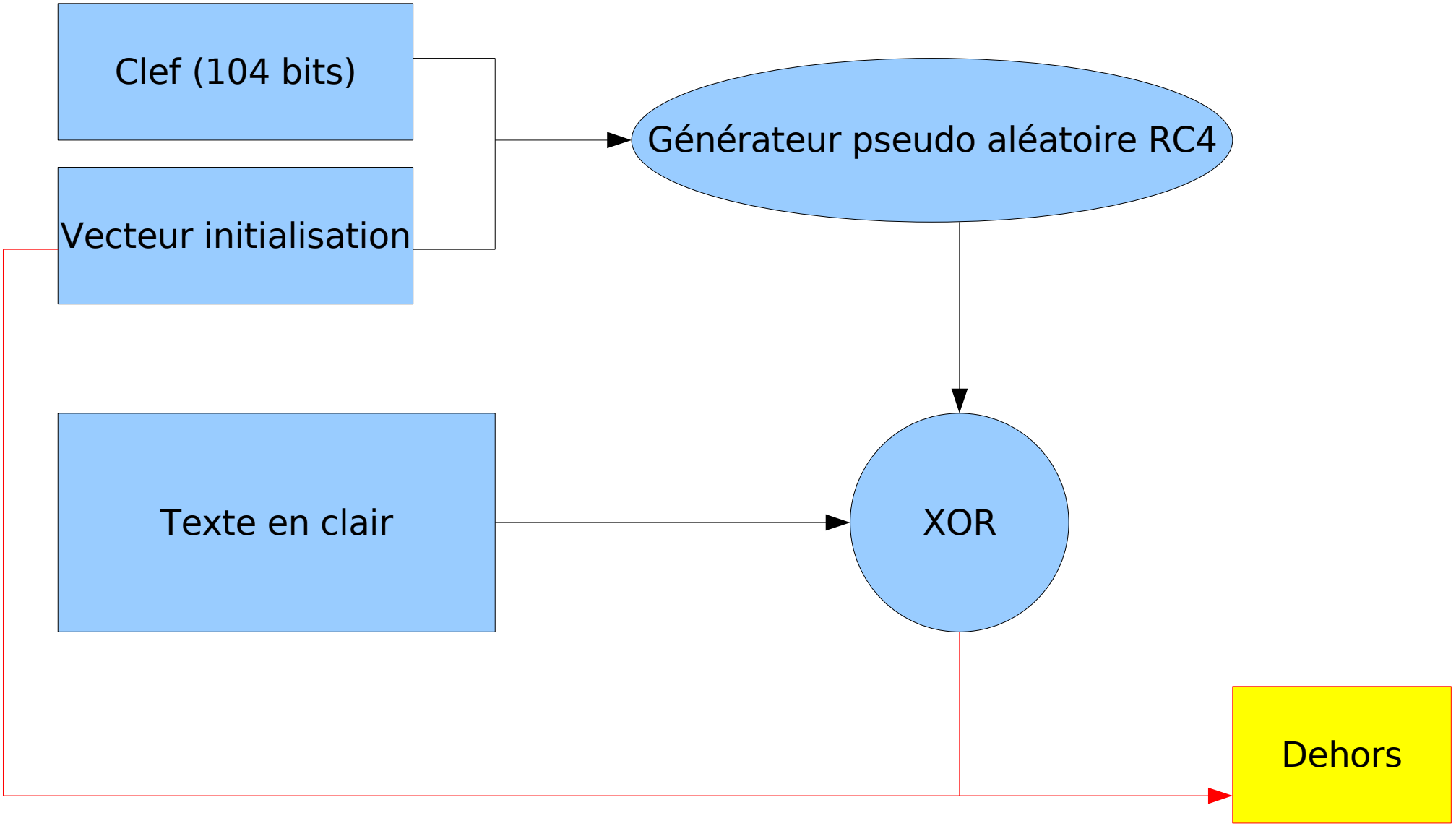
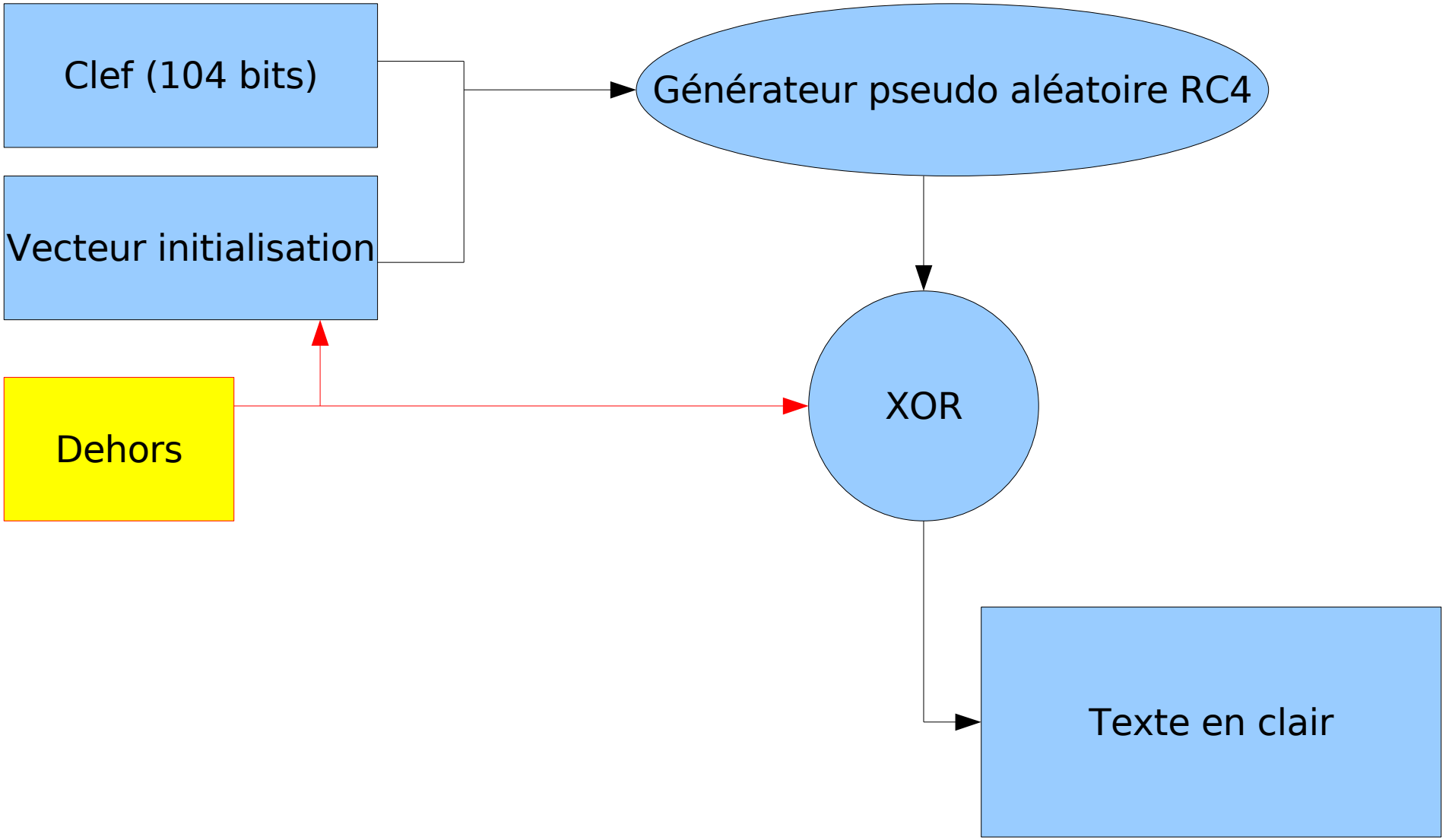
Principe de cryptage d'un flux de données



Décodage


Clef (104 bits)
Vecteur initialisation
Générateur pseudo aléatoire RC4
Dehors
XOR
Texte en clair

Bilan

Secret:

* La clef

Public:

* Le vecteur d'initialisation

* Le texte crypté et la longueur

* L'algorithme

* Le type de réseau

Mitigé:

* Caractéristiques physiques du réseau (adresses MAC, ESSID)

RC4

Données:

Vecteur K concaténé du vecteur d'initialisation et de la clef

Initialisation:

```
let make_S () =
let S = make_vect N 0 in
  (
    for i=0 to N-1 do S.(i) <- i done;
    let j = ref 0 in
      for i = 0 to N-1 do
        j:=(!j + S.(i) + K.(i mod L)) mod N;
        (swap S i !j);
      done;
      S);;
```

```
let swap s i j =
  let c = s.(i) in
    (s.(i)<-s.(j);s.(j)<-c);;
```

N=256 (on travaille sur des octets)
L=Longueur du vecteur K
(**swap** V i j) fait l'échange de V.(i) et V.(j)

Cette étape est systématiquement faite avant toute génération de nombre aleatoire

«Round» (=tranche de 256 octets aléatoires)

GLOBAL: let I = ref 0 and J=ref 0 and S = make_S ();;

Obtention d'un octet:

```
let suivant ()=
  let ip = (!I+1) mod N in
    let jp = (!J + S.(ip)) mod N in
    (
      (swap S ip jp);
      I:=ip;J:=jp;
      S.((S.(ip)+S.(jp)) mod N)
    );;
```

Corrélation:

La probabilité d'avoir une paire de 0 consécutifs est 1/(N.(N+1)) au lieu de 1/N^2

2^26 octets permettent de montrer que ça n'est pas aléatoire...

Principes des attaques

(on suppose la concaténation K=Clef|Vecteur d'initialisation)

Clef -> tableau S

```
let make_S () =
let S = make_vect N 0 in
  (
    for i=0 to N-1 do S.(i) <- i done;
    let j = ref 0 in
      for i = 0 to N-1 do
        j:=(!j + S.(i) + K.(i mod L)) mod N;
        (swap S i !j);
      done;
      S);;
```

La probabilité que S.(i) ne soit plus modifié après est de l'ordre de (1-1/256)^(255-1) soit en gros 1/e (0,37 environ)

On cherche S_ap défini par:

```
let make_S_approche () =
  let l = vect-length K in
  let s = make_vect l  0 in
    (
      for i=0 to l-1 do s.(i) <- i done;
      let j = ref 0 in
        for i = 0 to l-1 do
          j:=(!j + i + K.(i)) mod N;
          s.(i)<-(!j);
        done;
       s);;
```

A partir de ce S_ap=f(K), on retrouve K

(quelques cas pénibles où il est nécessaire de faire des essais sur un nombre limitée de K possibles)

Usuellement:

$$S\_ap.(i)=K.(0)+K.(1)+...+K.(i)+i.(i+1)/2\ [256]$$

Clefs faibles (1 sur 256):

$$S\_ap.(2)=K.(2)+3\ [256] \text{ lorsque } K.(0)+K.(1) = 0[256]$$

$$S\_ap.(i)=K.(i)+i.(i+1)/2\ [256] \text{ lorsque } K.(0)+..+K.(i-1) = 0[256]$$

Si le vecteur d'initialisation est mis avant la clef, cela concerne à chaque fois un cas sur 256. WEP: on retrouve K.(3), puis K.(4), ..., K.(L-1)

Il aurait fallu ne pas utiliser les premiers rounds de RC4

LA Faiblesse de RC4

```
let suivant ()=
  let ip = (!I+1) mod N in
    let jp = (!J + S.(ip)) mod N in
    (
      (swap S ip jp);
      I:=ip;J:=jp;
      S.((S.(ip)+S.(jp)) mod N)
    );;
```

K

I=ip

J=jp

K=S.(I)+S.(J) [N]

P(S.(J)+S.(K)=I [N]) = 2/N (au lieu de 1/N attendu)

Et, si C <> I

P(S.(J)+S.(K)=C [N]) = (N-2)/(N(N-1)) (au lieu de 1/N attendu)

Pour N=256, on obtient 0,78% au lieu de 0,39% et 0,389% au lieu de 0,39%

Exploitation

Considérons le premier octet fourni par RC4 (I=1)

Notons S.(i) la valeur de S.(i) au tout début (celle qui nous intéresse), on a donc lorsque la première valeur aléatoire est rendue:

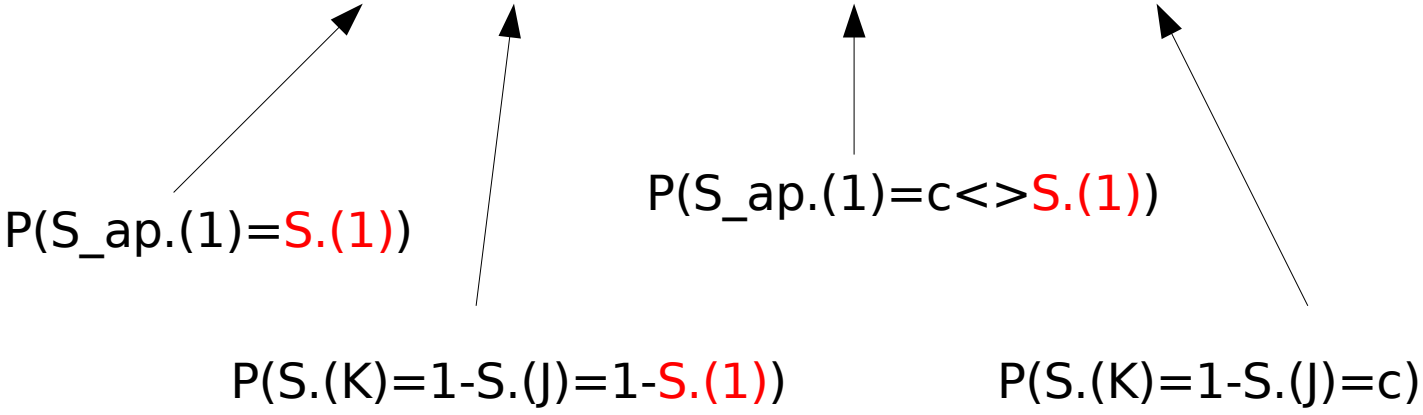
$$I=1\ ,\ J=S.(1)\ ,\ S.(1)=S.(J)\ ,\ S.(J)=S.(1)\ ,\ K=S.(1)+S.(J)$$

On veut la valeur de S.(1) soit donc S.(J). Ce qui est renvoyé est S.(K) or

$$\mathbf{P(S.(J)+S.(K)=1\ [N])\ =\ 2/N}$$

donc la probabilité que S.(K) renvoyé soit égal à 1-S.(1) [N] est en gros

$$(1/e).(2/N)\ +\ (1-1/e).(N-2)/(N.(N-1))\ \text{soit}\ 1{,}36/N$$



La technique consiste donc à regarder les valeurs successives du premier octet du flux engendré. La valeur 1-S_ap.(1) est majoritaire.

De même, si on regarde le k ième octet du flux engendré, la valeur k-S_ap.(k) (modulo N) sera majoritaire

Par cette méthode, on retrouve toutes les valeurs (en fait 3-4 candidats potentiels quand j'ai essayé) du tableau S_ap permettant de retrouver la clef K excepté la valeur S_ap.(0).

Celle ci se fera par essais (N=256 essais donc).

Utilisation du texte codé seul

* Chaque requête ARP est de longueur 16 (la longueur d'un paquet est en clair)

* Les requêtes ARP sont fréquentes

* Chaque requête ARP commence par

AA AA 03 00 00 00 08 06 00 01 08 00 06 04 00

puis dans la cas d'une demande suivi de 01 et de 02 pour une réponse.

* Un texte fréquent T permet de retrouver un biais de sur représentation sur le code fabriqué par RC4 dans le texte codé. On calcule donc une sur représentation dans le texte codé et on en déduit la sur représentation dans le code. Le biais est diminué mais reste significatif et exploitable.

Tactiques d'attaques

Une stupidité: l'authentification

Le point d'accès envoit un texte en clair T que le client qui veut s'authentifier doit renvoyé codé. Un pirate à l'écoute voit donc passer en clair:

* Un texte en clair T
* Le vecteur d'initialisation VI
* Le texte codé $T \oplus C=F$

Il en déduit immédiatement le code RC4 correspondant à VI: $C=F \oplus T =T \oplus T \oplus C$

IL PEUT S'AUTHENTIFIER SANS PROBLÈME

De plus si T' est un nouveau texte, posons $M=T \oplus T'$, T' codé par C sera

$F'=T' \oplus C=T \oplus M \oplus C=M \oplus T \oplus C=M \oplus F$ connu

On peut envoyer un texte codé de même longueur sans problème

L'utilisation de cela avec le protocole réseau permet de «générer» du trafic sur un réseau WIFI sans problème (1)

Il existe d'autres techniques (rajout d'un octet au code et test sur le point d'accès,...)

(1) (sauf au CIRM!)

EXERCICE:

Sur http://boisson.homeip.net/Luminy/ un fichier RC4-test fabrique des flux RC4 à partir d'une clef de 5 octets + 3 octets de VI mis à la fin (non donnés)

Source avec une clef différente sur le même répertoire ainsi qu'un fichier RC4-verif permettant de voir les «rounds» successifs du vecteur VI
[| 9;5;7 |]