Sécurité des réseaux WIFI

PTUT

Année universitaire 2019 - 2020

DUT Informatique - Année 1 - Semestre 2

IUT d'Aix en Provence

Florian Ancelin - Lucas Duval - Clément Mathieu–Drif
Jean-Christophe Nguyen - Nils Ponsard - Hugo Tritz

Tuteur
Samuele Anni

02/06/2020

Auteurs des parties

Éléments du WEP, Authentification, Chiffrement avec WEP, Spécificité du protocole, Fin de vie : Lucas Duval et Jean-Christophe Nguyen

RC4, KSA, PRGA, Fin de l'exemple, Déchiffrement : Florian Ancelin et Clément Mathieu–Drif

WPA2 (Historique, Modes de chiffrement, AES) : Clément Mathieu–Drif

Détection des intrusions : Lucas Duval, Jean-Christophe Nguyen et Nils Ponsard

Failles de sécurité du WEP : Lucas Duval, Jean-Christophe Nguyen

Description du 4-way handshake : Florian Ancelin

Failles de sécurité du WPA2 (KRACK, KR00K, autres attaques) : Clément Mathieu–Drif

Liens utiles

Site internet du projet : `https://cmdetu.gitlab.io/ptut_securite_wifi/`
Dépôt GitLab : `https://gitlab.com/cmdetu/ptut_securite_wifi` (diagramme de Gantt disponible sur ce dépôt)

Table des matières

1 Introduction

Les technologies sans fil (WIFI, Bluetooth, NFC, etc) sont utilisés tous les jours et par tous mais ne sont maitrisées que par une minorité de leurs utilisateurs.

Étant nous même utilisateurs, nous avons décidé, dans le cadre du projet tutoré du second semestre de DUT informatique, de nous intéresser à l'une d'entre elles et plus particulièrement à la manière dont elle sécurise les données transmisses. Le travail effectué durant ce projet aura donc pour but de comprendre comment fonctionne la sécurisation des connexions WIFI.

Les sujets principaux traités dans le cadre de ce projet sont :
- Étude de protocoles WEP et WPA2
- Description des failles de ces deux protocoles
- Présentation des méthodes de détection des intrusions

Première partie
Protocoles

2 WEP

WEP (Wired Equivalent Privacy) est un protocole réseau qui protège les données circulant sur un réseau WIFI. Le WEP est employé dans un WLAN (Wireless Local Area Network) pour rendre inintelligible à un tiers non autorisé les données encapsulées dans des trames.

Le WEP a pour objectif de garantir la confidentialité, l'authentification et l'intégrité des données et il est inclus dans la norme IEE 802.11 (1997). Le but recherché est d'atteindre le niveau de sécurité d'une communication filaire.

Le WEP utilise une clé RC4 (Rivest Cipher 4 : algorithme de chiffrement symétrique décrit plus loin dans ce dossier) composée d'une partie de 40 bits et d'un vecteur d'initialisation de 24 bits. Un contrôle de redondance cyclique (CRC-32) est également utilisé pour assurer l'intégrité des données. Quand le WEP est activé sur un réseau local sans fil, chaque trame est chiffrée dans un flux codé RC4, généré par une clé 64 bits. Ce protocole a été remplacé par le WPA (Wi-Fi Protected Access) en 2003 (Wifi Alliance) puis déprécié en 2004.

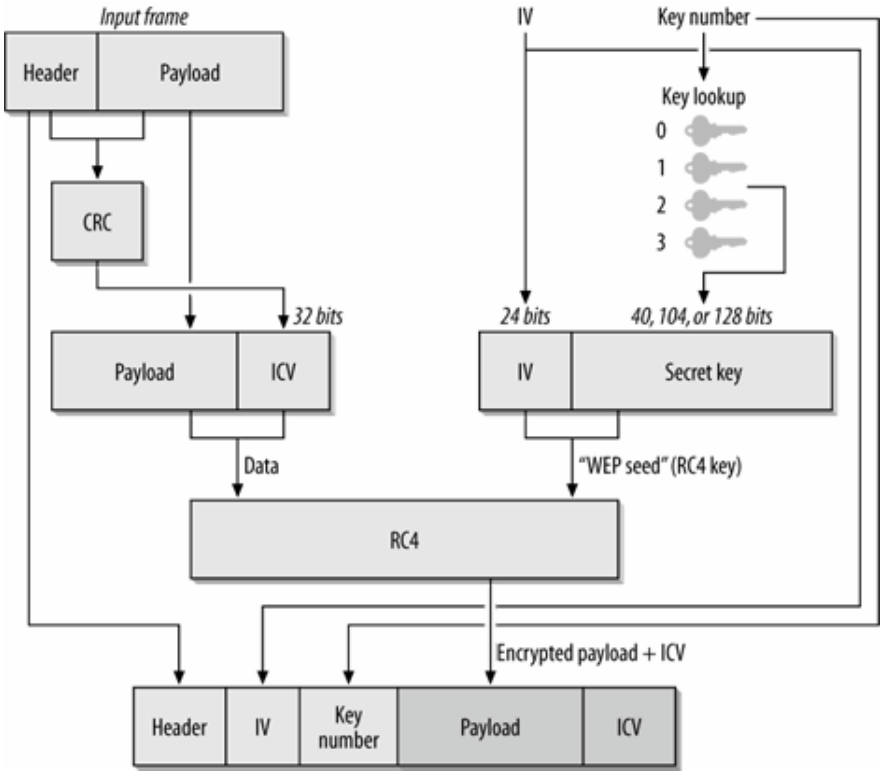
2.1 Éléments du WEP

— WEP utilise une clé secrète qui est partagée entre les clients ; elle sert au chiffrement, au déchiffrement et à l'authentification. Cette clé contient une séquence "secrète" de 40 bits ou 104 bits. Elle peut être composée de caractères ASCII (5 pour 40bits) ou des caractères hexadécimaux (10 pour 40 bits). Les caractères ASCII doivent être affichables, ce qui réduit les possibilités par rapport à l'hexadécimal.

— Les clés de chiffrement supportées et principalement utilisées sont composées de 64 bits avec 40 bits pour la clé et 24 bits pour le vecteur d'initialisation ou de 128 bits dont 104 bits pour la clé et 24 bits pour le vecteur d'initialisation. La clé peut être considérée comme une suite de caractères qui changent selon les bits. Pour WEP-64, c'est 5 caractères ASCII pour 40 bits ou 10 caractères hexadécimaux pour 40 bits. Pour WEP-128, ce sont 13 caractères ASCII pour 104 bits ou 26 caractères hexadécimaux pour 104 bits.

— Le vecteur d'initialisation (IV) est une séquence de bits qui change régulièrement. Combiné à la clé statique, il introduit une notion aléatoire au chiffrement. Ainsi, deux messages identiques ne donneront pas le même contenu chiffré, puisque l'IV est dynamique. Le vecteur d'initialisation peut avoir 2^{24}=16 777 216 combinaisons différentes ce qui permettra d'obtenir des clés de chiffrement "uniques" rendant le déchiffrement plus difficile. Plus la clé est longue, plus la sécurité est optimisée (une clé de 128 bits mettra plus de temps à être trouvée qu'une clé de 64 bits).

— Le WEP utilise un mécanisme nommé Integrity Check Value (ICV), destiné à contrôler l'intégrité des trames. Pour cela, un code équivalent au CRC32 est calculé. Il résulte du message en clair M et non du contenu chiffré. Le CRC32 correspond en fait au reste de la division en binaire du message par un diviseur fixé à l'avance (polynôme de degré 32).



https://flylib.com/books/en/2.519.1/wep_cryptographic_operations.html

2.2 Authentification

2.2.1 Open System auth

— Cette méthode peut-être considérée comme une authentification nulle car pas d'authentification. La clé doit être utilisée lors de l'appairage.

2.2.2 Shared Key auth

4 étapes :
— Le client envoie une requête d'authentification
— Le point d'accès répond avec un challenge (non chiffré)
— Le client chiffre le challenge et le renvoie

— Le point d'accès déchiffre le challenge chiffré et regarde si le texte est le même

Si le réseau est perturbé l'authentification échouera.

Il y a une faille plutôt évidente : si on intercepte le challenge non chiffré puis la réponse chiffrée, un simple XOR (OU exclusif) permet à un attaquant de recueillir suffisamment d'informations pour s'authentifier à son tour.

2.3 Spécificités du protocole

2.3.1 Différents types de WEP

Il y a deux types de WEP :

— *Le WEP statique :* il ne doit être pas être utilisé n'importe quand et n'importe comment, c'est une solution qui n'est pas la meilleure mais qui reste adaptée dans certains cas.
Le WEP statique est capable de contrer des attaques peu intenses et occasionnelles mais pas de grosses attaques. Cependant des appareils tels que des lecteurs de code-barres portables supportent exclusivement le WEP statique, c'est donc la seule option viable pour sécuriser la couche de liaison.

— *Le WEP dynamique :* les meilleures solutions sont basées sur le WEP dynamique. Chaque station utilise 2 clés au lieu d'une :
-La 1ère clé est une clé de mappage partagée entre la station et le point d'accès, elle protège les trames unicast.
-La 2ème clé est une clé par défaut, partagée entre toutes les stations, elle protège les trames multicast et broadcast.

Il suffit de quelques minutes pour reconstituer tous les morceaux de la clé WEP qui circulent de temps à autres sur votre réseau. La raison pour laquelle ils circulent est intimement liée à l'algorithme utilisé car celui-ci doit être initialisé à chaque échange pour ne pas utiliser deux fois la même clé.

Nous verrons dans ce dossier la raison pour laquelle une clé de chiffrement ne doit pas être réutilisée lorsque l'on fait appel à un algorithme de chiffrement qualifié de "One time pad".

2.4 Fin de vie

— Le protocole WEP est donc composé de plusieurs éléments, se décline sous plusieurs formes et a été beaucoup utilisé entre 1999 et 2004 car il était l'un des protocoles les plus sécurisés. Mais aujourd'hui, s'il n'est (quasiment) plus utilisé c'est parce qu'il a été remplacé par le WPA puis le WPA2, des protocoles plus performants et moins vulnérables.

2.5 RC4

RC4 (Rivest Cipher 4) est un algorithme de chiffrement par flot crée en 1987 par Ronald Rivest. Ce document décrit son fonctionnement au travers d'un exemple.

RC4 utilise deux fonction, KSA (Key Scheduling Algorithm) et PRGA (Pseudo Random Generation Algorithm) pour générer un flot de bits pseudo aléatoire qui sera utilisé pour le chiffrement. De manière générale, la taille du tableau à initialiser est de $n = 2^8 = 256$. Les éléments de ce tableau sont des mots de n bits Dans notre cas, le tableau contiendra donc 256 éléments de 1 octet.

"For real applications, n=8 is used as it is a good trade-off between security and memory requirements, and it is also natural and easy to implement" Evaluation of RC4 Stream Cipher - Ed Dawson, Helen Gustafson, Matt Henricksen, Bill Millan

Remarque 1 : les implémentations fournies dans ce document (en C++) utilisent n = 256.

Remarque 2 : les tableaux représentés dans ce document le sont avec l'élément d'indice $i = 0$ à gauche et celui d'indice $i = tailleDuTableau - 1$ à droite. Avant l'exécution de ces deux fonctions, le tableau est initialisé ainsi :

```
array<uint8_t, 256> tab;
for(unsigned i(0); i < tab.size(); ++i)
    tab[i] = uint8_t(i);
```

Soit la permutation identité de $Sym(n)$, $id_n = (0, 1, ..., n-1)$

$j_i \in \mathbb{Z}/n\mathbb{Z}$
$\hat{j_i} \in \mathbb{Z}/n\mathbb{Z}$

$\sigma_i \in Sym(n)$
$\hat{\sigma_i} \in Sym(n)$

les $K_i \in \mathbb{Z}/n\mathbb{Z}$ sont les caractères de la clé K de taille $l \in [1, n]$
$id_n = (0, 1, ..., n-1)$

2.6 KSA

Après initialisation, le vecteur est parcouru une première fois par l'indice i afin d'appliquer une série de permutations. Les permutations effectuées par KSA sont calculées en fonction de la clé K.

Les permutations sont définies par ces relations :

$$j_i = \begin{cases} 0 & \text{si i} = 0 \\ j_{i-1} + \sigma_{i-1}(i-1) + K_{i-1} \mod l & \text{si } 1 \leq i \leq n \end{cases}$$

$$\sigma_i = \begin{cases} id_n & \text{si i} = 0 \\ (i-1, j_i) \circ \sigma_{i-1} & \text{si } 1 \leq i \leq n \end{cases}$$

l'implémentation du KSA est la suivante :

```
uint8_t j(0);
for(unsigned i(0); i < tab.size(); ++i)
{
    j = (j + tab[i] + cle[i % cle.size()]) % 256;
    swap(tab[i], tab[j]);
}
```

où

```
j = (j + tab[i] + cle[i % cle.size()]) % n
```

correspond à

$$j_i = j_{i-1} + \sigma_{i-1}(i-1) + K_{i-1} \mod l$$

Puisque $\sigma_0 = id_n$ et que cela correspond à la première initialisation du tableau, la boucle peut commencer à σ_1. La permutation effectuée par le programme lorsque $i = 0$ correspond donc à σ_1. Le programme effectue alors $n-1$ permutations. La i^{eme} permutation de l'algorithme correspond alors à la $i+1^{eme}$ permutation calculée avec la relation précédente. L'indice `i` de l'implémentation varie donc entre 0 et $n-1$.

Voici un exemple avec `n` = 5, `K` = | 4 | 2 | 1 | et `chaine` = | 0 | 1 | 2 |

$j_0 = 0$
$\sigma_0 = id_5$

$j_1 = j_0 + \sigma_0(0) + K_0 = 0 + 0 + 4 = 4$
$\sigma_1 = (0, 4) \circ id_5 = (0, 4)$

$j_2 = j_1 + \sigma_1(1) + K_1 = 4 + 1 + 2 = 7 \equiv 2 \mod 5$
$\sigma_2 = (1, 2) \circ (0, 4)$

$j_3 = j_2 + \sigma_2(2) + K_2 = 2 + 1 + 1 = 4$
$\sigma_3 = (2, 4) \circ ((1, 2) \circ (0, 4))$

$j_4 = j_3 + \sigma_3(3) + K_0 = 4 + 3 + 4 = 11 \equiv 1 \mod 5$
$\sigma_4 = (3,1) \circ ((2,4) \circ ((1,2) \circ (0,4)))$

$j_5 = j_4 + \sigma_4(4) + K_1 = 1 + 1 + 2 = 4$
$\sigma_5 = (4,4) \circ ((3,1) \circ ((2,4) \circ ((1,2) \circ (0,4))))$

ce qui correspond donc à :

0	1	2	3	4
		-		
4	1	2	3	0
		-		
4	2	1	3	0
		-		
4	2	0	3	1
		-		
4	3	0	2	1
		-		
4	3	0	2	1

les valeurs de j calculées par l'algorithme pour ce même exemple sont :
i = 0, j = 4
i = 1, j = 2
i = 2, j = 4
i = 3, j = 1
i = 4, j = 4

et les permutations qui en découlent sont de la forme : `swap(tab[i], tab[j])` soit (0, 4), (1, 2), (2, 4), (3, 1) et (4, 4) (dans l'ordre d'application) ce qui correspond à ce qui a été calculé précédemment.

2.7 PRGA

PRGA (Pseudo-Random Generation Algorithm) est un générateur pseudo aléatoire qui effectue des permutations sur un tableau `tab` pour en initialiser un second `tab2` (appelé "keystream"). A chaque itération, PRGA échange deux éléments du tableau `tab` (qui est le résultat du KSA). Le nombre d'itérations dépend de la taille du message à chiffrer Le tableau mélangé `tab2` est utilisé pour chiffrer le message à l'aide de l'opération XOR (OU exclusif) entre le vecteur mélangé et le message à chiffrer. Chaque élément du tableau est déplacé au moins une fois toutes les $n = 256$ itérations.

implémentation en C++ :

```
uint8_t a(0), b(0);
vector<uint8_t> tab2;
for(unsigned i(0); i < chaine.size(); ++i)
{
    a = (a + 1) % 256;
    b = (b + tab[a]) % 256;
    swap(tab[a], tab[b]);
    tab2.push_back((tab[(tab[a] + tab[b]) % 256]));
}
```

Les permutations appliquées sur `tab` sont définies par les relation :

$$\hat{j}_i = \begin{cases} 0 & \text{si i} = 0 \\ j_{i-1} + \hat{\sigma}_{i-1}(i) & \text{si } i \geq 1 \end{cases}$$

$$\hat{\sigma}_i = \begin{cases} \sigma_n & \text{si i} = 0 \\ (i, j_i) \circ \hat{\sigma}_{i-1} & \text{si } i \geq 1 \end{cases}$$

$i \in [0, l-1]$

les $z_i \in \mathbb{Z}/n\mathbb{Z}$ sont les éléments du second tableau :

$$z_i = \hat{\sigma}_{i+1}(\hat{\sigma}_{i+1}(i) + \hat{\sigma}_{i+1}(j_{i+1}))$$

La taille finale du tableau `tab2` est `chaine.size()` $= l$

Suite de l'exemple avec `n = 5`, `K` = | 4 | 2 | 1 | et `chaine` = | 0 | 1 | 2 |

$j_0 = 0$
$\hat{\sigma}_0 = \sigma_5$

$j_1 = j_0 + \sigma_0(1) = 0 + 3 = 3$
$\hat{\sigma}_1 = (1,3) \circ \sigma_5$

$j_2 = j_1 + \sigma_1(2) = 3 + 0 = 3$
$\hat{\sigma}_2 = (2,3) \circ ((1,3) \circ \sigma_5)$

$j_3 = j_2 + \sigma_2(3) = 3 + 0 = 3$
$\hat{\sigma}_3 = (3,3) \circ ((2,3) \circ ((1,3) \circ \sigma_5))$

$j_4 = j_3 + \sigma_3(4) = 3 + 1 = 4$
$\hat{\sigma}_3 = (4,4) \circ ((3,3) \circ ((2,3) \circ ((1,3) \circ \sigma_5)))$

les différents états de `tab` sont :

4	3	0	2	1

-

4	2	0	3	1

-

4	2	3	0	1

-

4	2	3	0	1

-

4	2	3	0	1

2.8 Fin de l'exemple

Calculons les valeurs des éléments de `tab2` (z_i)
$z_0 = \hat{\sigma}_1(\hat{\sigma}_1(1) + \hat{\sigma}_1(j_1)) = \hat{\sigma}_1(\hat{\sigma}_1(1) + \hat{\sigma}_1(3)) = \hat{\sigma}_1(2+3) = \hat{\sigma}_1(0) = 4$
$z_1 = \hat{\sigma}_2(\hat{\sigma}_2(2) + \hat{\sigma}_2(j_2)) = \hat{\sigma}_2(\hat{\sigma}_2(2) + \hat{\sigma}_2(3)) = \hat{\sigma}_2(3+0) = \hat{\sigma}_2(3) = 0$
$z_2 = \hat{\sigma}_3(\hat{\sigma}_3(3) + \hat{\sigma}_3(j_3)) = \hat{\sigma}_3(\hat{\sigma}_3(3) + \hat{\sigma}_3(3)) = \hat{\sigma}_3(0+0) = \hat{\sigma}_3(0) = 4$

Après `chaine.size()` itérations, les éléments de `tab2` sont : | 4 | 0 | 4 |

la fin du processus de chiffrement consiste en un XOR bit à bit (défini plus loin) entre `tab2` et `chaine`.

```
for(unsigned i(0); i < chaine.size(); ++i)
    tab2[i] = tab2[i] ^ chaine[i];
```

Les implémentations du PRGA et de l'étape finale (XOR) peuvent être regroupées ainsi :

```
uint8_t a(0), b(0);
vector<uint8_t> tab2;
for(unsigned i(0); i < chaine.size(); ++i)
{
    a = (a + 1) % 256;
    b = (b + tab[a]) % 256;
    swap(tab[a], tab[b]);
    tab2.push_back((tab[(tab[a] + tab[b]) % 256]) ^ chaine[i]);
}
```

Le XOR (ou exclusif) est noté $\oplus$ (et `^` en C++) et est défini ainsi :

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

`tab2 ^ chaine`

`tab2` :

4	0	4

`chaine` :

0	1	2

message chiffré :

4 XOR 0	0 XOR 1	4 XOR 2

message chiffré :

4	1	6

2.9 Déchiffrement

Le déchiffrement est très simple dans la mesure où il utilise la même fonction que celle que nous venons de décrire. La raison pour laquelle le chiffrement et le déchiffrement utilisent le même algorithme découle du fait que la fonction correspondante est involutive.

2.9.1 Une fonction involutive est bijective

Soit $f : E \mapsto E$ une fonction involutive.
Soient x et y deux éléments de E

$$f(x) = f(y) \Rightarrow f(f(x)) = f(f(y)) \Rightarrow x = y$$

f est injective

$$f(x) = y \Rightarrow f(f(x)) = f(y) \Rightarrow x = f(y)$$

f est surjective et donc f est bijective

2.9.2 Application au RC4

Soit $f : E \mapsto E$ avec $f_G(M) = M \oplus G$ et E l'ensemble des messages (chiffrés ou non chiffrés).
M est le message à chiffrer, G est le tableau pseudo aléatoire dépendant de la clé de chiffrement (sortie du PRGA) et C est le message chiffré donc $C = f_G(M) = M \oplus G$.
La fonction f est utilisée dans le chiffrement RC4 de la manière suivante :

$$f_G(M) = M \oplus G = C$$

Montrons que $f_G(C) = M$

$f_G(C) = C \oplus G = (M \oplus G) \oplus G$
$= \neg(\neg M.G + M.\neg G).G + (\neg M.G + M.\neg G).\neg G$
$= (\neg(\neg M.G).\neg(M.\neg G)).G + (\neg M.G.\neg G) + (M.\neg G.\neg G)$
$= (\neg(\neg M.G).\neg(M.\neg G)).G + (M.\neg G)$
$= ((M + \neg G).(\neg M + G)).G + (M.\neg G)$
$= ((M.\neg M) + (M.G) + (\neg G.\neg M) + (\neg G.G)).G + (M.\neg G)$
$= ((M.G) + (\neg G.\neg M)).G + (M.\neg G)$
$= (M.G.G) + (\neg G.\neg M.G) + (M.\neg G)$
$= (M.G) + (M.\neg G)$
$= M.(G + \neg G)$
$= M.1 = M$

Nous avons donc :

$$f_G(M) = M \oplus G = C$$

et

$$f_G(C) = C \oplus G = M$$

donc la fonction est involutive.

$$f_G \circ \ f_G = Id_E$$

Dans RC4, la fonction de chiffrement est la même que celle de déchiffrement.

2.9.3 Conséquence

Nous avons vu précédemment qu'une application involutive est bijective. De ce fait, dans le cas de RC4, deux messages différents ne peuvent pas donner lieu au même message chiffré pour un keystream donné. Cette propriété découle de l'injectivité.

3 WPA2

3.1 Historique

Suite à la découverte des vulnérabilités du WEP une nouvelle version du système de sécurisation des réseaux WIFI était nécessaire. La norme IEEE 802.11i est alors rédigée. Une première version, le WPA a été publiée en 2003 alors que la nouvelle norme était encore en rédaction. Finalement en 2004 la norme IEEE 802.11i est terminée et donne naissance au WPA2 encore largement utilisé aujourd'hui.

Il existe deux versions de WPA2 :

— WPA2 personnel qui se base sur un secret partagé

— WPA2 entreprise qui utilise un serveur d'authentification (comme eduroam).

En 2018 la wifi Alliance a annoncé WPA3 corrigeant des problèmes de sécurité de WPA2 (voir à ce sujet la partie consacrée à la faille KRACK). WPA3 a cependant déjà été compromis par Mathy Vanhoef (à l'origine de la faille KRACK du WPA2) et Eyal Ronen, qui ont présenté en avril 2019 l'attaque DragonBlood se basant sur le Dragonfly Handshake utilisé par ce protocole. Des correctifs ont depuis été publiés.

3.2 Chiffrement et modes de chiffrement

3.2.1 TKIP

Temporal Key Integrity Protocol, utilise le chiffrement RC4. Ce type de connexion est considérée comme non sécurisée et est uniquement utilisé pour assurer la compatibilité avec de vieux équipements. TKIP n'est pas traité en détail dans ce document.

3.2.2 CCMP

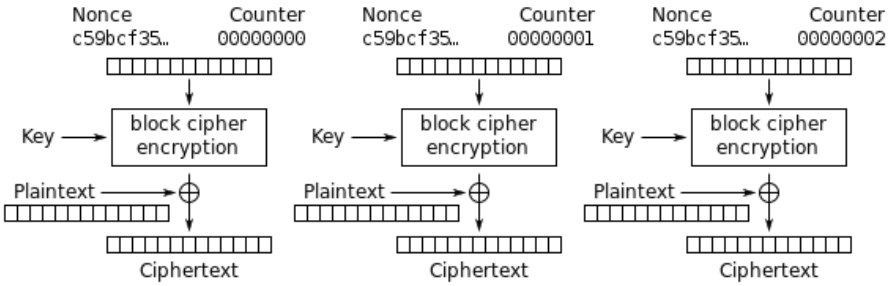
Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP), basé sur le standard AES (Advanced Encryption Standard), permet une protection supérieure et une vérification d'intégrité plus efficace que TKIP. L'utilisation de cette méthode de chiffrement est nécessaire pour dépasser un débit de 54 Mbit/s (spécification 802.11n).
CCMP utilise une clé de 128 bits et un numéro de paquet PN de 48 bits qui n'est jamais réutilisé et évite les attaques de type rejeu. Ce protocole encapsule les données dans un MPDU (MAC Protocol Data Unit).
CCMP utilise le mode CTR pour le chiffrement des données avec AES et CBC-MAC pour contrôler l'intégrité des messages. Commençons par décrire ces deux modes. [1]

1. `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`

3.2.3 CTR - Counter mode

Ce mode ne crée pas de lien entre les blocs de données contrairement au mode CBC décrit dans la partie suivante. Cette particularité peut permettre d'effectuer le chiffrement en parallèle car le traitement d'un bloc ne dépend pas de celui des blocs précédents.

Notons M_i le i^{eme} bloc non chiffré, C_i le i^{eme} bloc chiffré. La fonction de chiffrement est notée AES_k avec k la clé de chiffrement.

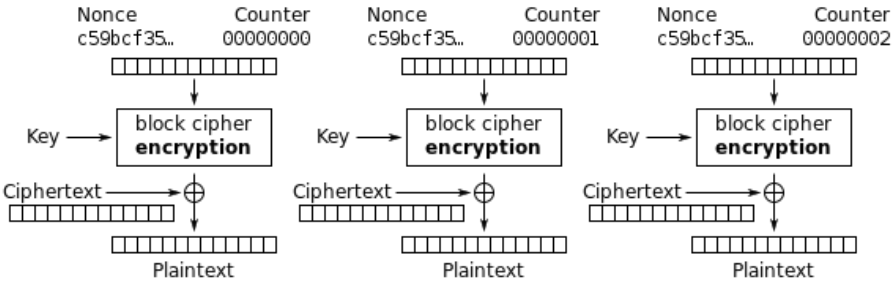


Counter (CTR) mode encryption

(source : Wikipédia)

$$C_i = AES_k(Nonce||Counter_i) \oplus M_i$$

Où $||$ *est l'opérateur de concaténation. Et* $\oplus$ *représente l'opérateur XOR.*



Counter (CTR) mode decryption

(source : Wikipédia)

$$M_i = C_i \oplus AES_k(Nonce||C_i)$$

Dans ce mode, comme pour le chiffrement RC4 par exemple, la fonction de chiffrement est la même que celle de déchiffrement (voir la partie sur RC4).

$$C_i = AES_k(Nonce||Counter_i) \oplus M_i$$

ce qui nous donne donc :

$$AES_k(Nonce||Counter_i) \oplus C_i$$

$$= AES_k(Nonce||Counter_i) \oplus AES_k(Nonce||Counter_i) \oplus M_i = M_i$$

CCMP utilise une valeur appelée *CTR Preload*, elle correspond à la concaténation $(Nonce||Counter_i)$ présente sur les descriptions. Cette valeur est composée d'un octet de drapeaux (flags), d'un octet provenant du champs QoS, de l'adresse de l'émetteur sur 6 octets, d'un PN (6 octets) et d'un compteur PL sur deux octets initialisé à 1 et qui sera incrémenté pour chaque nouveau chiffrement. Soit un total de 128 bits. Ce qui donne finalement :

$$M_i = C_i \oplus AES_k(CTR\ Preload)$$

$$C_i = AES_k(CTR\ Preload) \oplus M_i$$

l'indice i n'apparaît pas ici mais le compteur PL est bien présent dans le $CTR\ Preload$. Ce compteur est incrémenté à chaque nouveau chiffrement. Notons également que lors du chiffrement du MAC (défini dans la partie suivante), le compteur PL du $CTR\ Preload$ est égal à 0.

3.2.4 CBC - Cipher Block Chaining

Chaque bloc à chiffrer subit un XOR avec le bloc chiffré précédent. Pour le premier bloc, un vecteur d'initialisation IV est utilisé.

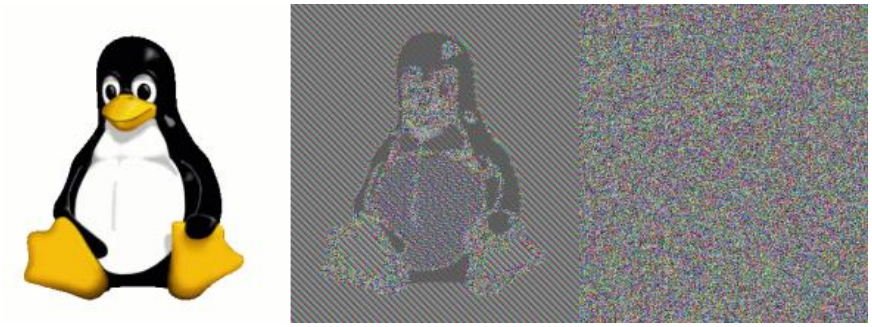
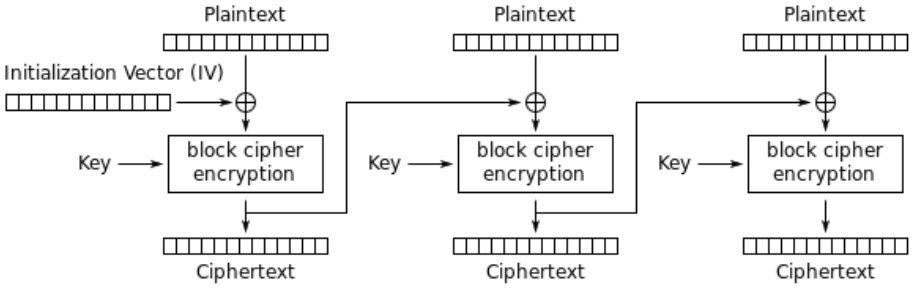


Illustration : à gauche, les données non chiffrées, au centre, le chiffrement utilise le mode ECB et à droite, un autre mode tel que CBC (source : Wikipédia)

Avec le mode ECB (Electronic codebook), deux blocs identiques donnent le même bloc chiffré (ce qui permet de trouver les blocs clairs réutilisés), ce qui n'est pas le cas avec le mode CBC puisque le bloc chiffré dépend des blocs précédents. De plus, l'interdépendance des blocs permet d'assurer qu'un bloc de la chaîne ne peut pas être modifié sans conséquences sur les suivants.



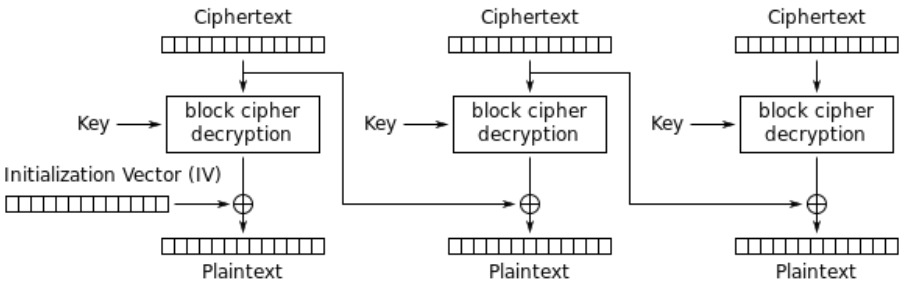
Cipher Block Chaining (CBC) mode encryption

(source : Wikipédia)

Notons M_i le i^{eme} bloc non chiffré, C_i le i^{eme} bloc chiffré avec $i \geq 1$ et $C_0 = IV$. La fonction de chiffrement est notée AES_k et le déchiffrement AES_k^{-1} avec k la clé de chiffrement.

$$C_i = AES_k(M_i \oplus C_{i-1})$$

(source : Wikipédia)



Cipher Block Chaining (CBC) mode decryption

$$C_i = AES_k(M_i \oplus C_{i-1})$$
$$AES_k^{-1}(C_i) = M_i \oplus C_{i-1}$$
$$AES_k^{-1}(C_i) \oplus C_{i-1} = M_i \oplus C_{i-1} \oplus C_{i-1}$$
$$AES_k^{-1}(C_i) \oplus C_{i-1} = M_i$$

3.2.5 CBC-MAC - Message Authentication Code

Ce calcul a pour but de construire un code d'authentification (MAC) en se basant sur le mode CBC afin de contrôler l'intégrité du message ainsi que l'authentification car le processus nécessite de connaitre la clé de chiffrement.

Dans ce cas le vecteur d'initialisation est le résultat du chiffrement par AES d'un *Nonce* très similaire au *CTR Preload*. Ce *Nonce* est construit par concaténation avec un octet de drapeaux, un octet provenant du champ QoS, l'adresse de l'émetteur sur 6 octets, un PN sur 6 octets et deux octets contenant la longueur totale du message à chiffrer.

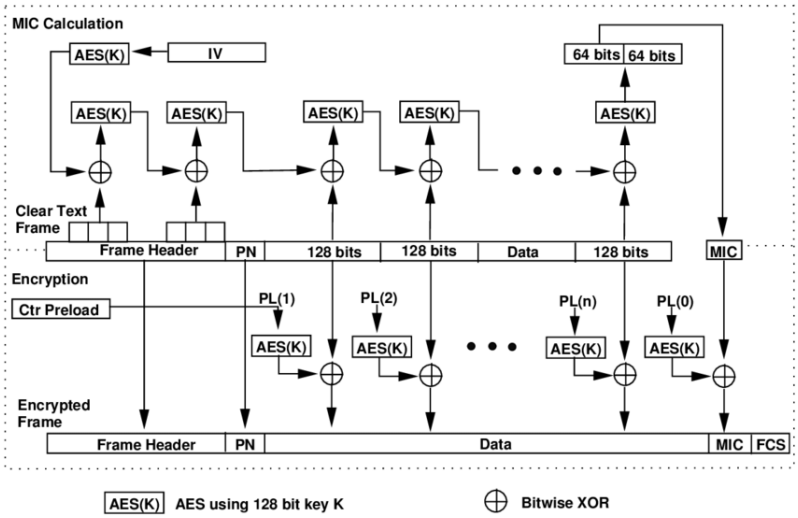
Les 64 premiers bits (bits de poids fort) du dernier bloc chiffré de 128 bits sont conservés pour former le MAC.

Ce MAC permet de vérifier l'intégrité de l'intégralité du message (en-tête comprise) mais dans la trame finale, l'en-tête n'est pas chiffrée.

Lors de la réception, le destinataire déchiffre le message (mode CTR) puis calcule le MAC (mode CBC) qu'il compare au MAC reçu, si les deux résultats correspondent, les données n'ont pas été modifiées et peuvent donc être conservées.

Voici un schéma résumant le fonctionnement de CCMP [2].

2. www.researchgate.net/figure/Diagram-of-the-CCMP-encapsulation-process_fig15_321759837



3.3 AES

Advanced Encryption Standard (AES) est une spécification de chiffrement symétrique par blocs définie par la National Institute of Standards and Technology. La taille des blocs est de 128 bit et la taille de clé peut être de 128,192 ou 256 bits.

Algorithm	Key length (bits)	Block size (bits)	No of round
AES- 128	128	128	10
AES- 192	192	128	12
AES- 256	256	128	14

`https://www.researchgate.net/figure/AES-key-lengths-and-number-of-rounds_tbl2_326557872`

Ce document traite de la version AES-128

AES utilise une matrice carrée d'ordre 4 pour représenter les données à chiffrer (128 bits soit 16 octets, ce qui correspond au nombre d'éléments de la matrice). Notons que dans une implémentation, il est également possible de représenter les données comme un tableau unidimensionnel à 16 éléments.

— indices 0 à 3 : première colonne
— indices 4 à 7 : deuxième colonne
— indices 8 à 11 : troisième colonne

— indices 12 à 15 : quatrième colonne

Octet 00	Octet 01	Octet 02	Octet 03	Octet 04	Octet 05	Octet 06	Octet 07	Octet 08	Octet 09	Octet 10	Octet 11	Octet 12	Octet 13	Octet 14	Octet 15

Octet 00	Octet 04	Octet 08	Octet 12
Octet 01	Octet 05	Octet 09	Octet 13
Octet 02	Octet 06	Octet 10	Octet 14
Octet 03	Octet 07	Octet 11	Octet 15

Quelle que soit la représentation utilisée, les octets sont considérés comme des éléments de $\mathbb{F}_{256}$.

Dans la suite de ce document nous utiliserons la représentation matricielle et la matrice sera notée S.

3.3.1 Opérations

Le corps fini $\mathbb{F}_{256}$ est muni de deux opérations : l'addition et la multiplication.

0 est l'élément neutre pour l'addition et est l'élément absorbant pour la multiplication. 1 est l'élément neutre pour la multiplication.

La multiplication est distributive par rapport à l'addition.

L'addition dans $\mathbb{F}_{256}$ est simple à comprendre et à implémenter car elle est équivalente à un XOR (OU exclusif). Ainsi $0x19 + 0x0d$ est équivalent à $00011001_2 \oplus 00001101_2 = 00010100_2$ donc $0x19 + 0x0d = 0x14$.
La multiplication, quant à elle, est moins directe. Ici, il est nécessaire de représenter les éléments de $\mathbb{F}_{256}$ comme étant des polynômes dans $\mathbb{F}_2[x]/x^8+x^4+x^3+x+1$ (le polynôme $x^8 + x^4 + x^3 + x + 1$ est irréductible). La multiplication se fait en deux étapes :

— multiplication des polynômes
— recherche du reste de la division par $x^8 + x^4 + x^3 + x + 1$

Le résultat est le reste de la seconde étape. La recherche du reste étant similaire à celle utilisée dans le calcul de CRC (Contrôle de redondance cyclique), voici un cours expliquant son fonctionnement :
`https://ensiwiki.ensimag.fr/images/f/f3/Tp_crc16.pdf` Un programme en C++ permettant de réaliser ce calcul de reste est disponible dans le répertoire CPP du dépôt GitLab (lien au début de ce dossier).

Calculons $0x23 * 0x1d$. dans un premier temps voici les polynômes correspondants :

$0x19 = 00100011_2$ donc $x^5 + x + 1$
$0x0d = 00011101_2$ donc $x^4 + x^3 + x^2 + 1$

$$(x^5 + x + 1) * (x^4 + x^3 + x^2 + 1)$$
$$= x^9 + x^8 + x^7 + x^5 + x^5 + x^4 + x^3 + x + x^4 + x^3 + x^2 + 1$$

dans $\mathbb{F}_2$, $1 + 1 = 0$ ce qui nous donne :

$$x^9 + x^8 + x^7 + x^2 + x + 1$$

cherchons le reste de la division par $x^8 + x^4 + x^3 + x + 1$

$$\begin{array}{cccccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \\
- & - & - & - & - & - & - & - & - & \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
 & - & - & - & - & - & - & - & - & - \\
 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}$$

le polynôme trouvé est $x^7 + x^5 + x^3 + x$ soit $10101010_2 = 0xaa$
Notons que la représentation utilisant les polynômes peut être utilisée pour l'addition également, c'est ce qui a été fait pour calculer ceci :

$$x^9 + x^8 + x^7 + x^5 + x^5 + x^4 + x^3 + x + x^4 + x^3 + x^2 + 1$$

$$= x^9 + x^8 + x^7 + x^2 + x + 1$$

Les opérations décrites dans cette partie sont utilisées tout au long de ce document.

3.3.2 SubBytes

La première étape de l'algorithme consiste à calculer l'inverse multiplicatif de chaque élément de la matrice de départ dans $\mathbb{F}_{256}$ et à appliquer la transformation affine suivante dans $(\mathbb{F}_2)^8$ (chaque élément de cet ensemble représente un octet dont les bits sont : $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ avec b_0 le bit de poids faible et b_7 le bit de poids fort) :

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Notons que la matrice utilisée est circulante.

$$\begin{aligned} s_0 &= b_0 + b_4 + b_5 + b_6 + b_7 + 1 \\ s_1 &= b_0 + b_1 + b_5 + b_6 + b_7 + 1 \\ s_2 &= b_0 + b_1 + b_2 + b_6 + b_7 + 0 \\ s_3 &= b_0 + b_1 + b_2 + b_3 + b_7 + 0 \\ s_4 &= b_0 + b_1 + b_2 + b_3 + b_4 + 0 \\ s_5 &= b_1 + b_2 + b_3 + b_4 + b_5 + 1 \\ s_6 &= b_2 + b_3 + b_4 + b_5 + b_6 + 1 \\ s_7 &= b_3 + b_4 + b_5 + b_6 + b_7 + 0 \end{aligned}$$

ce qui correspond à :

$$s_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

avec $c = 0x63 = 01100011_2$. Il est ici question d'associer à la valeur b de chaque octet une nouvelle valeur $s = S(b)$.

D'après le théorème de Lagrange sur les groupes, l'ordre d'un élément d'un groupe est un diviseur du cardinal de ce groupe. donc pour un élément a d'ordre t, dans un groupe de cardinal n, on a $n = k*t, k \in \mathbb{N}$. donc $a^n = (a^t)^k = 1^k = 1$. Alors a^{n-1} est l'inverse de a car $a^{n-1} * a = a^n = 1$. Dans le cas d'AES nous cherchons des inverses dans $\mathbb{F}_{256} - \{0\} = (\mathbb{F}_{256})^*$ (groupe par rapport à la multiplication car tous les éléments ont un inverse). $(\mathbb{F}_{256})^*$ est de cardinal 255. Donc pour tout $A \in (\mathbb{F}_{256})^*$, nous avons $A^{254} = A^{-1}$

Puisque 0 n'a pas d'inverse, nous considérons que sa valeur reste inchangée lors de cette étape et nous avons donc $S(0) = 0x63$.

Exemple : dans cet exemple, un élément de la matrice S est $b = 90 = 0x5a = 01011010_2$. $b \in \mathbb{F}_{256}$ Nous cherchons l'inverse multiplicatif de b noté b^{-1}. Comme nous l'avons vu, $b^{255} = 1$ et $b * b^{-1} = 1$ donc $b^{-1} = b^{254}$

Calculer des produits d'éléments de $\mathbb{F}_{256}$ peut être fait avec une fonction de ce type [3] qui utilise la technique de multiplication dite Russe :

```
uint8_t gmul(uint8_t a, uint8_t b)
{
    uint8_t p = 0;
    while (a && b)
    {
        if (b & 1)
            p ^= a;

        if (a & 0x80)
            a = (a << 1) ^ 0x11b; // x^8 + x^4 + x^3 + x + 1
        else
            a <<= 1;
        b >>= 1;
    }
        return p;
}
```

3. https://en.wikipedia.org/wiki/Finite_field_arithmetic

Élevons b à la puissance 254 :

```
uint8_t resultat = gmul(90, 90); // 2
resultat = gmul(resultat, 90); // 3
uint8_t p3 = resultat;
resultat = gmul(resultat,resultat); // 6
resultat = gmul(resultat, resultat); // 12
uint8_t p15 = gmul(resultat, p3); // 15
resultat = gmul(resultat, resultat); // 24
resultat = gmul(resultat, resultat); // 48
resultat = gmul(resultat, p15); // 63
resultat = gmul(resultat, resultat); // 126
resultat = gmul(resultat, 90); // 127
resultat = gmul(resultat, resultat); // 254
```

Après exécution, nous trouvons $b^{254} = b^{-1} = 34 = 00100010_2$.
ce qui nous donne :

$$s_0 = 0 + 0 + 1 + 0 + 0 + 0 + 1 = 0$$
$$s_1 = 0 + 1 + 1 + 0 + 0 + 0 + 1 = 1$$
$$s_2 = 0 + 1 + 0 + 0 + 0 + 0 + 0 = 1$$
$$s_3 = 0 + 1 + 0 + 0 + 0 + 0 + 0 = 1$$
$$s_4 = 0 + 1 + 0 + 0 + 0 + 0 + 0 = 1$$
$$s_5 = 1 + 0 + 0 + 0 + 0 + 1 + 1 = 1$$
$$s_6 = 0 + 0 + 0 + 1 + 0 + 0 + 1 = 0$$
$$s_7 = 0 + 0 + 1 + 0 + 0 + 0 + 0 = 1$$

soit $S(b) = 10111110_2 = 0xbe$. ce qui correspond bien à la valeur trouvée dans la S-box :

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

(source : Wikipédia)

En pratique, il est possible de conserver la S-box dans le code source d'un programme pour éviter les calculs, c'est ce qui a été fait dans le noyau Linux [4] par exemple :

```
static volatile const u8 __cacheline_aligned aes_sbox[] = {
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
        0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
        0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc
        ...
};
```

Toutes les valeurs de la S-box sont différentes et il n'existe aucun $b \in \mathbb{F}_{256}$ telle que $S(b) = b$. L'application décrite ici est donc une bijection sans point fixe. Notons également qu'il n'existe pas non plus d'élément $b \in \mathbb{F}_{256}$ tel que $S(b) \oplus b = 0xff$ (soit $S(b) + b = 0xff$).

4. https://github.com/torvalds/linux/blob/master/lib/crypto/aes.c

3.3.3 ShiftRows

Cette étape modifie les lignes de la matrice obtenue après l'étape SubBytes. Pour cela, des décalages circulaires vers la gauche sont appliqués sur les lignes selon les données suivantes :

— ligne 0 : reste inchangée
— ligne 1 : 1 rang
— ligne 2 : 2 rangs
— ligne 3 : 3 rangs (équivalent à un décalage d'un rang vers la droite)

S(Octet 00)	S(Octet 04)	S(Octet 08)	S(Octet 12)
S(Octet 01)	S(Octet 05)	S(Octet 09)	S(Octet 13)
S(Octet 02)	S(Octet 06)	S(Octet 10)	S(Octet 14)
S(Octet 03)	S(Octet 07)	S(Octet 11)	S(Octet 15)

S(Octet 00)	S(Octet 04)	S(Octet 08)	S(Octet 12)
S(Octet 05)	S(Octet 09)	S(Octet 13)	S(Octet 01)
S(Octet 10)	S(Octet 14)	S(Octet 02)	S(Octet 06)
S(Octet 15)	S(Octet 03)	S(Octet 07)	S(Octet 11)

À gauche, la matrice après l'étape SubBytes et à droite le résultat de l'étape ShiftRows.

3.3.4 MixColumns

Une fois les rotations appliquées sur les lignes, nous représentons une colonne de la matrice avec un élément de $(\mathbb{F}_{256})^4$.

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

donne (a_0, a_1, a_2, a_3), (a_4, a_5, a_6, a_7), $(a_8, a_9, a_{10}, a_{11})$ et $(a_{12}, a_{13}, a_{14}, a_{15})$.
La transformation suivante est appliquée pour toutes les colonnes :

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$ *La matrice utilisée est circulante.*

Exemple : Considérons une matrice dont une des colonnes serait représentée par $C \in (\mathbb{F}_{256})^4$

$$C = \begin{bmatrix} 01 \\ 20 \\ 6c \\ 2f \end{bmatrix}$$

Les valeurs notées ici sont en hexadécimal.
Ceci nous donne donc :

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} 01 \\ 20 \\ 6c \\ 2f \end{bmatrix} = \begin{bmatrix} 2*01+3*20+1*6c+1*2f \\ 1*01+2*20+3*6c+1*2f \\ 1*01+1*20+2*6c+3*2f \\ 3*01+1*20+1*6c+2*2f \end{bmatrix}$$

$$= \begin{bmatrix} 2+3*20+6c+2f \\ 1+2*20+3*6c+2f \\ 1+20+2*6c+3*2f \\ 3+20+6c+2*2f \end{bmatrix}$$

calculons $3*20$, $2*20$, $3*6c$, $2*6c$, $3*2f$ et $2*2f$.
$0x20 = 00100000_2 = x^5$.
$0x6c = 01101100_2 = x^6 + x^5 + x^3 + x^2$.
$0x2f = 00101111_2 = x^5 + x^3 + x^2 + x + 1$.
$0x03 = 00000011_2 = x + 1$.
$0x02 = 00000010_2 = x$.
ce qui nous donne :

$$3*20 = (x+1)*x^5 = x^6 + x^5 = 01100000_2 = 0x60$$

$$2*20 = x*x^5 = x^6 = 01000000_2 = 0x40$$

$$3*6c = (x+1)*(x^6+x^5+x^3+x^2) = x^7+x^6+x^4+x^3+x^6+x^5+x^3+x^2$$
$$= x^7 + x^5 + x^4 + x^2$$
$$= 10110100_2 = 0xb4$$

$$2*6c = x*(x^6+x^5+x^3+x^2) = x^7+x^6+x^4+x^3 = 11011000_2 = 0xd8$$

$$3*2f = (x+1)*(x^5+x^3+x^2+x+1) = x^6+x^4+x^3+x^2+x+x^5+x^3+x^2+x+1$$
$$= x^6 + x^5 + x^4 + 1$$

$$= 01110001_2 = 0x71$$

$$2 * 2f = x * (x^5 + x^3 + x^2 + x + 1) = x^6 + x^4 + x^3 + x^2 + x = 01011110_2 = 0x5e$$

$$\begin{bmatrix} 2 + 3 * 20 + 6c + 2f \\ 1 + 2 * 20 + 3 * 6c + 2f \\ 1 + 20 + 2 * 6c + 3 * 2f \\ 3 + 20 + 6c + 2 * 2f \end{bmatrix}$$
$$= \begin{bmatrix} 2 + 60 + 6c + 2f \\ 1 + 40 + b4 + 2f \\ 1 + 20 + d8 + 71 \\ 3 + 20 + 6c + 5e \end{bmatrix}$$
$$= \begin{bmatrix} 21 \\ da \\ 88 \\ 11 \end{bmatrix}$$

3.3.5 Préparation de la clé - Key Schedule

L'objectif de cette étape est d'obtenir 11 clés (la clé initiale plus une clé par tour, soit un total de 176 octets) à partir de la clé initiale (16 octets). Comme pour les données à chiffrer, la clé peut être représentée comme une matrice carrée d'ordre 4 et le résultat par une matrice à 4 lignes et 44 colonnes dont les 4 premières colonnes sont celles de la clé initiale.
Nous allons voir comment calculer les colonnes suivantes.

Il est nécessaire de définir les $RCon$ (Round constants) qui sont des éléments de $(\mathbb{F}_{256})^4$. Chaque $RCon$ correspond à un tour, il est donc nécessaire d'en calculer 10.
$RC(i) = x^{i-1}$ pour $1 \leq i \leq 10$.
Représentons ceci sous la forme d'une matrice à 4 lignes et 10 colonnes.

$$\begin{bmatrix} RC(1) & RC(2) & ... & RC(10) \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \end{bmatrix} = \begin{bmatrix} x^0 & x^1 & ... & x^9 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \end{bmatrix}$$

Il est nécessaire de trouver le reste de la division de x^9 par $x^8 + x^4 + x^3 + x + 1$.
Il en est de même pour x^8 (les autres polynômes ont déjà un degré inférieur à 8).
Pour x^8 on a 100000000_2

$$\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ - & - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

le reste est $x^4 + x^3 + x + 1 = 00011011_2 = 0x1b$
Pour x^9 on a 1000000000_2

$$\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \\ - & - & - & - & - & - & - & - & - & \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}$$

le reste est $x^5 + x^4 + x^2 + x = 00110110_2 = 0x36$

Voici la matrice finale obtenue :

$$R = \begin{bmatrix} 1 & 2 & 4 & 8 & 10 & 20 & 40 & 80 & 1b & 36 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

La i^{eme} colonne est utilisée comme RCon pour le i^{eme} tour.

Calcul pour calculer la i^{eme} $(5 \leq i \leq 44)$ colonne de la matrice finale :

— si $i \equiv 1 \mod 4$

pour B, la $(i-1)^{eme}$ colonne de la matrice représentée ainsi :

$$B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

A, la $(i-4)^{eme}$ colonne :

$$A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

La i^{eme} colonne C se calcule ainsi :

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} S(b_1) \\ S(b_2) \\ S(b_3) \\ S(b_0) \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} RC(i-4) \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

remarquons l'ordre des éléments dans :

$$\begin{bmatrix} S(b_1) \\ S(b_2) \\ S(b_3) \\ S(b_0) \end{bmatrix}$$

ceci sera expliqué plus précisément dans l'exemple suivant.

— sinon
pour B, la $(i-1)^{eme}$ colonne et A la $(i-4)^{eme}$ de la matrice (A et B définis comme précédemment).
La i^{eme} colonne C se calcule ainsi :

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Exemple Choisissons la clé initiale suivante :

$$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad a \quad b \quad c \quad d \quad e \quad f$$

représentée ainsi :

$$\begin{bmatrix} 0 & 4 & 8 & c \\ 1 & 5 & 9 & d \\ 2 & 6 & a & e \\ 3 & 7 & b & f \end{bmatrix}$$

colonne 5 :
Nous considérons ici la 4^{eme} colonne.

$$\begin{bmatrix} c \\ d \\ e \\ f \end{bmatrix}$$

Nous appliquons une rotation de 1 octet vers la gauche (la forme matricielle n'est qu'une représentation), ce qui nous donne :

$$\begin{bmatrix} d \\ e \\ f \\ c \end{bmatrix}$$

Effectuons maintenant la même substitution que celle opérée lors de l'étape SubBytes. Pour cela, cherchons les éléments correspondants dans la S-box.

$$\begin{bmatrix} S(d) \\ S(e) \\ S(f) \\ S(c) \end{bmatrix} = \begin{bmatrix} d7 \\ ab \\ 76 \\ fe \end{bmatrix}$$

Ici, nous sommes au premier tour donc la valeur RCon recherchée correspond à la première colonne de la matrice R.

$$\begin{bmatrix} d7 \\ ab \\ 76 \\ fe \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} d6 \\ aa \\ 74 \\ fd \end{bmatrix}$$

colonne 6 :

$$\begin{bmatrix} d6 \\ aa \\ 74 \\ fd \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} d2 \\ af \\ 72 \\ fa \end{bmatrix}$$

colonne 7 :

$$\begin{bmatrix} d2 \\ af \\ 72 \\ fa \end{bmatrix} + \begin{bmatrix} 8 \\ 9 \\ a \\ b \end{bmatrix} = \begin{bmatrix} da \\ a6 \\ 78 \\ f1 \end{bmatrix}$$

colonne 8 :

$$\begin{bmatrix} da \\ a6 \\ 78 \\ f1 \end{bmatrix} + \begin{bmatrix} c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} d6 \\ ab \\ 76 \\ fe \end{bmatrix}$$

colonne 9 :

$$\begin{bmatrix} d6 \\ ab \\ 76 \\ fe \end{bmatrix} \text{ rotation : } \begin{bmatrix} ab \\ 76 \\ fe \\ d6 \end{bmatrix}$$

$$\begin{bmatrix} S(ab) \\ S(76) \\ S(fe) \\ S(d6) \end{bmatrix} = \begin{bmatrix} 62 \\ 38 \\ bb \\ f6 \end{bmatrix}$$

$$\begin{bmatrix} 62 \\ 38 \\ bb \\ f6 \end{bmatrix} + \begin{bmatrix} d6 \\ aa \\ 74 \\ fd \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b6 \\ 92 \\ cf \\ b \end{bmatrix}$$

colonne 10 :

$$\begin{bmatrix} b6 \\ 92 \\ cf \\ b \end{bmatrix} + \begin{bmatrix} d2 \\ af \\ 72 \\ fa \end{bmatrix} = \begin{bmatrix} 64 \\ 3d \\ bd \\ f1 \end{bmatrix}$$

colonne 11 :

$$\begin{bmatrix} 64 \\ 3d \\ bd \\ f1 \end{bmatrix} + \begin{bmatrix} da \\ a6 \\ 78 \\ f1 \end{bmatrix} = \begin{bmatrix} be \\ 9b \\ c5 \\ 0 \end{bmatrix}$$

colonne 12 :

$$\begin{bmatrix} be \\ 9b \\ c5 \\ 0 \end{bmatrix} + \begin{bmatrix} d6 \\ ab \\ 76 \\ fe \end{bmatrix} = \begin{bmatrix} 68 \\ 30 \\ b3 \\ fe \end{bmatrix}$$

...

...

...

Le résultat final est :

$$\begin{bmatrix} 0 & 4 & 8 & c & d6 & d2 & da & d6 & b6 & 64 & be & 68 & ... \\ 1 & 5 & 9 & d & aa & af & a6 & ab & 92 & 3d & 9b & 30 & ... \\ 2 & 6 & a & e & 74 & 72 & 78 & 76 & cf & bd & c5 & b3 & ... \\ 3 & 7 & b & f & fd & fa & f1 & fe & b & f1 & 0 & fe & ... \end{bmatrix}$$

0	4	8	c	d6	d2	da	d6	b6	64	be	68
1	5	9	d	aa	af	a6	ab	92	3d	9b	30
2	6	a	e	74	72	78	76	cf	bd	c5	b3
3	7	b	f	fd	fa	f1	fe	b	f1	0	fe

Clé initiale — Tour 1 — Tour 2

3.3.6 Add RoundKey

Cette étape consiste simplement à additionner La matrice S et une matrice représentant une clé (clé initiale ou clé de tour).

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix} + \begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

Le résultat est ensuite affecté à la matrice S.

3.3.7 Algorithme de haut niveau

Tous les composants d'AES étant maintenant décrits, voici son fonctionnement général :

```
AddRoundKey; //avec la clé initiale et la matrice S
pour i variant_de 1 a 9 faire
    SubBytes;
    ShiftRows;
    MixColumns;
    AddRoundKey; //avec la clé du i-ème tour et la matrice S
ffaire

//le tour final est différent (10ème tour)
SubBytes;
ShiftRows;
//pas de MixColumns dans le dernier tour
AddRoundKey;
```

Le déchiffrement est effectué en faisant toutes ces étapes dans le sens inverse.

Deuxième partie

Détection des intrusions

4 Pourquoi détecter les intrusions ?

Les réseaux WIFI privés sont construits pour être sécurisés et empêcher des connexions non voulues au réseau ; malgré cette "sécurité" il existe des failles qui permettent d'y accéder sans autorisation.

4.1 Quels sont les risques d'une intrusion ?

Avoir accès au réseau facilite la pénétration des systèmes connectés. L'attaquant peut aussi observer les données échangées entre les périphériques, et il lui est d'autre part possible d'utiliser le réseau dans le but d'usurper l'identité de la victime.

Pour éviter ces risques il faut détecter les intrusions malveillantes dans le réseau.

5 Les méthodes de détection

5.1 Détection par adresse MAC

L'adresse MAC est un identifiant particulier à la carte réseau qui permet d'identifier chaque périphérique (cette carte peut être sans fil ou filaire).

On peut détecter une intrusion en obtenant régulièrement les adresses MAC (on peut pour cela utiliser l'outil nmap) des périphériques connectés au réseau et comparer la liste d'adresse obtenue à une liste d'adresses autorisées. Si une adresse n'est pas dans la liste des adresses autorisées alors un nouveau périphérique s'est connecté.

Cette méthode a quelques problèmes :
- Il faut autoriser chaque nouveau périphérique que l'on veut ajouter au réseau.
- Il est possible de changer manuellement l'adresse MAC d'un périphérique (spoofing) et donc d'usurper l'identité d'un périphérique autorisé.

5.2 Détection d'usurpation d'adresse MAC

Les numéros de séquence servent à indiquer quel fragment (en cas de fragmentation) est envoyé dans le paquet. Si ce numéro de séquence change plusieurs fois rapidement (moins d'une seconde), il peut y avoir une usurpation d'adresse

MAC, et cette technique peut donner des "faux positifs" à cause du grand nombre de paquets envoyés.

Les paramètres optionnels sont des données dépendantes de la carte réseau utilisée ainsi que du driver, du firmware et de la configuration. Elles sont difficiles à modifier ; si ces paramètres optionnels changent rapidement (dans la seconde), il y a sûrement usurpation d'adresse MAC.

La qualité du signal reçu dépend de la distance entre le périphérique et le routeur ; ce paramètre n'est pas constant même si les deux équipements restent immobiles (du fait des interférences). Cependant si cette valeur change beaucoup plus que d'habitude et très rapidement (par exemple passer de -80dBm à -20dBm dans la même seconde) on peut suspecter une usurpation d'adresse MAC. Avec cette méthode il est possible d'obtenir des résultats "faux positifs".

Les systèmes d'exploitation ont des façons différentes de générer les trames réseau, ils laissent une «empreinte» qui peut être détectable. Si cette empreinte change rapidement plusieurs fois (moins d'une seconde) alors il y a sûrement usurpation d'adresse MAC.

En combinant ces facteurs il est possible de déterminer si une usurpation d'adresse MAC a lieu et de réduire le nombre de "faux positifs".

5.3 Détection de "Brutforce"

la technique de "Brutforce" consiste à essayer un grand nombre de combinaisons de mots de passe pour essayer de trouver "le bon". Ces combinaisons sont souvent formées à partir de mots du dictionnaire.

détection : La détection est plutôt simple : il suffit de compter le nombre d'essais avec un mot de passe erroné et alerter quand il y a un grand nombre d'essais dans un temps donné.

5.4 Détection d'injections WEP

Les injections WEP prennent souvent la forme de paquets envoyés avec le même vecteur d'initialisation dans un temps réduit. Détecter un grand nombre de paquets avec le même vecteur d'initialisation dans une période de temps courte (une seconde par exemple) est souvent indicateur d'une attaque par injection.

5.5 Détection des injection VLAN

Certains points d'accès peuvent être attaqués par des paquets VLAN de la norme 802.1Q. Pour détecter ce type d'intrusions il suffit de détecter la présence d'en-tête 802.1Q dans les paquets wifi.

5.6 Détection d'usurpation de point d'accès

Certaines attaques consistent à créer un point d'accès avec les mêmes caractéristiques que le point d'accès attaqué pour pouvoir contrôler les données transmises. La détection d'une attaque de ce type se fait au niveau du point d'accès qui doit lancer une alerte s'il détecte un autre point d'accès avec les mêmes caractéristiques que lui.

Troisième partie

Failles de sécurité

6 WEP

6.1 Introduction aux failles du protocole

Le WEP est malheureusement un protocole très vulnérable en raison de ses caractéristiques cryptographiques de qualité insuffisante. Déjà en 1995, des chercheurs ont pu démontrer des failles dans l'algorithme du RC4 (Wagner, Roos,...) utilisé dans le protocole WEP. Mais ce dernier restait encore un protocole plutôt fiable et c'est à partir du début des années 2000 que de plus en plus de faiblesses sont révélées, avec des articles et des publications sur le sujet. Puis en 2004-2005, des attaques montrent la quasi obsolescence du protocole.

Les failles suivantes ont été trouvées dans le protocole WEP :
- En terme de confidentialité, la partie secrète de la clé est la même pour tout le monde et la partie envoyée en clair (IV) a 24 bits (la clé est réutilisée après 224 trames). Les STA (Spanning Tree Algorithm) utilisent le même IV quand elles commencent à transmettre. Il existe des méthodes qui permettent de découvrir la clé WEP à partir d'un certain nombre de trames interceptées. Le paradoxe des anniversaires fait que dès que l'on a capturé environ 5000 paquets, on a plus d'une chance sur deux d'avoir utilisé la même clé.
- Concernant les doublons, aucune protection n'est mise en place.
- Le contrôle d'intégrité n'est pas cryptographiquement sûr ; l'utilisation du CRC est satisfaisante pour des erreurs aléatoires mais pas pour les erreurs délibérées et il est possible de modifier des messages sans être détecté (changer des données et ICV).
- Enfin, aucune authentification bidirectionnelle n'est prévue et seule la station (STA) est authentifiée. Le challenge d'authentification à clé partagée est envoyé en clair ce qui provoque une exposition dangereuse à des attaques connues.

6.2 Différentes attaques sur le protocole

Les attaques ont donc commencé dès 2001 (ces attaques avaient pour but d'exploiter les failles laissées par le protocole WEP). Une multitude d'attaque et de méthode existent aujourd'hui, par exemple :
- L'attaque des "mots de passe" par dictionnaire
- L'attaque "man in the middle"
- Packet injection
- Attaque FMS
- Fake authentication

— L'attaque chopchop de KoreK
— Bit flipping sur la somme de contrôle CRC-32
— Plaintext attack
— Attaque par fragmentation
— Attaque par IV connu
— Utilisation de vecteurs d'initialisation IV « faibles »
— Collision des vecteurs d'initialisation IV
— Réutilisation du vecteur d'initialisation IV

Dans ce document, nous allons nous intéresser plus particulièrement à 2 de ces attaques : Packet injection et FMS.

L'attaque par injection de paquets a pour but d'accélérer la production d'IV nécessaire à l'obtention de la clé. Cette attaque est très efficace lorsqu'une station connectée génère des requêtes ARP. L'attaquant essaie de trouver un paquet ARP chiffré afin de l'utiliser pour l'injection. L'injection de paquets peut être considérée comme une véritable attaque contre le WEP, car ce protocole n'a pas été conçu pour résister à ce type de menace. Un paquet envoyé dans un réseau protégé par le WEP et qui a été intercepté par un attaquant peut ensuite être injecté à nouveau dans le réseau, à condition que la clé n'ait pas été modifiée et que la station d'envoi d'origine soit toujours dans le réseau.

Autre type d'attaque, l'attaque FMS (pour les initiales des noms des chercheurs FLURHER, MANTIN et SHAMIR). En 2001, les chercheurs Scott FLURHER, Itsik MANTIN et Adi SHAMIR ont publié une attaque de récupération de clé qui consiste à exploiter le fait que le protocole WEP possède des IVs faibles. C'est à dire que l'attaquant va récupérer des paquets chiffrés en écoutant passivement le trafic et mémoriser les 3 premiers octets publics. Il connaît alors les l premiers octets d'une clé RC4 utilisée pour générer un keystream X. Il peut ainsi commencer à effectuer les premières étapes du RC4-KSA (Key scheduling algorithm) pour connaître S_l et j_l. Ensuite, $j_{l+1} = j_l + K[l] + S_l[l]$ et une permutation est faite entre $S_l[j_{l+1}]$ et $S_l[l]$. Mais si l'attaquant pouvait connaître $S_{l+1}[l]$ il pourrait alors obtenir $K[l]$ en faisant $S_l^{-1}[S_{l+1}[l]] - j_l - S_l[l]$. Les 3 chercheurs ont alors une méthode pour l'obtenir : supposons que les conditions suivantes se maintiennent après les l premières étapes du RC4-KSA (les conditions 3 et 4 ont été introduites plus tard, par Korek pour améliorer l'efficacité de l'attaque) :

— 1. $S_l[1] < l$
— 2. $S_l[1] + S_l[S_l[1]] = l$
— 3. $S_l^{-1}[X[0]] \neq 1$
— 4. $S_l^{-1}[X[0]] \neq S_l[1]$

Ensuite, une valeur k à laquelle on affecte $S_l[j_{l+1}]$ pour après la permuter avec $S_{l+1}[l]$. Si j change aléatoirement pendant le reste de l'algorithme, les valeurs $S[1]$, $S[S[1]]$ et $S[l]$ ne seront pas modifiées avec une probabilité d'environ

$(\frac{1}{e})^3 \simeq 5/100$ pour le reste de l'algorithme. Quand le premier octet de sortie est produit par le RC4-PRGA (Pseudo-random generation algorithm), j prendra la valeur $S_n[1]$ puis $S_n[1]$ et $S_n[j]$ seront échangés. Après cela, $S_l[1] + S_l[S_l[1]] = l$ tient toujours et le premier octet de sortie du RC4-PRGA $X[0]$ sera alors $S[l]$. Ensuite, si ces 4 conditions sont toujours vérifiées, la fonction suivante :

— $F_{fms}(K[0], ..., K[l-1], X[0]) = S_1^{-1}[X[0]] - j_l - S_l[l]$

prendra alors la valeur de $K[l]$ avec une probabilité de $(\frac{1}{e})^3 \simeq 5/100$. Ainsi, grâce à ces conditions et cette fonction, on peut donc lancer une attaque de récupération complète de la clé sur le WEP. L'attaquant, lorsqu'il a capturé suffisamment de paquets, sélectionne où se trouve les conditions résolues et calcule F_{fms} pour ces paquets. Chacun des résultats pourra être considéré comme un vote pour la valeur de $Rk[0]$ et l'attaquant devra choisir laquelle correspond bien à la vraie valeur de $Rk[0]$. Si la décision est bonne, alors il connaît les premiers l octets de chaque paquet et il fait de même pour $Rk[1]$ et ainsi de suite. Une fois tous les octets de Rk obtenus, l'attaquant vérifie alors la clé qu'il a pu former afin de savoir si elle est correcte ; dans l'affirmative l'attaque a réussi.

En moyenne il fallait au moins 4 000 000 de paquets chiffrés pour retrouver la clé avec une probabilité supérieure à $1/2$.

Autre type d'attaque, Korek en 2004 (qui introduit en même temps l'attaque chopchop). Korek est un hacker qui a publié une série de 16 attaques sur le WEP pour retrouver la clé de chiffrement. Pour cela la structure de l'attaque est la même que pour l'attaque FMS mais celle de Korek permet de réduire le nombre moyen de paquets nécessaire à 700 000 pour une probabilité qui est autour de $1/2$.
Toutefois c'est en 2005 qu'une équipe du FBI démontre la possibilité de rentrer dans un réseau protégé par le protocole WEP en moins de 5 minutes avec des outils pratiquement à la portée de tous. En effet, ces agents du FBI auraient utilisé, entre autre, Kismet, un logiciel de détection de réseaux et Airodump, qui fonctionne avec Aircrack, pour capturer les paquets. Durant cette démonstration, ils ont notamment utilisé une attaque par rejeu.

Cette démonstration prouvait alors définitivement l'insuffisance de ce protocole pour assurer la sécurité d'un réseau (le WPA existait déjà à ce moment-là). Aujourd'hui il est même possible d'entrer dans un réseau protégé par le protocole WEP en moins d'une minute grâce à de bons outils et c'est la raison pour laquelle il n'est presque plus utilisé sur les réseaux WIFI.

7 Authentification : 4-way handshake

Le 4-Way Handshake est utilisé par WPA2. Il permet l'authentification d'un appareil voulant se connecter à un réseau WIFI. Ce processus en 4 temps a pour but de générer différentes clés de chiffrement qui seront ensuite utilisée par le client pour communiquer sur le réseau.

Ce processus étant à la base de la faille KRACK que nous présenterons dans la prochaine partie, il est nécessaire de s'attarder quelques instants sur son fonctionnement.

7.1 Les différentes clés et outils utilisés

- A = Point d'accès
- S = Station
- ANonce = Authenticator Number (used) once : est un nombre généré aléatoirement par le point d'accès
- SNonce = Supplicant Number (used) once : est un nombre généré aléatoirement par le client
- MAC(AA) = adresse MAC du point d'accès
- MAC(SA) = adresse MAC de la station
- PMK = Pairwise Master Key : La PMK est une clé connue du point d'accés et de la station, elle correspond à la PSK (Pre-Shared Key qui représente un secret partagé entre la station et le point d'accès) dans le cas du WPA2 personnel.
- PTK = Pairwise Transcient Key : est une clé utilisée pour chiffrer les trames de la station à destination du point d'accès.
  PRF (PMK + Anonce + SNonce + Mac (AA)+ Mac (SA)) : PRF est une fonction pseudo-aléatoire avec en entrées différentes données citées plus tôt, cette combinaison aura pour but de calculer la PTK.
- GTK = Group Transient/Temporal Key : est une clé commune à toutes les stations associées au point d'accès. Elle est utilisée pour le chiffrement des trames multicast et broadcast.
- MIC = Message Integrity Code (contrôle d'intégrité des données).
- EAPoL-Key [5] = Paquet qui permet l'échange de clé de chiffrement, de forme :

MAC header	Ethernet Type	Version	Packet Type
12 octets	2 octets	1 octets	1 octets

Packet Body Length	Packet Body	Frame Check Sequence
2 octets	Taille variable	4 octets

Pour le 4-Way Handshake, le champ "Type de paquet" prend la valeur 0000 0011 qui désigne le type EAPoL-key.

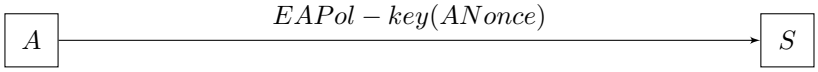
5. https://www.vocal.com/secure-communication/eapol-extensible-authentication-protocol-over-lan/

7.2 Utilisation du 4-Way Handshake

Le 4-way-handshake débute après l'EAP Success reçu par la station. Comme son nom l'indique le 4-way-handshake comporte 4 messages qui sont envoyés entre la station et le point d'accès, le premier message est envoyé par le point d'accès vers la station. Et comme dit précédemment, chaque message transmis est encapsulé avec EApoL-key.

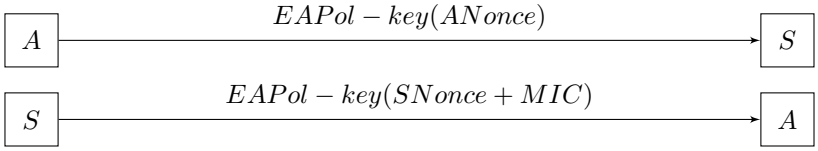
- Premier message :

  Le premier message (du point d'accès vers la station) contient le ANonce et est encapsulé avec EAPoL-Key comme montré ci-dessus (l'encaspulation ne sera plus précisée par la suite mais sera effectuée à chaque étape). À cette étape, la station possède toutes les informations pour calculer la PTK.



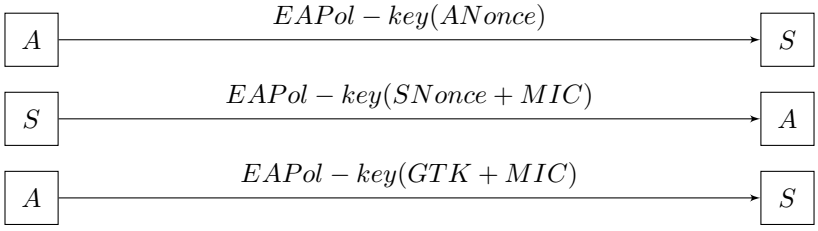
- Deuxième message :

  Le deuxième message (de la station vers le point d'accès) contient le SNonce et un MIC (chiffré par la station). Avec ces informations, le point d'accès est lui aussi capable de calculer la PTK, il peut alors vérifier le MIC.



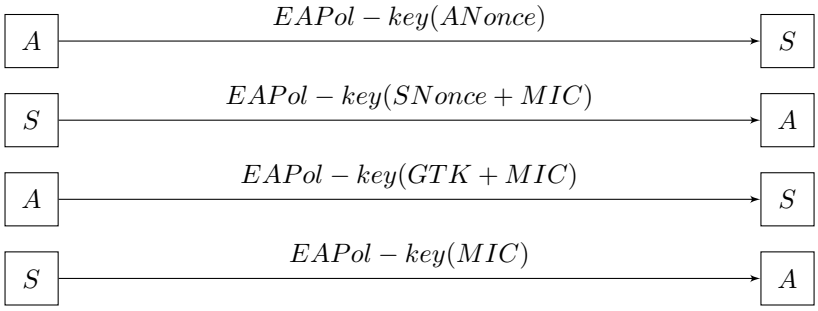
- Troisième message :

  Le troisième message (du point d'accès vers la station), contient la GTK. C'est à la réception de ce message que le point d'accès installe la PTK. Ce point sera traité plus précisément lors du chapitre sur la faille KRACK.



- Quatrième message :

  Le quatrième et dernier message, de la station vers le point d'accès, informe ce dernier que la GTK et la PTK ont été installées, à cette même étape le point d'accès installe lui aussi la PTK.



Après l'exécution du 4-way-handshake, le trafic est ouvert pour la station ; toutes les trames en unicast seront chiffrées avec la PTK et les trames en multicast et broadcast le seront avec la GTK.

8 WPA2

8.1 Faille KRACK

8.1.1 Résumé de la situation

Le protocoles WPA2 (chiffrement CCMP qui s'appuie sur AES) est, depuis l'abandon officiel du WEP par la Wi-Fi Alliance en 2004, le protocole le plus largement utilisé pour la sécurisation des réseau WIFI. Cependant, un type d'attaque nommé KRACK [6] (Key Reinstallation Attacks) découvert par Mathy Vanhoef et Frank Piessens en 2017 permet, grâce à une configuration "homme du milieu" (man in the middle), de déchiffrer des données transmises sur un réseau protégé par le protocole WPA2. Contrairement à d'autres attaques, l'existence de la vulnérabilité ne dépend pas des différentes implémentations du protocole mais se situe dans le protocole lui-même. C'est ici la phase de connexion d'un appareil au point d'accès qui est concernée, cette phase est appelée 4-way handshake (décrit dans la partie précédente). Cette étape permet de définir avec quelle clé (PTK : Pairwise Transient Key - différente pour chaque session) les messages envoyés sur le réseau seront chiffrés. Les protocoles WPA2 personnel et entreprise sont concernés par cette vulnérabilité car ils utilisent tous deux un 4-way handshake. Il existe différentes formes d'attaques KRACK, et nous prenons ici le parti de décrire la plus connue d'entre-elles qui opère sur le troisième message du 4-way handshake.

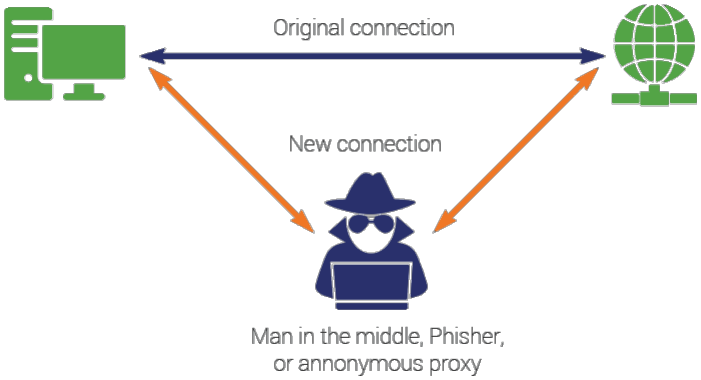
8.1.2 Outils de détection

Mathy Vanhoef a mis à disposition un ensemble de scripts permettant de détecter si un client ou un point d'accès est concerné par les attaques de type KRACK :
`https://github.com/vanhoefm/krackattacks-scripts`

6. KRACK ne permet de trouver ni le mot de passe d'un réseau WIFI protégé par WPA2 ni la clé de chiffrement mise en place lors du 4-way handshake

8.1.3 Attaque de l'homme du milieu (man in the middle : MITM)

Une telle attaque consiste, pour un attaquant, à rediriger le trafic du réseau afin qu'il passe par sa machine (pour cela, il existe différentes méthodes : ARP Poisoning, au DNS Spoofing, ou encore la création d'un point d'accès WIFI...). Cette configuration lui permet de lire les données échangées et dans certains cas de les modifier. Il peut également injecter des données sur le réseau. [7]
`www.thesslstore.com/blog/man-in-the-middle-attack-2/`



8.1.4 Associativité du XOR (eXclusive OR)

algèbre de boole :
XOR est noté : $\oplus$

$$(a \oplus b) \oplus c$$
$$= ((a.\neg b + \neg a.b).\neg c) + (\neg(a.\neg b + \neg a.b).c)$$
$$= (a.\neg b.\neg c + \neg a.b.\neg c) + (\neg(a.\neg b).\neg(\neg a.b).c)$$
$$= (a.\neg b.\neg c + \neg a.b.\neg c) + ((\neg a + b).(a + \neg b).c)$$
$$= (a.\neg b.\neg c + \neg a.b.\neg c) + (\neg a.a.c) + (\neg a.\neg b.c) + (b.a.c) + (b.\neg b.c)$$
$$= (a.\neg b.\neg c + \neg a.b.\neg c) + (\neg a.\neg b.c) + (b.a.c)$$
$$= a.((\neg b.\neg c) + (b.c)) + \neg a.(b.\neg c + \neg b.c)$$
$$= a.\neg(\neg(\neg b.\neg c).\neg(b.c)) + \neg a.(b.\neg c + \neg b.c)$$
$$= a.\neg((b + c).(\neg b + \neg c)) + \neg a.(b.\neg c + \neg b.c)$$

7. Plus d'informations : `https://www.ionos.fr/digitalguide/serveur/securite/attaque-de-lhomme-du-milieu-apercu-du-modele/` et `https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html`

$$= a.\neg(b.\neg b + b.\neg c + c.\neg b + c.\neg c) + \neg a.(b.\neg c + \neg b.c)$$
$$= a.\neg(b.\neg c + c.\neg b) + \neg a.(b.\neg c + \neg b.c)$$
$$= a \oplus (b \oplus c)$$

Donc $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

8.1.5 Fonctionnement de l'attaque

La mise en place de cette attaque débute par la création, par l'attaquant, d'un clone du point d'accès WIFI sur un canal différent (il y a par exemple 11 canaux sur la bande des 2.4GHz). Les données transmises seront ensuite relayées sur le canal réel du point d'accès. Dans cette situation, l'attaquant peut intercepter les données transmises mais est incapable de les déchiffrer. Voici un extrait de *Operating Channel Validation : Preventing Multi-ChannelMan-in-the-Middle Attacks Against Protected Wi-Fi Networks*[8] par Mathy Vanhoef, Nehru Bhandaru et Thomas Derham :
"3.3 Channel Switch Announcements (CSAs)We discovered a novel method to force clients into connecting to the rogue AP. In particular, an adversary can forge Channel SwitchAnnouncements (CSAs) to force a client to switch to the rogue AP'schannel. Normally these announcements are sent when the AP is switching to a different channel. For instance, when an AP operates on certain 5 GHz channels and detects (weather) radar pulses, it must switch to a different channel to comply with regulations.Channel switch announcements can be broadcasted using three different frames. The first is by including a CSA element inside a beacon. The second method is by including a CSA element inside a probe response. And the third technique is to send action frames with a CSA element to associated clients. The first two methods use unprotected management frames, and the third method is protected when MFP is enabled. As a result, even when MFP is used, an adversary can forge CSA elements inside beaconsand probe responses to force clients to switch channels."

La seconde étape consiste à transmettre les trois premiers messages du 4-way handshake puis à bloquer le quatrième. Du point de vue du client, la procédure de connexion est terminée mais le point d'accès n'a pas reçu le dernier message. Dans cette situation, le point d'accès procède à la ré-émission du troisième message qui est traité à nouveau, or, le client ayant déjà installé la clé de chiffrement (PTK), le quatrième message (envoyé pour la seconde fois) est désormais chiffré. L'attaquant doit ici bloquer ce message chiffré qui serait rejeté par le point d'accès et retransmettre la première version du quatrième message (non chiffré) au point d'accès afin qu'il installe la clé de chiffrement à son tour (l'utilisation de ce message 4 non chiffré implique que le numéro de paquet n'est pas le plus récent : toutefois de nombreuses implémentations l'acceptent tout de même car il n'a jamais été utilisé dans une trame de réponse de la part du client mais seulement en émission depuis le point d'accès). À cette étape, la clé est réinstallée par le client, ce qui entraîne la réinitialisation du compteur de trames

8. `https://papers.mathyvanhoef.com/wisec2018.pdf`

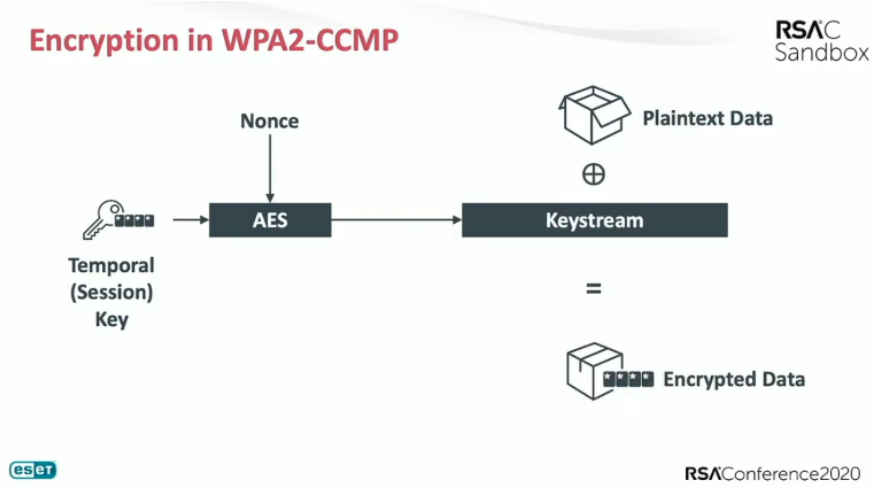
(Nonce) et donc, l'émission de trames avec un Nonce déjà utilisé. Cela permet de déchiffrer les trames envoyées dans la mesure où le keystream est utilisé pour la seconde fois. L'opération XOR bit à bit entre le message 4 chiffré et celui non chiffré donne :

note : Nous utilisons ici les mêmes notations que dans le document décrivant le fonctionnement du chiffrement RC4

$$C = G \oplus M$$

$$C \oplus M = (G \oplus M) \oplus M$$

donc par associativité : $(G \oplus M) \oplus M = G \oplus (M \oplus M) = G$



Représentation simplifiée du chiffrement avec WPA2-CCMP

Nous connaissons alors le keystream utilisé pour les trames chiffrées correspondant à ce Nonce, et elles deviennent donc facilement déchiffrables.

L'utilisation d'un chiffrement sur la couche transport (SSL) permet d'éviter que les données soient déchiffrables grâce à une attaque de type KRACK. Il est cependant possible de contourner cette sécurité grâce à sslstrip [9] comme présenté par Mathy Vanhoef. [10] Il est donc préférable d'utiliser du matériel à jour (non exposé à ce type d'attaques).

8.1.6 Cas particuler : wpa_supplicant 2.4 et 2.5

wpa_supplicant est un client WIFI utilisé par linux et android. Après l'installation de la clé de chiffrement par le noyau linux, wpa_supplicant ne conserve

9. `https://tools.kali.org/information-gathering/sslstrip`
10. `https://www.youtube.com/watch?v=Oh4WURZoR98`

pas de copie de cette clé et l'efface avec des 0 dans la mémoire (conseillé par la norme ieee 802.11).

```
os_memset(sm->ptk.tk,0, WPA_TK_MAX_LEN);
```

Lors de la réinstallation, c'est la clé appelée "all-zero key" qui est installée, et à partir de cet instant, toutes les trames transmises en unicast sont chiffrées avec cette nouvelle clé. Ainsi, les données sont facilement déchiffrables. C'est l'exécution à deux reprises de la fonction `wpa_supplicant_install_ptk()` qui est à l'origine de la vulnérabilité.

Voici une partie du commit [11] qui empêche l'installation d'une clé "all-zero" dans la version 2.6 de wpa_supplicant :

```
+        if (!sm->tk_to_set) {
+                wpa_dbg(sm->ctx->msg_ctx, MSG_DEBUG,
+                        "WPA: Do not re-install same PTK to the driver");
+                return 0;
+        }
```

puis

```
+        sm->tk_to_set = 0;
```

qui empêche une réinstallation de la clé si la fonction est exécutée plus d'une fois. wpa_supplicant 2.6 est cependant vulnérable à un autre type d'attaque de la famille KRACK qui agit sur le premier message du 4-way handshake.

8.1.7 Impact

Dans la pratique, cette attaque permet de déchiffrer les données transmises après l'exécution du 4-way handshake, or, ces données ne sont pas "intéressantes" pour un attaquant. Il est cependant possible de forcer la "désautentification" du client, le système d'exploitation procède alors à un nouveau 4-way handshake afin de se reconnecter au point d'accès. Les données transmises après cette reconnexion pourront être déchiffrées (car les connexions TCP ne sont pas fermées pendant ce processus).

Dans le cas de l'installation d'une "all-zero key" il est possible de calculer le keystream utilisé et donc de déchiffrer les données transmises pour chaque valeur de nonce. Notons tout de même ceci : *"The real AP will then ignore data frames from the victim (because it fails to decrypt them)."* (ceci nous a été précisé lors d'une conversation avec Mathy Vanhoef)

8.1.8 Informations complémentaires

Il est possible de contrôler le nombre de valeurs de nonce qui seront réutilisées en choisissant l'instant de réémission du troisième message du 4-way handshake. *"You can delay the key reinstallation to, in principle, decrypt arbitrary many*

11. https://w1.fi/cgit/hostap/commit/?id=ad00d64e7d8827b3cebd665a0ceb08adabf15e1e

frames. E.g. you let the victim send N frames, and only then trigger to key reinstallation, after which you can decrypt N frames."(ceci nous a également été précisé par Maty Vanhoef).

De plus, si nous ne disposons pas de la version non chiffrée d'une trame mais seulement de deux trames chiffrée avec une réutilisation du Nonce, il est possible d'utiliser des méthodes telles que le crib dragging [12] pour déchiffrer les données.

Voici ce qui nous a été précisé par Mathy Vanhoef *"Here we assume that if there are two message with a reused nonce, we know the plaintext of one of them, e.g. based on the length of the frame we can predict what type of packet it is, and this packet may have a very predictable content. For example, traffic analysis can be used to detect (based on encrypted frames) which plaintext HTTP website a victim is visiting, allowing is to predict the content of a lot of frames. This predicted plaintext can then be used to decrypt the other frames with the reused nonce."*

8.1.9 Correctifs

La majorité des appareils qui étaient encore maintenus lors de cette découverte ont reçu une mise à jour de sécurité empêchant l'exploitation de KRACK. Il faut cependant veiller à ce que les points d'accès (également vulnérables) aient été mis à jour dans la mesure où ce processus n'est généralement pas automatique.

8.2 Faille Kr00k

Une vulnérabilité permettant de déchiffrer les données transmises sur les réseaux utilisant le protocole WPA2 a été découverte par la société ESET en 2019 et rendue publique le 26 février 2020 (RSA Conference). Les chercheurs d'ESET ont découvert cette vulnérabilité alors qu'ils cherchaient à déterminer si l'Amazon echo 2 était concerné par une attaque de type KRACK. Nommée Kr00k, [13] elle concerne la phase de désassociation d'une station et d'un point d'accès qui peut être amorcée de plusieurs manières.

8.2.1 Fonctionnement

Après une désassociation, la clé de chiffrement utilisée est effacée dans la mémoire de la puce WIFI et est remplacée par des 0. Il est cependant possible que des données se trouvent encore dans le tampon d'émission (d'une capacité de 32Kb). Ces trames sont chiffrées avec la clé en mémoire ("all-zero key") et envoyées sur le réseau. Les données sont alors facilement déchiffrables si elles ne sont pas chiffrées à un niveau supérieur (SSL par exemple sur la couche transport).

Les désassociations se produisent naturellement lorsque la station passe d'un point d'accès à un autre, par exemple. Il est cependant possible de forcer cette

12. `http://www.cribdrag.com/about.html`
13. `https://www.welivesecurity.com/wp-content/uploads/2020/02/ESET_Kr00k.pdf`

désassociation par des trames spéciales : "802.11 management frames" qui ne sont pas chiffrées et peuvent donc être envoyées par un attaquant.

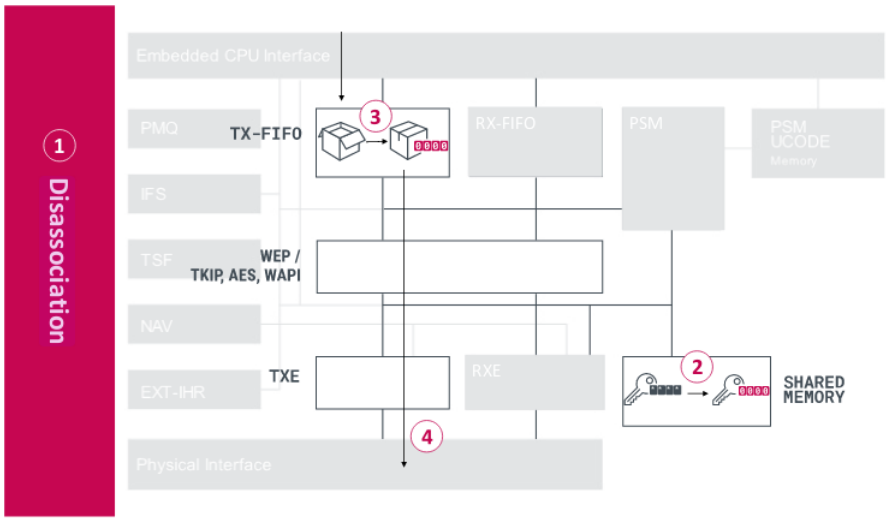


Figure 1 // Kr00k causes transmission of data encrypted with an all-zero key

Source : RSA Conference 2020, ESET

Le contenu des trames récupérées est alors aléatoire car il dépend de l'instant auquel se produit la désassociation. La répétition du processus permet d'obtenir un grand nombre de trames et donc potentiellement des données intéressantes pour un attaquant.

8.2.2 Appareils concernés

Les puces WIFI touchées sont celles des fabricants Broadcom et Cypress, ce qui concerne plus d'un milliard d'appareils (clients et points d'accès). Voici une liste non-exhaustive des modèles concernés :

- Amazon Echo 2nd gen
- Amazon Kindle 8th gen
- Apple iPad mini 2
- Apple iPhone 6, 6S, 8, XR
- Apple MacBook Air Retina 13-inch 2018
- Google Nexus 5
- Google Nexus 6
- Google Nexus 6P
- Raspberry Pi 3
- Samsung Galaxy S4 GT-I9505
- Samsung Galaxy S8
- Xiaomi Redmi 3S

— Asus RT-N12
— Huawei B612S-25d
— Huawei EchoLife HG8245H
— Huawei E5577Cs-321

Si l'attaque est menée sur un point d'accès, ce sont les trames envoyées par ce dernier qui seront déchiffrables même si le client connecté n'est pas vulnérable.

8.2.3 Correctifs

Des correctifs logiciels ont été développés et déployés en 2019 et 2020 par les fabricants des puces WIFI concernées pendant la période de 120 jours octroyée par ESET avant que Kr00k ne soit rendue public.

8.3 Autres attaques

Il existe d'autres attaques visant les réseaux utilisant le protocole WPA2 [14]. Certains logiciels implémentent des attaques par force brute qui ont pour but de trouver le mot de passe sécurisant le réseau en forçant un client à se déconnecter et à procéder à un nouveau 4-way handshake. Ce type d'attaque est très rapide sur les réseaux protégés par un mot de passe faible susceptible de se trouver dans une liste de mots de passe appelée "dictionnaire". Le protocole WPA3 ayant pour objectif d'empêcher les attaques connues sur le WPA2 a été introduit par la WIFI Alliance en 2018 et commence à être utilisé :

WPA3-Personal brings better protections to individual users by providing more robust password-based authentication, even when users choose passwords that fall short of typical complexity recommendations. This capability is enabled through Simultaneous Authentication of Equals (SAE), which replaces Pre-shared Key (PSK) in WPA2-Personal. The technology is resistant to offline dictionary attacks where an adversary attempts to determine a network password by trying possible passwords without further network interaction. [15]

Il est à noter que Mathy Vanhoef et Eyal Ronen ont découvert une attaque [16] visant le "Dragonfly handshake" utilisé par le WPA3 permettant de trouver le mot de passe d'un réseau utilisant ce protocole. Des correctifs ont cependant déjà été apportés et l'utilisation de WPA3 en remplacement de WPA2 est conseillée (si possible) du fait de la complexité des attaques par lesquelles ce protocole est concerné.

14. https://www.privateinternetaccess.com/blog/pmkid-dumping-wifi-password-attacks-are-easier-than-previously-thought/
15. https://www.wi-fi.org/discover-wi-fi/security
16. https://wpa3.mathyvanhoef.com/

Conclusion et références

Conclusion

Comme nous l'avons vu, le chemin qui mène du WEP au WPA2 a été rapidement parcouru, du fait des évidentes failles de sécurité qui sont apparues au fur et à mesure de l'utilisation du WEP, notamment en raison de la présence de l'algorithme de chiffrement RC4 aujourd'hui considéré comme non fiable.
Le WPA a vite montré ses limites et il a fallu l'améliorer, pour arriver au WPA2. Toutefois depuis 2017 déjà, on s'est rendu compte que le WPA2 (IEEE 802.11i) lui-même n'est pas exempt de faiblesses. Les travaux de Mathy Vanhoef et Frank Piessens, deux chercheurs belges, ont dévoilés ses failles, en particulier pour la partie jusque-là considérée comme sûre : la phase d'authentification nommée 4 way handshake.
L'IEEE a donc établi un nouveau cahier des charges, afin de mettre au point le WPA3. Toutefois, les enjeux de la sécurité réseau obligent à des mises à niveau permanentes et il y a fort à parier que le WPA3 devra rapidement être mis à jour. D'autant que Mathy Vanhoef et Eyal Ronen ont signalé qu'ils avaient découvert une faille de sécurité (corrigée depuis) dans le Dragonfly Handshake utilisé par ce protocole.

Références

https://connect.ed-diamond.com/MISC/MISC-053/Cryptanalyse-du-protocole-WEP

https://www.ipv6.com/wireless/wep-wired-equivalent-privacy/

https://whiteflag.blog/books/pentest-wifi/page/comment-fonctionne-wep

https://books.google.fr/books?id=76CWYWYCwXUC&pg=PA343&lpg=PA343&dq=wep+calcul+icv&source=bl&ots=YOiirHovYq&sig=ACfU3U0H-f2B9B5xCryLx23609KfapQk7w&hl=en&sa=X&ved=2ahUKEwj7vZPxwuDnAhXL3oUKHTVqD-IQ6AEwCHoECAkQAQ#v=onepage&q=wep%20calcul%20icv&f=false

https://www.irisa.fr/prive/sgambs/cours2_authentification.pdf

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=2ahUKEwiapv_qxODnAhWJyIUKHa6mBu4QFjACegQIARAB&url=https%3A%2F%2Fwww.marcos-rubinstein.ch%2Fapp%2Fdownload%2F9629721184%2FWireless%2BSecurity%2BWEP.pdf%3Ft%3D1538152580&usg=AOvVaw0pbtbHqpG7ACbXrlfmML3_

https://slideplayer.fr/slide/9881399/

https://dept-info.labri.u-bordeaux.fr/~guermouc/SR/SR/cours//cours6.pdf
https://eprint.iacr.org/2007/471.pdf

http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/wifi

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=2ahUKEwiapv_qxODnAhWJyIUKHa6mBu4QFjACegQIARAB&url=https%3A%2F%2Fwww.marcos-rubinstein.ch%2Fapp%2Fdownload%2F9629721184%2FWireless%2BSecurity%2BWEP.pdf%3Ft%3D1538152580&usg=AOvVaw0pbtbHqpG7ACbXrlfmML3_

http://repository.root-me.org/Réseau/FR%20-%20Wifi%20protocole%20WEP%3A%20mécanismes%20et%20failles.pdf

https://fr.wikipedia.org/wiki/RC4#Attaque_de_Fluhrer,_Mantin_et_Shamir_(attaque_FMS)

https://www.ssi.gouv.fr/uploads/IMG/pdf/NP_WIFI_NoteTech.pdf

https://fr.qwe.wiki/wiki/Fluhrer,_Mantin_and_Shamir_attack

https://connect.ed-diamond.com/MISC/MISC-053/Cryptanalyse-du-protocole-WEP

https://www.lama.univ-savoie.fr/pagesmembres/hyvernat/Enseignement/1213/info528/tp3.html

https://www.memoireonline.com/04/12/5653/m_Mise-en-place-sur-le-point-dacces-dun-reseau-wifi15.html

https://repo.zenk-security.com/Protocoles_reseaux_securisation/Securite%20Wi-Fi%20-%20WEP,%20WPA%20et%20WPA2.pdf

https://www.researchgate.net/publication/291013210_Wi-Fi_Security_Analysis

https://www.kaspersky.com/resource-center/definitions/replay-attack

https://www.networkcomputing.com/wireless-infrastructure/fbi-teaches-lesson-how-break-wi-fi-networks

https://dl.aircrack-ng.org/breakingwepandwpa.pdf

https://mrncciew.com/2014/08/19/cwsp-ccmp-encryption-method/

http://www.gabes.fr/wiki/index.php?title=SecuWifi#Le_protocole_CCMP

https://www.eetimes.com/diving-into-the-802-11i-spec-a-tutorial/

https://www.vocal.com/wp-content/uploads/2012/05/CCMP.pdf

https://dsimg.ubm-us.net/envelope/261982/453333/1702crash_file.pdf

https://cs.gmu.edu/~yhwang1/INFS612/Sample_Projects/Fall_06_GPN_6_Final_Report.pdf

https://cosc.canterbury.ac.nz/research/reports/HonsReps/2005/hons_0505.pdf

http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/blocs

http://www.dil.univ-mrs.fr/~jfp/master/m03/CompresChap5AES.pdf

https://www.utc.fr/~wschon/sr06/txPHP/aes/Key/Key.php

https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

https://www.di.ens.fr/~fouque/mpri/des-aes.pdf

http://math.univ-lyon1.fr/~roblot/resources/masterpro_chapitre_4.pdf

https://en.wikipedia.org/wiki/IEEE_802.11i-2004

https://www.wifi-professionals.com/2019/01/4-way-handshake

https://www.vocal.com/secure-communication/eapol-extensible-authentication-protocol-over-lan/

https://www.sstic.org/media/SSTIC2006/SSTIC-actes/Detection_d_intrusion_dans_les_reseaux/SSTIC2006-Article-Detection_d_intrusion_dans_les_reseaux-butti.pdf

RC4 is not Cryptographically Secure - Julian Bhardwaj
https://www.geeksforgeeks.org/rc4-encryption-algorithm/

https://tools.ietf.org/pdf/draft-kaukonen-cipher-arcfour-03.pdf

https://en.wikipedia.org/wiki/RC4