Travail d'Étude et de Recherche

Key Reinstallation Attacks :

Breaking WPA2 by forcing nonce reuse

González Boris Antonio, Ducrettet Damien

Encadré par M. Guermouche

Mai 2018

Contents

1 Introduction

La création des premiers réseaux sans fil (WiFi) tout public remonte aux années 1990 (1991-1997) avec l'arrivée des normes IEEE 802.11 afin de les structurer. Cette nouvelle technologie a ouvert de nouvelles perspectives aux moyens de communications. En effet, grâce à l'utilisation des ondes radio pour faire communiquer les appareils informatiques entre eux, de nouveaux réseaux ont pu être créés à partir de machines qui n'étaient pas directement connectées, par opposition aux connections filaires. Toutefois, le fait que les informations soient communiquées dans l'espace aérien, permet à tout récepteur présent dans cette zone d'avoir accès au trafic qui y circule. C'est pourquoi la sécurité de tels protocoles de communication a été rapidement un sujet important à étudier.

Comme nous allons le détailler dans une première partie, trois protocoles on été crée successivement afin de pallier à ces manquement. Pourtant l'année dernière, en 2017, une faille sur WPA2, soit le dernier est le plus sécurisé des protocoles, a été exposée à la conférence du Computer and Communications Security (CCS). C'est donc du fait de la nouveauté de ce problème que nous orientons notre TER sur l'étude de cette méthode générique d'attaques intitulée KRACK (Key Réinstallation Attacks).

Pour mener notre étude à bien, nous allons continuer en approfondissant en détail le fonctionnement de WPA2, basé sur son prédécesseur WPA. Dans cette partie, nous décrirons principalement l'architecture et le fonctionnement des clés dans ces protocoles. À la suite de quoi, nous ferons un bref état de l'art des différentes attaques déjà présente. Puis nous entamerons la descriptions du principe générale de l'attaque et différentes manières de l'implémenter. Toutefois, nous étudierons principalement le cas de la poignée de mains à quatre vois. Pour finir, nous regarderons comment la preuve de concept fonctionne. À partir de tout ce que nous verrons, nous serons donc en mesure à la fois de voir le fonctionnement général des protocoles de sécurisation et les détailles sur lequel peuvent reposer certain failles.

2 De WEP à WPA2

Le premier protocole permettant la sécurisation des réseaux sans fil remonte au mois de septembre 1999. Dénommé *Wired Equivalent Privacy* (WEP), celui-ci utilise une clé de n bits (principalement 64, 128 mais aussi 256) dont 24 seront toujours réservés au vecteur d'initialisation. Les messages devaient théoriquement pouvoir être envoyées de façon sécurisée à l'aide d'une opération **xor**, de l'emploi du protocole de chiffrement à flot **RC4**, et d'une fonction de hachage permettant le contrôle d'intégrité **CRC**. Pourtant, à partir de 2001, de nombreuses failles ont été découverts. La première attaque de Fluhrer, Mantin et Shamir, abordait le défaut, lié à la mauvaise utilisation de **RC4**, de la génération des nombres pseudo-aléatoires. Bien d'autres attaques ont été élaborées les années suivantes, ce qui a abouti à la conclusion qu'il était nécessaire de créer un autre protocole de sécurité. **WEP** sera officiellement abandonné en 2004 par la Wi-Fi Alliance.

L'apparition de **WPA** (Wi-Fi Protected Access) commencera en 2002 suit à l'échec de son prédécesseur. Cet autre protocole a été créé dans le but de faire une passerelle entre **WEP** et un autre protocole plus sécurisé. Développé de manière à fonctionner avec le même matériel que **WEP**, celui-ci sera mis à jour pour combler les principaux défauts de ce dernier. L'ajout du protocole **TKIP** (Temporal Key Integrity Protocol) a permis de rendre tolérable l'utilisation du matériel déjà présent, notamment en remplaçant **CRC**. Pour cela, le vecteur d'initialisation n'a plus été envoyé en clair, mais a été haché. De plus, les paquets se voient chiffrés (toujours par **RC4**) à partir de la combinaison d'une clef secrète, qui cette fois sera périodiquement modifiée, et du vecteur d'initialisation. Toutefois, cela n'étant toujours pas suffisant, une autre solution a dû voir le jour.

Depuis 2006 c'est l'emploi de **WPA2** qui reste le plus sécurisé à travers les réseaux sans fils communs et professionnel. La principale modification par rapport à **WPA** sera le remplacement du protocole **TKIP** par **CCMP** (Counter-Mode/CBC-Mac Protocol). Ce dernier peut utiliser l'algorithme de chiffrement par bloc AES (Advanced Encryption Standard process). Toutefois **WPA2** garde une compatibilité avec l'ancienne protocole afin de pouvoir fonctionner quelque soit le support matériel. Avec une bonne configuration il assure la meilleur sécurisation des réseaux que l'on dispose actuellement.

Solution	Protocole	Chiffrement	Intégrité
WEP	WEP	RC4	CRC
WPA	TKIP	RC4	Michael
WPA2	TKIP	RC4	Michael
WPA2	CCMP	AES	CBC

Table 1: Tableau des technologies WiFi

3 WPA2

Bien que le protocole ait changé, **WPA** et **WPA2** partage la même architecture. C'est à dire que la méthodologie de vérification et de création des clefs reste identique. En fin de compte, c'est le protocole qui gère les algorithmes de chiffrement et de contrôle d'intégrité des message qui varie. Évidemment, ces deux algorithmes aussi peuvent être amené à changer.

Il existe deux manière d'implémenter cette architecture. La première consiste à utiliser une clef partagé, on parlera alors de **WPA2** Personnel. Quand à la deuxième, à savoir **WPA2 Entreprise**, on utilise la norme **802.1x** et la structure réseau adéquate qui est spécifier et codifier dans cette même norme.

3.1 WPA Personnel

Pour déployer une sécurité **WPA** ou **WPA2**, le mécanisme le plus simple consiste à utiliser une clé partagée, ou Pre-Shared Key (**PSK**), configurée dans le poste de chaque utilisateur et dans chaque point d'accès (PA). Cette solution repose donc sensiblement sur le même principe que le **WEP** : le partage d'une même clé par tous les équipements.

Il faut saisir un mot de passe (aussi long que possible), appelé la « passphrase » et cette passphrase est passée automatiquement au travers d'une fonction pour générer la **PSK**. Nous allons parler de cette fonction plus tard. C'est une solution recommandée pour les particuliers ou les très petites entreprises car elle ne suppose l'installation d'aucun serveur d'authentification et elle offre un niveau de sécurité bien supérieur au **WEP**.

Malheureusement, la **PSK** a quelques défauts, par exemple si le mot de passe choisi est trop court, il peut être attaqué par force brute. Nous pouvons dire aussi que tous les utilisateurs partagent la même clé, ce qui augmente le risque qu'elle soit compromise et permet à tous les utilisateurs d'espionner le trafic.

3.2 Architecture 802.1x - Entreprise

Afin de garantir une meilleur sécurité, une architecture différente a été élaboré dans le cadre des milieux sensible telle que les entreprises.

La méthode recommandée pour déployer le **WPA** ou le **WPA2** est d'utiliser l'architecture 802.1x. Pour implémenter le protocole sera nécessaire de suivre les étapes suivantes:

- installer et configurer un serveur (normalement de type RADIUS) compatible avec le protocole **EAP**, dont on parlera plus tard.
- activer la gestion du 802.1x et du **WPA** (ou du **WPA2**) dans tous les points d'accès. Dans le cas du **WPA2**, il faut choisir le mode de chiffrement : **TKIP** ou **CCMP**.
- choisir une ou plusieurs méthodes d'authentification **EAP**, et les configurer dans le serveur.

Dans ce type de contexte trois acteurs sont à prendre en compte. On aura le client qui tentera toujours de se connecter au réseau privé. Puis on trouvera également un contrôleur d'accès (le point d'accès en générale) dont la principale fonctionnalité est d'interdire toute connection tant que l'identité du client n'a pas été validé. Enfin, la présence d'un serveur d'authentification sera nécessaire. Pour ce dernier sa mission consiste à vérifier les droits d'accès du client.

3.3 Hiérarchie des clefs

Afin de permettre une connexion complète, la spécification de l'architecture **WPA** et **WPA2** préconise de passer par trois étapes. De celle-ci, on peut en extraire une série de clef *(passphrase, PSK, PMK, PTK)* qui permet d'apercevoir ce qu'on appel la hiérarchie des clefs.

- **L'association WIFI :**

  Cette premier étape a pour objectif de permettre au client de détecter la préséance des machines qui lui sont potentiellement accessible (authentification des supports réseau). Après avoir trouvé le PA la plus proche grâce à son *SSID*, le client va émettre des requêtes dans le but de confirmer l'association. Une fois l'accorde de la borne obtenus, elles pourront continuer à échanger par la suite. Il est nécessaire de préciser qu'à ce moment, le point d'accès ne donne toujours pas accès au réseau.

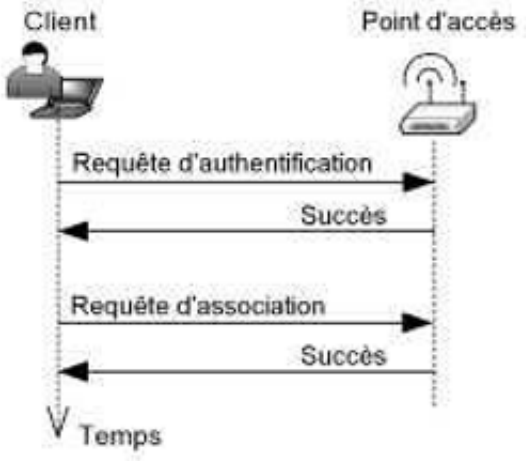

Figure 1: L'association WiFi [1]

- **L'authentification du client :**

  Le rôle de cette étape est de vérifier l'identité du client, de savoir si il possède le droit ou non d'accéder au réseau privé. Il existe deux manières distinctes de gérer l'authentification selon si l'on utilise le mode personnel ou entreprise (architecture **802.1x**) de **WPA** et **WPA2**. Dans le première cas celle-ci sera effectué lors du contrôle de la clé partagé. Au préalable on aura nécessairement rentré la passphrase (mot de passe). Dans le deuxième, cela doit être réalisé par un tiers. Nous donneront de plus une explication de tous ceci dans les parties suivantes. Nous aurons comme résultat de l'authentification du client un clé maître ou *Pairwise Master Key* (**PMK**).

- **La négociation des clés temporaires :**

  Pour ce qui est de la troisième étape, son objectif consiste à partager entre le client et le point d'accès la clé temporaire (**PTK**). Une fois celle-ci obtenus (par dérivation de la **PMK**), tout les messages qui suivrons pourrons être chiffré dans un canal qui ce veux sécuriser.

  Le client et le PA se mettent maintenant d'accord sur une nouvelle clé dérivée de la clé **PMK** : cette clé temporaire s'appelle la *Pairwise Transient Key* (**PTK**). Ce faisant, ils en profitent pour vérifier qu'ils possèdent bien la même clé de départ **PMK**. Dorénavant, tous leurs échanges sont chiffrés avec la clé **PTK** : un tunnel sécurisé est en place.

  Grâce à ce tunnel, le PA peut envoyer secrètement au client la clé qu'il utilise pour chiffrer le trafic en utilisant un paquet **EAPoL-Key** contentant la **GTK** chiffrée. Main-

tenant, le client peut également déchiffrer le trafic de groupe.

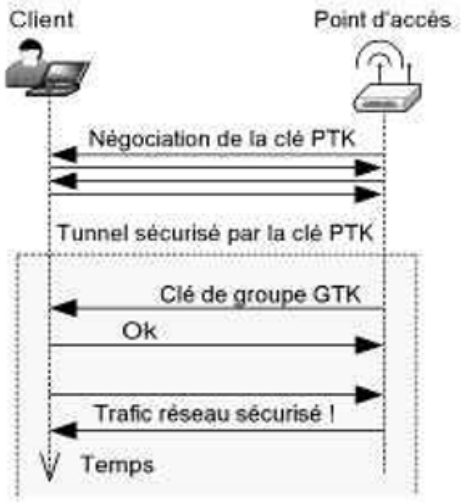

Figure 2: Négociation de la PTK [1]

On aura besoin de mieux comprendre le fonctionnement des clés, par exemple la clé maîtresse **PMK**, négociée au cours de l'authentification ; si l'on utilise la solution de la clé partagée à l'avance (**PSK**), plutôt que l'architecture 802.1x, alors la clé **PSK** est utilisé dans une fonction dont résultat sera utilisé comme **PMK**. On a aussi la clé temporaire **PTK**, dérivée de la **PMK** dans le *Four Way Handshake*, on va approfondir ce sujet après. Et pour finir on a la clé temporaire de groupe **GTK**, générée par le PA.

En réalité, lorsque le point d'accès décide de générer une nouvelle clé de groupe, il génère d'abord aléatoirement une clé de 128 bits appelée la *Group Master Key* (**GMK**). Il dérive ensuite la **GTK** à partir de la **GMK**. La raison pour laquelle une clé temporaire (**PTK, GTK**) est dérivée à partir de la clé maîtresse (**PMK, GMK**) est que cela permet d'éviter que la clé maîtresse soit directement utilisée pour les opérations de chiffrement, ce qui la rendrait potentiellement vulnérable à des attaques.

Grâce à ce bref aperçu des étapes menant à une connexion complète, on est plus à même de comprendre le schéma de la figure 3, qui montre la hiérarchie des clés dans le protocole WPA/WPA2.

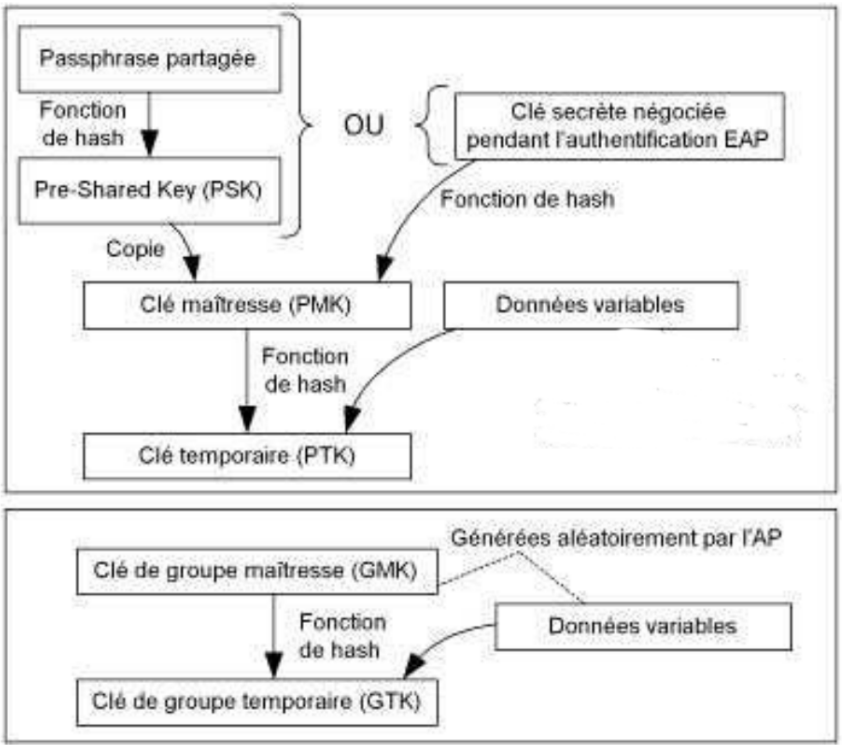

Figure 3: Hiérarchie des clés [1]

3.3.1 EAP et EAPoL

EAP *Extensible Authentication Protocol* est un protocole dont le rôle consiste à identifier les utilisateurs avant de les laisser entrer sur les réseaux. La sécurité de **WPA** et **WPA2** dépend directement d'**EAP**. Le principe est très simple: si un client cherche à accéder à un réseau, un contrôleur d'accès lui barrera le chemin jusqu'à ce qu'il s'identifie auprès du serveur d'authentification. Le contrôleur d'accès sert d'intermédiaire pour la communication entre le client et le serveur d'authentification.

Exemple de configuration du protocole EAP dans l'architecture 802.1x:

- le client possède une connexion WiFi compatible avec **EAP** et supporte 2 méthodes d'authentification: **PEAP/MS-CHAP-2** et **EAP/TLS**.
- le contrôleur d'accès est un PA compatible avec **EAP** (il n'a pas besoin de connaître les méthodes d'authentification). Il est toutefois capable de relayer les requêtes **EAP** vers le client et vers le serveur d'authentification.
- le serveur d'authentification est un serveur **RADIUS** avec **EAP** qui dispose des méthodes d'authentification **EAP/TLS** et **TTLS/PAP**. Le serveur demandera au client à s'identifier selon une méthode. Si le client ne la gère pas, le serveur en suggérera une autre et ainsi

de suite jusqu'à ce que le client en accepte une. (dans cet exemple le client va accepter le **EAP/TLS**)

Il est possible d'utiliser ce protocole avec *TCP/IP, UDP/IP* ou directement dans les paquets WiFi. Afin que **EAP** soit mieux adapté au support physique de la WiFi, il a été modifie de façon à ce qu'il donne un nouveaux protocole nommé **EAPoL** (EAP over LAN). Ce dernier est définit au sein du standard **802.1x**. Avec **EAPoL** un certain nombre de types de messages est disponible:

- **EAPoL-Start:** permet au client de prévenir le contrôleur d'accès qu'il souhaite se connecter.
- **EAPoL-Packet:** ce sont les paquet qui en-capsulent les paquets EAP.
- **EAPoL-Key:** permet l'échange des clés de chiffrement.
- **EAPoL-Logoff:** permet au client de demander la fermeture de sa session.

La composition des paquets **EAPoL** est décrit dans le tableau ci-dessous :

MAC	**Version**	**Type**	**Longueur**	**Message EAPoL**
30 octets	1 octet	1 octet	2 octet	n octet

- **Version:** version du protocol EAPOL utilisé
- **Type:** s'il s'agit d'un paquet Start, Key, ou Logoff.
- **Longueur:** la longueur du message qui suit

À noter que **EAP** et donc **EAPoL** peuvent utiliser différentes méthodes d'authentification. Les plus connu étant **EAP/PEAP, EAP/TTLS, EAP/FAST**. Elles sont tous basé sur l'emploi d'un tunnel sécurisée, généralement c'est à **TLS** (*Transport Layer Security*) que l'on confit cette tache. Dans un premier temps, après une poignée de main initial, le client reçoit un certificat de la part du serveur. La machine de l'utilisateur pourra utiliser la clé publique contenue dans ce certificat afin de chiffrer le message qui contiendra sa clé symétrique. Seul le serveur pourra donc déchiffrer ce message, et donc récupérer la clé. De ce fait, le reste de la communication pourra être chiffré de façon sur.

3.3.2 Authentification 802.1X

WPA2 entreprise utilise donc l'architecture **802.1x** et les requêtes de type **EAPoL** pour permettre une meilleur sécurité. Cette authentification va se dérouler selon les étapes suivantes:

- Premièrement, le poste utilisateur envoie une requête **EAPoL Start** au point d'accès. C'est le signal nécessaire pour démarrer la séquence d'authentification auprès du serveur dédié à cette fin.
- Deuxièmement, un canal sécurisé va être ouvert via une méthode spécifique entre le client et le serveur d'authentification.

– Troisièmement, si cette méthode est génératrice de clé, comme c'est le cas avec **PEAP** par exemple, alors le client et le serveurs vont se mettre d'accord pour créer une clé de 256 bits dérivée de la clé secrète. La nouvelle clé ainsi générée sera appelé **PMK** (*Pairwaise Master Key*).

– Quatrièmement, ce même serveur va transmettre la **PMK** au point d'accès pour que les futures échanges entre celle-ci et la machine cliente soient placés dans un tunnel sécurisé.

– Cinquième, pour faire cette échange, une dernière requête du type **EAPoL** succès est transmise. Une fois la réception effectué par l'initiateur de la connection, alors tous les étapes de l'authentification sont validés.

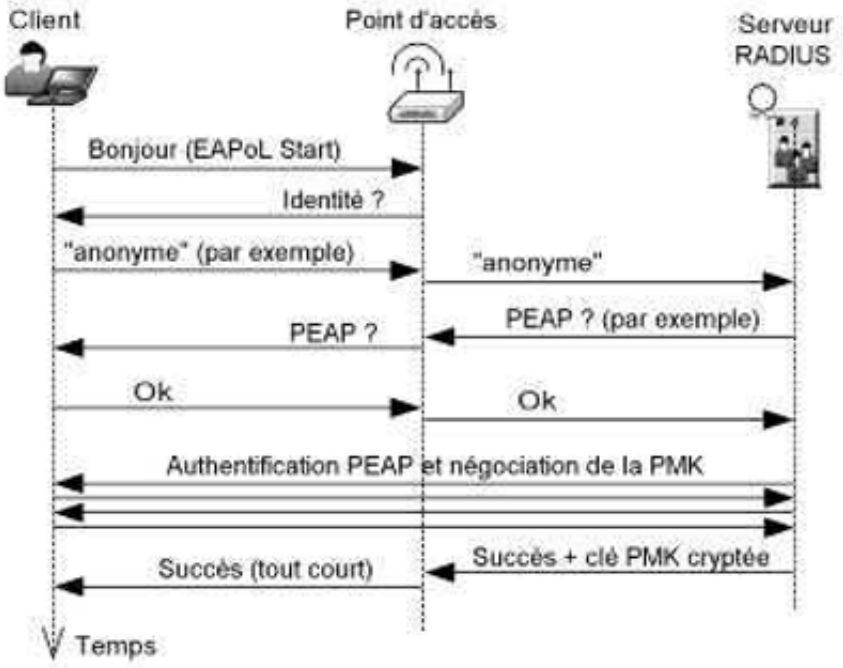

Figure 4: Authentification 802.1x [1]

Comme on peut le remarquer, la grande force de cette technique réside dans l'emploi du tunnel chiffré. Il est donc d'abord nécessaire de trouver une faille dans ce système pour pouvoir par la suite mener d'autre attaque. C'est donc pour cela que nous restreindrons notre domaine de recherche au mode opératoire de l'attaque **KRACK** exclusivement sur le mode personnel de **WPA2**.

3.4 WPA2 Personnel, de la PassPhrase à la PMK

Nous avons vus précédemment comment le mode entreprise de **WPA** permet d'obtenir une clé **PMK**. Toutefois dans le cadre de **WPA2** personnel, la méthode diffère et nécessite une autre clé.

Un utilisateur qui veut se connecter à un réseau via **WPA** ou **WPA2** personnel, doit systématiquement rentrer une passphrase. Celle-ci doit être dérivé en **PSK**, et dans ce cadre, la **PMK** final n'est que la copie de la **PSK** (d'autre présentation prefere dire que la passphrase est la **PSK** que l'on change en **PMK**, ces deux explication sont équivalentes).

Cette transformation de la passphrase en **PSK** se fait à l'aide d'une méthode nommé *Password-Based Key Derivation Function* (**PBKDF2**). Cinq paramètres doivent être fournit à une telle fonction pour qu'elle soit applicable.

PSK = PBKDF2 (MAC-SHA1, Passphrase, SSID, 4096, 256)

Maintenant nous allons expliciter un peu plus en détaille comment fonctionne cette égalité. Le but d'un tel procéder est d'obtenir un hash de la longueur souhaité qui découle des informations connues par les machines qui veulent communiquer entre elle. La procédure **PBKDF2** va donc appliquer la fonction de hashage **MAC-SHA1** 4096 fois sur les paramètres commun que sont la passphrase et le SSID. Le dernier paramètre est la taille que l'on souhait obtenir après l'opération effectué, car par défaut **MAC-SHA1** retourne un clé de 160 bits. Une fonction d'extension sera appliqué à cette effet.

Regardons de plus près comment est implémenté **MAC-SHA1**. Tout d'abord il faut se rappeler que les **MAC** (*Message Authentication Code*) sont une catégorie de message qui sert de code d'authentification et qui vérifient aussi l'intégrité des donnée. **HMAC** est le *keyed-hash* de ce type de message. N'importe qu'elle fonction itérative de hachage peut être utilisé à cette fin, comme **MD5** ou **SHA1**, ce qui donne respectivement **HMAC-MD5** ou **HMAC-SHA1**. Dans un premier temps le message va être divisé en blocs selon sa taille. Puis les opération de **SHA1** vont pouvoir être appliqué.

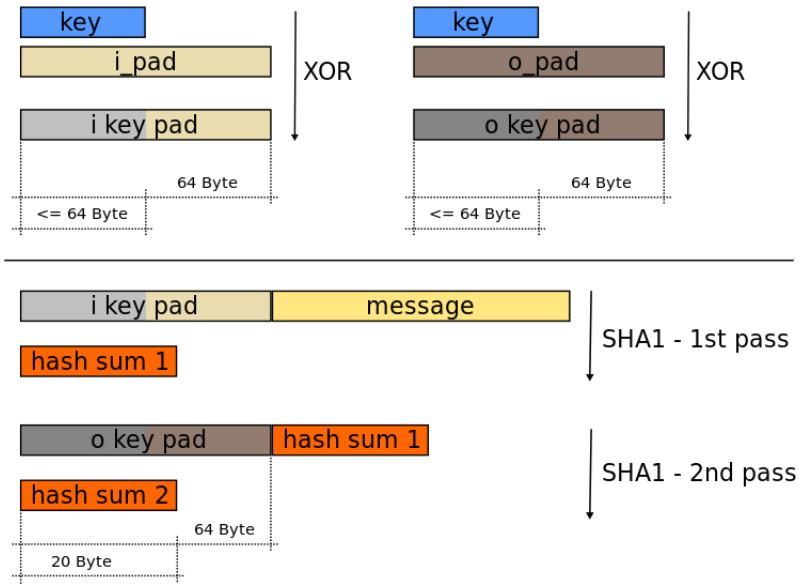

Figure 5: Schema du fonctionnement de la fonction HMAC

On définit la fonction **HMAC** comme il suit:

$$\mathbf{HMAC} = \mathbf{H(\ (K\ \hat{}\ opad)\ ||\ H((K\ \hat{}\ ipad)\ ||\ M)\ )}$$

Avec :

- **H** : la fonction de hachage itérative
- **K** : la clé secrète
- **M** : le message à authentifier
- Ou || est une concaténation et ˆ la fonction **xor**
- **ipad** et **opad** sont des blocs définies par ipad = 0x3636...36 et opad = 0x5c5c...5c

Dans notre cas, c'est le SSID du réseau qui va servir de clé. Le message à transmettre sera dans un premier temps la passephrasse puis ce sera le résultat des opérations précédentes. On voit que le résultat ne dépend que du SSID et de la passephrasse, donc le résultat final est prévisible si on connaît au préalable ces deux éléments. Ainsi après extension de ce hash, le client aura réussi à calculer la **PSK**. L'obtention de la **PMK** se fera par la simple copie de la **PSK**.

3.5 Poignée de main à quatre voies

Une fois que le client et le point d'accès ont calculé chacun de leur côtes la **PMK**, il faut qu'il vérifie que ces deux clés soient bien identiques. Dans le mode personnel c'est sur cette seul confirmation que se fait l'authentification du client. Puis ils devront se mettre d'accord sur une clé temporaire **PTK** *(Pairwise Transient Key)* qui assura le chiffrement des échanges. La poignée de main à quatre voix (Four Way Handshake - 4WH) a donc cette double fonctionnalité. Qui plus

est, une contrainte supplémentaire doit être respecté, à savoir qu'il faut pouvoir générer différents **PTK**, une par session, toujours à partir de la même **PMK**. Il est donc évident que cette échange est particulièrement important pour la sécurité. C'est pourquoi l'implementation de celui-ci est fait de manière à ce que la **PMK** ne soit jamais transmise directement. Nous allons maintenant expliquer en détaille son fonctionnement.

Avant de continuer plus loin il faut savoir que la **PTK** est le résultat d'une fonction de hachage auquel on fournie en entré cinq arguments. Le premier est la clé maîtresse **PMK**, ce qui fait le lien entre la passphrase initiale qu'a tapé l'utilisateur. Puis viennent les deux adresses **MAC** appartenant au client et à la borne. Mais comme ces trois valeurs son fixe, elles ne permettent pas la présence de l'aléa nécessaire à la satisfaction de la contrainte vus au-dessus. C'est pourquoi les deux arguments manquant sont des nonces, des valeurs généré pseudo-aléatoirement par les deux machines. Celle du point d'accès est appelé **ANONCE** *(pour Access Point Nonce)* et celle du client **SNONCE** *(pour Supplicant nonce)*.

Le PA et le client ont déjà connaissance de la **PMK**, et sont également en possession des deux adresse physique notamment grâce à l'échange antécédent. Chacune des machines peut calculer son propre nonce. Il ne reste plus qu'à les échanger et à installer la **PTK**. Le 4WH va procéder à ceci de la manière suivant :

- Le point d'accès envoie sont **ANONCE** au client, en clair.
- Comme le client possède tous les éléments utiles à la fonction de hachage, il peut générer la **PTK** de son côté. Ensuite, via un message du type **EAPoL**, il va transmettre un message contenant son propre nonce. Un deuxième champ complet ce message; il s'agit d'un code de contrôle d'intégrité généré grâce à la clé d'intégrité **EAPoL** présente dans la **PTK**. Une description de la décomposition de la **PTK** est donné dans le chapitre suivant.
- De même le PA va commencer par calculer la **PTK**. À partir de cette dernière, en regardant dans le premier champ qui compose la clé temporaire (clé d'intégrité **EAPoL**), elle va pouvoir vérifier le code d'intégrité envoyé par le client. Si ce code est bon, le PA sait que le client possède la bonne **PTK**, donc la bonne **PMK**. Sinon, c'est l'échec de la connexion car l'authentification est invalide. En suite elle envoi un message, du type accusée de réception *(ACK pour acknowledgement)*, afin d'annoncer que tout est opérationnelle de son côté. Elle ajoute à ce dernier sont propre message **MAC** (message d'authentification).
- Pour finir, le client contrôle à son tour le code d'intégrité en provenance du PA en vus de s'assurer qu'elle possède aussi la bonne **PMK**. Puis il va pouvoir installer la clé **PTK**. Et pour finir il retournera également un ACK pour que la borne puisse aussi installer la clé temporaire, ce qui ouvrera la nouvelle session.

L'échange de ces quatre messages forment donc ce que l'on appelle le four-way handshake. Après la dernière transmission effectué, les deux machines chiffreront tous les messages de la session.

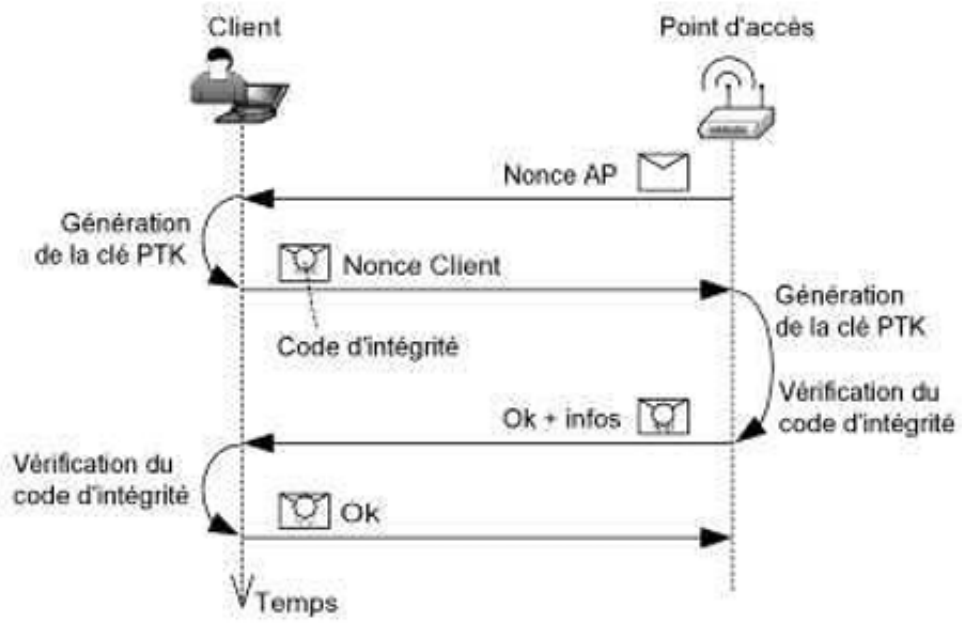

Figure 6: La poignée de main à quatre voix [1]

3.6 Protocole de chiffrement et d'authentification sous WPA/WPA2

Rappelons que **WPA** utilise comme protocole de chiffrement et d'authentification **TKIP**. Quand à **WPA2**, bien qu'il soit aussi compatible avec **TKIP**, privilégie **CCMP** qui est plus performant. Les clés temporaires, à savoir **PTK** et **GTK** sont découpées en morceaux pour donner d'autres sous-clés. Le besoin de cette décomposition s'explique par le fait que l'on doit à la fois chiffrer et faire le contrôle d'intégrité des paquets. Toutefois ce découpage varie selon si on utilise **TKIP** ou **CCMP**, car ils n'utilisent pas les même algorithmes de chiffrement et de vérification.

3.6.1 Décomposition de la clé PTK sous TKIP

Par le biais du dialogue qu'est le four-way handshake, une **PTK** de 512 bits est dérivée de la **PMK** initiale qui contenait 256 bits. La **PTK** est composée de quatre sous-clés de 128 bits chacune:

PTK			
Intégrité EAPoL	Chiffrement EAPoL	Chiffrement du tunnel	Intégrité du tunnel
128 bits	128 bits	128 bits	128 bits

C'est le trafic de paquets **EAPoL** entre le client et le PA qui nécessite l'utilisation des deux premières clés. Les deux suivantes vont être exploités pour le reste du trafic (après le 4WH), ce

qui fait d'elles les clés les plus utilisés.

La dérivation de la **GMK** de 128 bits en **GTK** de 256 bits se fait simplement en utilisant un algorithme de hachage. Tout comme précédemment elle est découpée, mais cette fois ci en seulement deux clés.

GTK	
Chiffrement des données	Intégrité des données
128 bits	128 bits

La clé **GTK** permet la protection du trafic broadcast et multicast envoyé par le PA.

3.6.2 Décomposition de la clé PTK sous CCMP

CCMP a l'avantage d'utiliser la même clé pour le chiffrement et le contrôle d'intégrité des données. Par conséquent les clés **PTK** et **GTK** sont plus courtes que celle de **TKIP**. Dans ce cas la **PTK** ne fait plus que 384 bits.

PTK		
Intégrité EAPoL	Chiffrement EAPoL	Chiffrement et intégrité
128 bits	128 bits	128 bits

Comme dans le cas précédant, les deux premières clés servent à protéger et vérifier tout le trafic **EAPoL**. La clé d'Integrité **EAPoL** et la clé de chiffrement **EAPoL** sont appelés *Key Confirmation Key* (**KCK**) et *Key Encryption Key* (**KEK**) respectivement. Les 128 bits suivant est donc la clé qui sert à la fois au chiffrement et au contrôle d'intégrité pour le reste du trafic entre le client et le PA, aussi appelé *Temporal Key* (**TK**).

Avec ce protocole, la **GTK** n'a plus qu'une longueur de 128 bits. La seul clé qui la compose permet à la fois le chiffrement et la vérification du trafic de groupe émis par le PA.

GTK
Chiffrement et intégrité
128 bits

3.6.3 Protocole CCMP

En vus de l'étude de l'attaque sur **WPA2**, et comme le protocole **CCMP** est amené à être le plus utilisé, nous allons maintenant regarder de plus près comment celui-ci fonctionne.

1. **AES**

   Une des grands spécificité de **CCMP** est l'emploi de **AES** (*Advanced Encryption Standard*) comme algorithme de chiffrement. La renommer de ce dernier n'est plus à faire,

il est le plus puissant qui existe à ce jour. En effet, sa sécurité est telle que son emploi est fréquent jusque dans les agences gouvernementaux. Qui plus est, ces performances sont aussi enviable, car l'algorithme n'utilise que des opérations simples comme les additions, le décalages de bits ou encore le XOR.

AES implémente le chiffrement par blocs. Il prend un bloc de 128 bits et à l'aide d'une clé de chiffrement (de même taille) il fabrique un nouveau bloc chiffré. Ce résultat a un aspect aléatoire, ce qui est l'une des forces de **AES**. Une autre avantage des algorithmes de chiffrement par bloc est qu'on ne peut pas savoir à quel bit chiffré correspond tel bit non chiffré. En d'autres termes, si l'on modifie un seul bit du message non chiffré, alors le message chiffré sera peut-être entièrement différent (ou en tout cas, au moins un bloc sera différent).

2. **Counter-Mode**

On peut voir **AES** comme une fonction de chiffrement, dans ce cas il faut que le protocole **CCMP** puisse définir le cadre dans lequel on l'utilise. Les "modes" ont justement ce but, c'est à dire que ce sont des stratégies d'utilisation pour permettre le chiffrement et il spécifie également l'algorithme qu'on doit utiliser pour le contrôle d'integrité. Dans notre contexte le choix a été porté vers l'un des plus répandu de ce domaine, à savoir CM (*Counter Mode*). Son principe est le suivant:

- Un compteur est incrémenté en permanence.
- Les valeurs résultantes de ce compteur sont chiffrés par un algorithme dédié à cet effet. Pour cela on utilisera un clé qui doit être préalablement connu.
- On obtient un flux infini de bits pseudo-aléatoires, et il est combiné à l'aide de l'opération XOR au message.

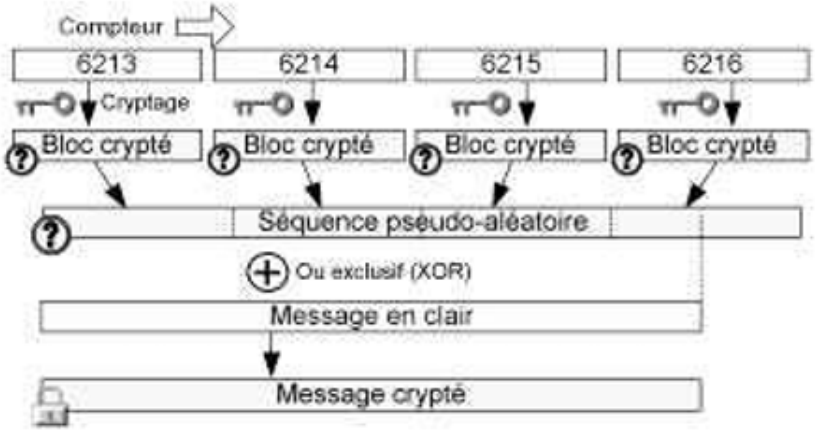

Figure 7: Counter-Mode [1]

Counter Mode apporte quelques avantages, la taille du message à chiffrer n'a plus besoin d'être un multiple de la taille du bloc. En outre, l'opération de déchiffrement est strictement identique à l'opération de chiffrement, ce qui permet de ne pas avoir à mettre en œuvre la fonction de déchiffrement de l'algorithme par bloc utilisé.

Malheureusement, avec cette démarche on réintroduit un problème connu, lié au fait que les bits sont chiffrés un à un et ça peut engager le message. Il faudra donc que l'algorithme qui assure le contrôle d'intégrité soit très sûr. Le protocole **CCMP** va donc s'assurer de cela grâce a **CBC**.

3. **CCM**

Le mode **CCM** (*Counter-Mode + CBC-MAC*) a été inventé par le groupe de travail **802.11i**. Le chiffrement est assuré par le Counter-Mode, et l'algorithme de contrôle d'intégrité à employer est **CBC-MAC**. C'est la réutilisation de la clé à la fois dans le chiffrement et dans la vérification qui fait que la **PTK** qu'on utilise pour **CCMP** ne possède plus que un seul et même champ pour sécuriser les données (non **EAPoL**). Un élément important est ajouté à **CCM**, un numéro de paquet séquentiel (on l'appelle aussi nonce ou PN) qui ajoute de l'aléa lors du chiffrement. Le but de celui-ci étant d'assurer que deux messages identiques d'origine ne donnent pas le même résultat.

Maintenant, étudions de plus prêt en quoi consiste **CBC-MAC**. Tout d'abord c'est l'acronyme de *Cipher Block Chaining - Message Authenticator Code*. À noter que dans le contexte qu'est la sécurité, l'abréviation **MAC** n'a pas de lien avec la couche réseau du même nom. L'implémentation de **CBC** est faite de la manière suivante :

- Le premier bloc du message est chiffré avec **AES**.
- Le résultat qui en découle est combiné via un XOR avec le bloc en claire suivant.
- Tout ceci est pour finir une nouvelle fois chiffré avec **AES**. L'opération se répète ainsi bloc par bloc.

Bien que un peu plus exigeant en calcule que **Michael** (utilisé par **TKIP** notamment), il possède plusieurs avantages. Sa valeur est totalement imprévisible, et si un seul bit du message initial est modifié alors tout le code **CBC** change. Par conséquent il est un excellent code de contrôle d'intégrité. Mais il ne peut fonctionner que si le message a une longueur égale à un multiple de la taille des blocs. Le protocole **CCMP** résout ce problème en complétant le message avec des zéros jusqu'à obtenir une taille adéquate.

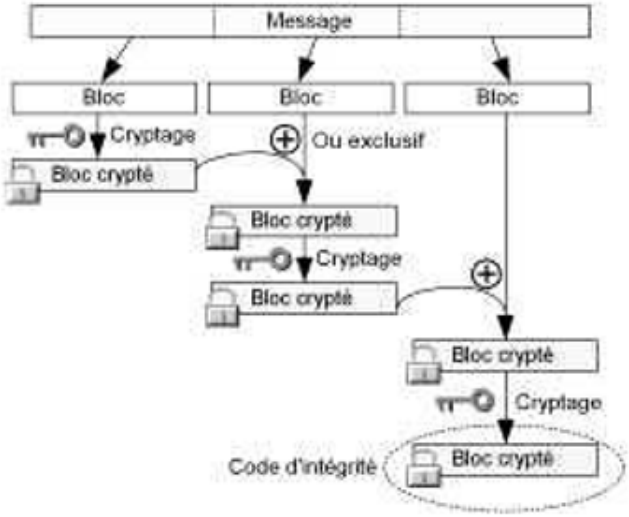

Figure 8: Code d'intégrité CBC [1]

4. **CCMP:**

Malgré le fait que **CCM** soit crée dans le cadre de la norme **802.11i**, il reste assez général sur la manière dont il doit être utilisé. C'est pourquoi le protocole **CCMP** (*CCM Protocole*) a été rédigé, il spécifie son emplois dans le contexte applicatif de la WiFi.

Pour comprendre **CCMP** de façon plus approfondi, il est nécessaire de rappeler succinctement comment les paquets envoyés sont traités par la couche MAC. Les paquets fournit à cette couche réseau (*MSDU*) vont être découpés en plusieurs petits paquets (*MPDU*). Ces dernier composé d'une en-tête MAC et de données. Qui plus est, avec **CCMP** une autre en-tête est ajouté, elle contient le numéro de paquet **PN** (pour *packet number*) ainsi que l'index de la clé **PTK** ou **GTK** utilisé lors du chiffrement. Ce nouveaux segment se verra rajouté entre l'en-tête MAC et les données chiffrés.

En vérité, seul la moitié du code généré par **CBC** est conservée. Cette partie utile est appelé code d'intégrité ou **MIC** (*Message Integrity Code*), et elle est inséré à la suite des données chiffrés. On peut donc résumer la structure d'un paquet chiffré **WPA2** avec **CCMP** par le schéma suivant:

En-tête MAC	**En-tête CCMP**	**Données chiffrés**	**MIC chiffré**	**CRC**
30 octets	8 octets	0 à 2296 octets	8 octets	4 octets

Une spécificité à **CCMP** est que le code d'intégrité **MIC** est issu d'un calcule sur l'ensemble de message incluant les en-tête **CCMP** et **MAC**. Ainsi si deux stations utilisent

la même clé et le même **PN**, elles ne généreront pas le même code d'intégrité. Un autre de ces effets positif, est que cela permet de protéger ces champs contre d'éventuelles modifications malveillantes, ce qui était une des failles exploitables des anciens protocoles.

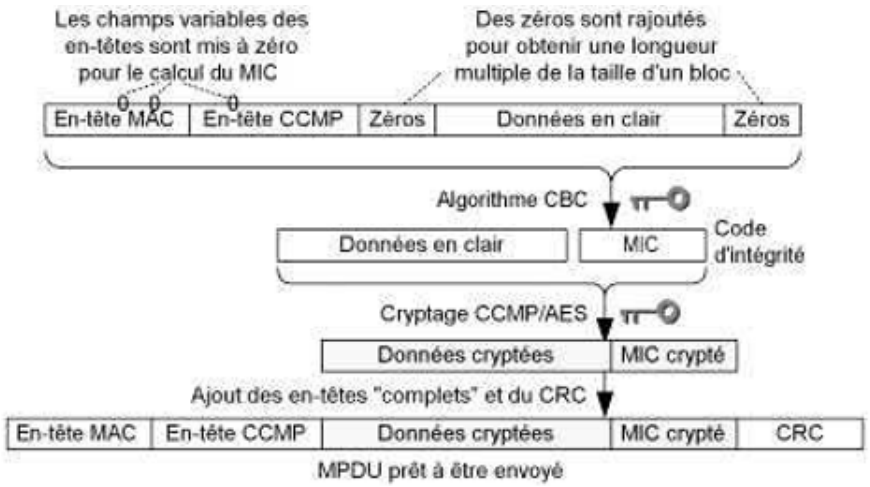

Figure 9: Construction du paquet par CCMP / AES [1]

Depuis quelques années, et après l'implémentation du protocole **WPA2**, le protocole **CCMP** est devenu le protocole de chiffrement le plus utilisé dans les réseaux sans fil dans le monde. On a vu que le **CCMP** est basé sur l'algorithme de chiffrement **AES** en utilisant le mode **CCM** (*Counter Mode* avec **CBC-MAC**). La sûreté de ce protocole est assuré aussi longtemps que le compteur ou le vecteur d'initialisation (IV) ne soit pas réutilisé avec la même clé dans le Counter Mode.

Dans **CCMP**, le compteur ou **IV** est la concaténation de l'adresse MAC de la machine qui envoie le message, un nonce de 48 bits, et des autres flags qui sont dérivés du paquet. Le nonce est aussi utilisé par le récepteur du paquet comme le compteur du message (on verra ça de façon plus détaillée après), qui va être incrémenté de 1 après l'envoi de chaque paquet, et initialisé à 0 au moment d'installer la **PTK**, ce qui va nous assurer que le **IV** ne soit pas répété. La **PTK** est utilisé pour protéger directement la communication dans les deux sens.

La re-utilisation d'un même nonce abouti à une faille de sécurité, c'est pourquoi le bonne fonctionnement du protocole tant à éviter cette situation, mais l'attaque que nous étudierons se base justement sur ce principe.

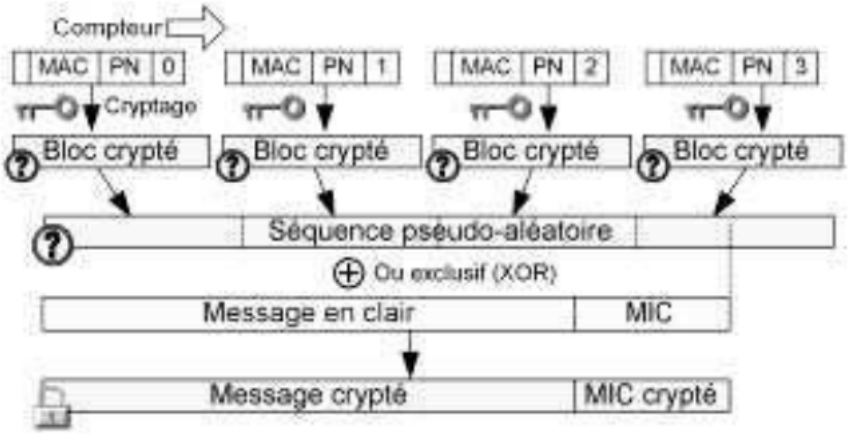

Figure 10: Le chiffrement de CCMP [1]

4 Étude des failles dans le protocole

Plusieurs vulnérabilités ont été découvertes dans **WPA/WPA2**, certaines plus dangereux que d'autres. Normalement, les attaques qui sont faites contre les réseaux qui implémentent ces protocoles, sont basées sur la force brute ou sur des vulnérabilités liées à son contexte. En éffet le protocole et ses parties cruciales ont été prouvé sûrs mathématiquement, comme par exemple la génération de la **PTK**. Maintenant nous allons présenter l'état de l'art concernant les attaques et les failles les plus répondus.

4.1 Attaques sur WPA/WPA2

1. **Mots de passe faibles**

   Si nous utilisons **WPA/WPA2** personnel, les clés pre-partagés, ou mots de passes que nous utilisons pour nous connecter aux réseaux, sont vulnérables aux attaques de dictionnaire si elles sont faibles. Le principal outil pour mener ce type d'attaques est généralement *Aircrack*. Pour se protéger nous pouvons changer le nom du SSID, et évidemment configurer des mot de passe plus forts, avec au moins 12 caractères minuscules, majuscules ou caractères spéciaux.

2. **Spoof et déchiffrement des paquets WPA**

   Le principe de cette attaque consiste à injecter un nombre arbitraire de paquets dans les réseaux qui utilisent le protocole **WPA-TKIP** afin de se faire passer pour la borne. Après quoi, on peut déchiffrer arbitrairement des paquets qui sont envoyés vers le client. Cette attaque ne récupère pas la clé de session (**PTK**) entre le client et le PA. Toutefois la principal limite de cette attaque est qu'il est nécessaire d'appartenir à l'un de ces réseaux au préalable.

3. **Récupération du PIN WPS**

   En 2011 des chercheurs ont trouvé une vulnérabilité qui affecte les routeurs sans fil avec **WPS**, indépendamment de la méthode de chiffrement implémenté. La plupart des modèles récentes ont cette fonctionnalité activé par défaut.

   WPS a été créée pour diminuer le risque que les clients créent des mots de passe faibles, donc les fabricants ont implémenté des méthodes alternatives dans les routeurs pour générer et distribuer la clé. Par exemple on peut introduire un PIN de 8 digits ou appuyer sur un bouton dans le routeur. De toute façon, le PIN a une vulnérabilité qui va permettre à des attaquants de récupérer le **WPS PIN** et ainsi le mot de passe **WPA/WPA2** du réseau.

4. **HOLE 196**

   Trou 196 est une vulnérabilité dans le protocole **WPA2** qui abuse de la **GTK** (*Group Temporal Key*). Il peut être utilisé pour faire une attaque *Man in the Middle* et un Denial of Service **DoS**. Pour qu'elle fonctionne on a besoin que l'attaquant soit déjà authentifié. Pour

que ça marche, l'attaquant doit envoyer un message avec la **GTK** mais dirigé à une adresse MAC défini, au lieu d'un adresse broadcast. Ainsi on aura une empoisonnement d'IP qui va permettre que le PA soit supplanté.

5. **Chop-Chop**

   Le principal attaque à TKIP s'appelle CHOP CHOP et il ne s'agit pas de récupérer les clés. Cette attaque a été originalement implémenté sur WEP et il permet à l'attaquant de déchiffrer les N dernières octets des paquets chiffrés, en envoyant environ N*128 au réseaux. Il est basé sur la faille du checksum CRC32 appelé ICV qui est ajouté à la partie data du paquet. L'attaquant tronque le dernier octet du paquet chiffré, il essaie de deviner la bonne valeur du dernier octet et il retourne le paquet au point d'accès.

   Si ce n'est pas correct, le paquet va être jeté, grâce au checksum incorrect, et comme ça l'attaquant peut savoir s'il l'a deviné. Une fois que l'attaquant a bien deviné la valeur pour le dernier octet, il peut continuer en arrière vers les autres octets du paquet. Ça prend environ 128 suppositions par octet.

   Toutefois, comme le MIC et les replay counters sont ajoutés dans WPA, cet protocole peut éviter cette manière d'implémenter l'attaque, sauf qu'il existe une autre manière de le faire. L'attaquant peut capturer les paquets et trouver un canal à faible trafic, dans lequel le replay counter soit encore bas et essayer de faire l'attaque. S'il devine pas bien le dernier octet du paquet le PA va le rejeter, mais s'il a bien deviné donc un message d'échec MIC va être envoyé vers le client. Quand le client reçoit cet message, l'attaquant peut savoir que la supposition était correcte et il doit attendre au moins 60 secondes pour faire une autre supposition, pour éviter que le client soit deconnecté.

   Dès que le client a bien déchiffré les 12 dernières octets, il va avoir le MIC et le ICV en clair. Avec le ICV, l'attaquant peut deviner le reste du paquet et exécuter le CRC32 jusqu'à obtenir un match entre les valeurs et ainsi savoir qu'il a bien déchiffré le paquet.

6. **KRACK Attack :**

   En 2017, **Mathy Vanhoef, PhD.** un chercheur de l'université *KU Leuven*, Belgique, a trouvé la première vulnérabilité importante dans le protocole **WPA2** et plus spécifiquement dans le 4 Way Handshake. Nommé **Key Réinstallation Attack - Nonce Reuse (KRACK)**, il s'agit d'exploiter une vulnérabilité qui réinstalle la clé de session **PTK** dans le 4 Way Handshake au moment de faire la connexion entre le client et le PA. Ainsi on peut déchiffrer, rejouer et/ou forger des paquets, en function du système d'exploitation et du contexte de l'attaque. Et c'est donc cette attaque que notre travail d'étude et de recherche étudie.

5 KRACK Attack

Le protocole **WPA2** est le plus utilisé aujourd'hui et il a été prouvé que son coeur cryptographique était sûr. Mais si une attaquant souhaite compromettre une victime il peut exploiter plusieurs méthodes, dont l'une d'elle est appelé **KRACK Attack**. Plus concrètement, les attaquants peuvent implémenter cette attaque pour lire l'information qui a été sécurisé par chiffrement. Ça peut être utilisé pour voler des informations sensibles comme les mots de passes, numéros des cartes bancaires, messages, etc. À noter que l'attaque fonctionne pour presque tous les réseaux WiFi modernes.

Cet attaque abuse des fautes d'implémentations liés à l'emploi du protocole cryptographique pour re-installer une clé qui a déjà été utilisé. L'attaque peut ainsi réinitialiser quelques paramètres associés à la clé, comme les nonces de transmission et les compteurs des messages.

Tous les connections WiFi qui implémentent les protocoles **WPA/WPA2** utilisent le **4WH** (*Four-Way Handhsake*) pour générer une clé de session. Ce **4WH** a été prouvé sur en ce qui concerne la génération de cette clé, mais l'attaque démontre la présence d'une vulnérabilité sur son emplois. On aperçoit cela lors de la manipulation des messages du **4WH** et de leur rejeux.

5.1 Fonctionnement général de l'attaque

On peut résumer l'attaque en disant que quand le client va se connecter à un réseau, il va exécuter le **4WH** pour négocier une nouvelle clé de session et qu'il va installer celle-ci après avoir reçu le troisième message du **4WH**. À partir du moment ou la clé est installé, on va utiliser le protocole de chiffrement pour chiffrer tous les paquets. Toutefois, comme le message 3 peut être perdu (pour plusieurs raisons), le PA peut potentiellement retransmettre le message 3 si il n'a pas reçu une bonne réponse, comme le message 4 du type **ACK**. En résulte le fait que le client peut recevoir plusieurs fois le message 3. Chaque fois qu'il reçoit un de ces messages il va installer ou réinstaller la même clé, et ainsi réinitialiser le nonce et le compteur de messages. Il a été montré qu'un attaquant peut forcer la réutilisation des nonces s'il collect et rejoue la retransmission du message 3. En faisant ça, le protocole de chiffrement peut être attaqué.

L'attaque est spécifiquement dévastatrice pour les versions **2.4** et **2.5** de *wpa_ supplicant*, qui est un client WiFi très utilisé chez *Linux* et *Android*. Ici, le client va installer une *clé-zero* (suite de zéro) au lieu de réinstaller la vraie clé de session. On peut remarquer que la clé de session (**PTK**) ne fuite jamais au cours de cette attaque, donc l'attaquant ne peut pas forger des paquets **EAPOL**. Ça veut dire que en général il ne peut pas imiter le PA ou le client pendant et après le **4WH**.

5.2 Le Four Way Handshake plus détaillé

Rappelons que le **4WH** fournit l'authentification mutuelle entre le PA et le client, qui est basé sur un secret partagé appelé *Pairwise Master Key* (**PMK**), et qui fait la négociation d'une clé de session appelé *Pairwise Transient Key* (**PTK**). La **PMK** est dérivée du mot de passe de la connection WiFi, avec la fonction qu'on a étudié dans le chapitre 3.4. La **PTK** est dérivé de la **PMK**, du **ANonce**, **SNonce**, et des adresses **MAC** du client et du point d'accès.

Lorsqu'on a généré la clé **PTK**, elle va être divisé en trois parties, à savoir la *Key Confirmation Key* (**KCK**), *Key Encryption Key* (**KEK**), et la *Temporal Key* (**TK**). Comme on a pu le voir précédemment, la **KCK** et la **KEK** vont être utilisé pour protéger les messages du **4WH**, et la **TK** va protéger les paquets normaux (les paquets de donnés) grâce au protocole de chiffrement associé.

Tous les messages du **4WH** sont définis par la norme **EAPOL** et en-capsulés dans ce type de paquets. Au début de ces paquets on a l'entête, qui indique la nature du paquet, c'est à dire à quel message du **4WH** il appartient (message 1, message 2 ...). L'espace suivant, celui du *replay counter* est utilisé pour détecter si des paquets son rejoués. L'authenticator (le PA) augmente ce compteur tous les fois qu'il transmet un nouveau message. Quand le client répond à un paquet envoyé par le PA, il va utiliser le même *replay counter* pour l'émition du message réponse correspondant. L'espace du *nonce* transport les nonces utilisés par le PA et le client pour générer la clé. La clé du group (**GTK**) va être transporté dans l'espace *Key Data*, qui va être chiffré avec la **KEK**. L'authenticité du paquet va être protégé par la **KCK** et le *Message Integrity Check* (**MIC**).

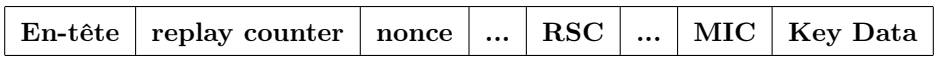

Schéma simplifié des paquets EAPoL

Le PA commence le **4WH** en envoyant le **message 1**. Il contient le **ANonce**, et il est l'unique message **EAPOL** qui n'est pas protégé pour le **MIC**. Au moment de recevoir ce paquet, le client génère son **SNonce** et il génère aussi la **PTK** car il possède tous les éléments pour la calculer. Puis le client envoie le **message 2** avec son **SNonce** vers le PA. Une fois que la borne connaît le **SNonce**, elle génère aussi la **PTK**, et elle envoie le **message 3** avec la **GTK** au client. Pour finaliser, le client répond un **ACK** via le **message 4**, juste après quoi, il installe la **PTK** et la **GTK**. Après avoir reçu le **message 4**, le PA installe aussi la **PTK** (la **GTK** est déjà installé). À noter que dans une connection déjà existant, la **PTK** peut être renouvelé en initialisant un nouveau **4WH**. Au moment de cet échange, tous les messages du **4WH** vont être chiffres avec la **PTK** courant.

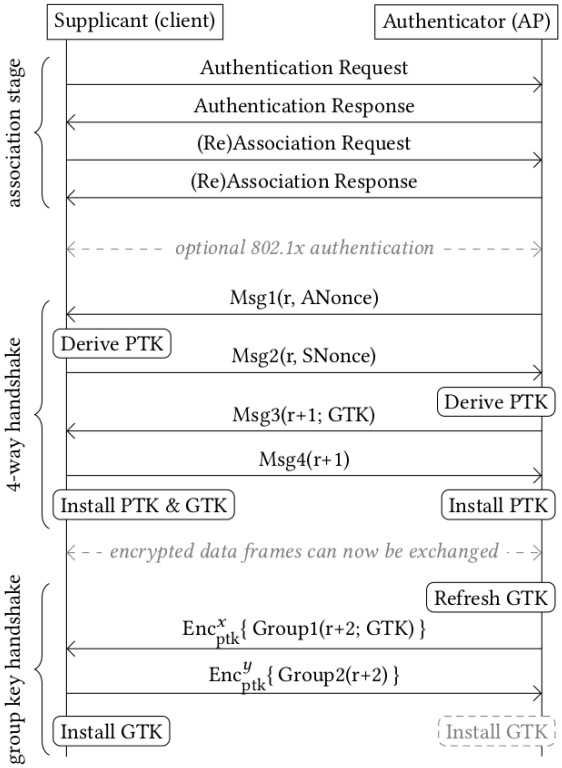

Figure 11: Échange de messages dans le 4-Way Handhsake [3]

La figure 11 illustre les messages qui sont échangés dans le **4WH**. On utilise la notation:

MsgN(r, Nonce; GTK)

Celle-ci représente le message **N** du **4WH**, avec le replay counter **r**, et avec le **nonce** (ANonce ou SNonce) selon le cas dans lequel on peut le trouver. Tous les autres paramètres sont stockés dans l'espace *Key Data*.

5.3 L'attaque de Réinstallation de clés

On peut voir que le client va accepter les retransmissions du **message 3**, donc il est possible de forcer la réinstallation de la **PTK**. Pour en arriver là, on doit établir une position de *man-in-the-middle* (**MiTM**) entre le client et le PA. On utilise cet **MiTM** pour déclencher la retransmission du **message 3**, en empêchant que le **message 4** arrive au PA. En effet cela implique que le PA va renvoyer le **message 3**, ce qui induit que le client va réinstaller la **PTK** qui est déjà utilisé, en réinitialisant le nonce utilisé dans le protocole de chiffrement et le compteur du message.

Par contre des complications peuvent avoir lieu au moment d'implémenter les attaques. Premièrement, pas tous les systèmes d'exploitation implémentent le protocole de la même manière. *Windows* et *iOS* n'acceptent pas la retransmission du **message 3**, et ça c'est une violation au

standard **802.11**. Toutefois, comme résultat, ces systèmes d'exploitation ne sont pas vulnérables à l'attaque de réinstallation des clés contre le **4WH**.

Un autre obstacle est que l'on doit avoir la position de **MiTM** entre le client et le PA. En effet ça ne marche pas si on se contente de mettre en place un borne frauduleuse avec une adresse MAC différent et en transmettant les paquets. On sait déjà que la clé de session **PTK** nécessite pour sa création l'adresse MAC du client et du PA, ce qui implique que si on utilise cette technique on va dériver une **PTK** différente, provoquant ainsi un échec lors du **4WH** et de l'attaque. Au lieu de cela, on doit donc implémenter une attaque **MiTM** basé sur l'emploi de différents canaux.

Pour commencer la machine attaquant doit disposer de deux interfaces réseau. Sur l'une d'elle il va se lier au point d'accès sur le canal standard. Puis, dans un autre, il va cloner la borne original avec la même adresse MAC. Après quoi, il va envoyer un message de dé-authentification afin que tout les machines se dis-associent de la borne. Et pour finir, il va récupérer sur sa deuxième interface réseau l'association du client en se faisant passer pour le point d'accès. Ainsi on obtient bien la position de MiTM tout en assurant que le client et le PA dérivent la même **PTK**.

Implémentation (SO)	**Re. M3**	**Clair EAPoL**	**4WH**
OS X 10.9.5	O	N	O
macOS Sierra 10.2	O	N	O
iOS 10.3.1	N	N/A	N
wpa_supplicant 2.3	O	O	O
wpa_supplicant 2.4-5	O	O	O
wpa_supplicant 2.6	O	O	O
Windows 7	N	N/A	N
Windows 10	N	N/A	N
OpenBSD 6.1	O	N	N

Table 2: Systèmes vulnérables

Le tableau 2 illustre le comportement de l'attaque selon les types de clients. La colonne **Re M3**: montre si le client accepte la retransmission du message 3. **Clair EAPOL**: montre si le client accepte des messages EAPOL lorsque la PTK a été déjà installé. **4WH**: montre si le 4WH est vulnérable à l'attaque.

5.4 Cas d'étude

Afin de mieux comprendre comment l'attaque KRACK opère, nous allons détailler le fonctionnement du rejeux des messages exploitant la faille de sécurité. Mais comme nous l'avons précisé avant, le 4WH possède quelque variation selon l'OS qui l'implémente. Donc, nous étudierons plusieurs cas pour voir comment, à partir d'un même principe, cette attaque peut être généralisée.

5.4.1 Retransmission du message 3 en clair

Si la victime accepte la retransmission du **message 3** en clair après l'installation de la clé **PTK**, l'attaque de réinstallation de clé est simple. Avant expliquer comment ça marche on doit expliquer la notation de la figure 12

$$Enc^{1}_{ptk}\{Data(...)\}$$

Où on veut dire que le message est un message de données, chiffré avec la clé **PTK** et 1 comme valeur du nonce.

Au début, l'adversaire utilise la position de **MiTM** basé dans le canal pour pouvoir manipuler les messages du **4WH**. Dans un premier temps il va laisser passer les trois premières messages. Puis il va bloquer le **message 4**, en empêchant qu'il arrive au PA. Immédiatement après avoir émit le **message 4**, le client va installer la **PTK** et la **GTK**. À partir de ce moment le client va ouvrir le port 802.1x pour commencer la transmission des paquets normaux (paquets de donnés).

On peut voir dans l'étape deux de la figure 12 que le premier paquet de données a une nonce de **1** dans le protocole de chiffrement. Dans l'étape trois le PA retransmet le **message 3** parce qu'il n'a pas reçu le **message 4**. L'adversaire transmet le **message 3** qui a été retransmis par le PA, ce qui induit que le client re-installe la **PTK** et la **GTK**. Comme résultat on obtient la ré-initialisation du nonce et du compteur utilisé dans le protocole de chiffrement.

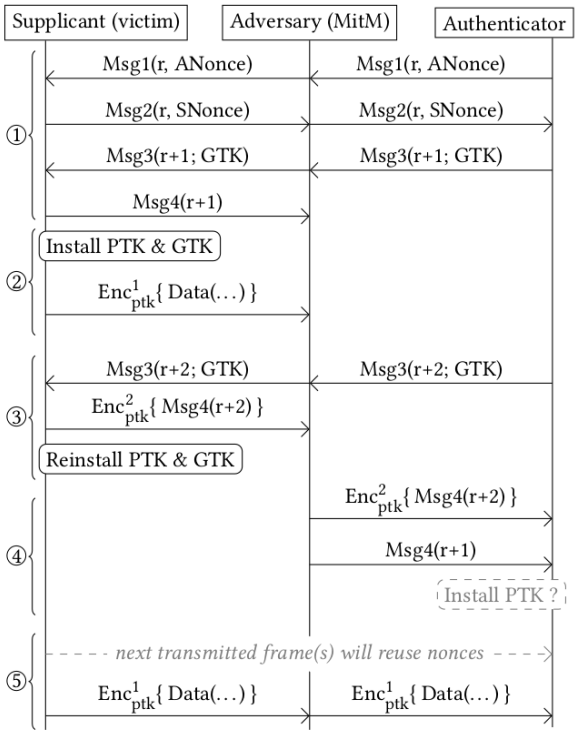

Figure 12: KRACK Attack sur le 4WH, quand le client accepte la retransmission du message 3 en clair. [3]

À noter que l'attaquant ne peut pas retransmettre un **message 3** antécédente, car le compteur **EAPOL** n'est plus bon. Finalement, quand le client transmet le prochain paquet de données, le protocole de chiffrement va réutiliser le nonce. Il est également important de savoir que l'adversaire peut attendre une quantité de temps arbitraire avant de retransmettre le **message 3**. Ainsi il peut contrôler la quantité de nonces que vont être réutilisés. De la même manière, l'attaquant peut toujours effectuer l'attaque une autre fois en faisant une de-authentification du client.

La quatrième étape illustrée dans la figure 12 a pour but de compléter le **4WH** du côté du point d'accès. Ça n'est pas trivial parce que le client a déjà installé la **PTK**, ce qui veut dire que le **message 4** arrive chiffré. Et comme le PA n'a pas encore installé la **PTK**, il peut normalement rejeter le **message 4**. Toutefois, dans le standard **802.11**, il est dit que le PA peut accepter n'importe quel compteur de message qui ait été utilisé dans le **4WH**, pas seulement le dernier.

Plusieurs points d'accès peuvent accepter les valeurs du compteur anciennes, et certaines acceptent les compteurs qui ont été utilisé dans un message vers le client mais qui n'ont pas encore été utilisé comme une réponse du client. Ces points d'accès vont accepter les vieux **message 4**, non chiffré, avec le compteur **r+1** (dans notre figure). Par conséquent, le PA va installer la **PTK** et il va commencer à envoyer des messages unicast chiffrés au client.

Cette variante de l'attaque fonctionne contre l'implémentation du client WiFi dans les systèmes *MediaTek* et avec quelques versions du *wpa_ supplicant*.

5.4.2 Retransmission du message 3 chiffré

Il existe des clients qui sont vulnérables à l'attaque sur le **4WH** mais qui n'acceptent pas la retransmission des messages 3 en clair après avoir installé la clé **PTK**. Donc, ils vont accepter le message 3 seulement s'il arrive chiffré. Pour faire ça il faut exploiter une condition de carrière entre l'entité qui exécute le **4WH** et celui qui implémente le protocole de chiffrement. Ce qui corresponde dans notre cas au CPU de l'appareil et à sa carte réseau (**NIC**).

Comme exemple, on choisit d'étudier l'implémentation de l'attaque sur un client *Android*. *Android* va accepter la retransmission du message 3 en clair seulement si les messages 3 retransmis sont envoyés immédiatement après le premier message 3.

Ici l'attaquant va laisser passer les deux premières messages du **4WH**, mais il va bloquer le premier message 3, et il attend que le PA rejoue un autre message 3. Après l'attaquant envoie les deux messages 3 de façon consécutive. Le **NIC**, (qui implémente le protocole de chiffrement) à ce moment là, n'a pas encore installé la PTK, donc il envoie les messages au CPU directement. Le CPU (qui implémente le **4WH**) réponde au premier message 3 et indique au **NIC** d'installer la **PTK**. Après le CPU prend le deuxième message 3, et même s'il trouve que le message est arrivé en clair il l'accepte. En effet *Android* et *Linux* tolèrent des messages **EAPOL** non chiffrés comme exception, ainsi le CPU va traiter le paquet. Comme le **NIC** vient d'installer la **PTK**, la réponse va être chiffré avec la valeur du nonce à **1**. Après ça, le CPU indique au **NIC** de re-installer la **PTK**, et comme ça, le **NIC** va réinitialiser le nonce et le *replay counter* associés à la **PTK**, ainsi le prochaine paquet de données va réutiliser le nonce **1**.

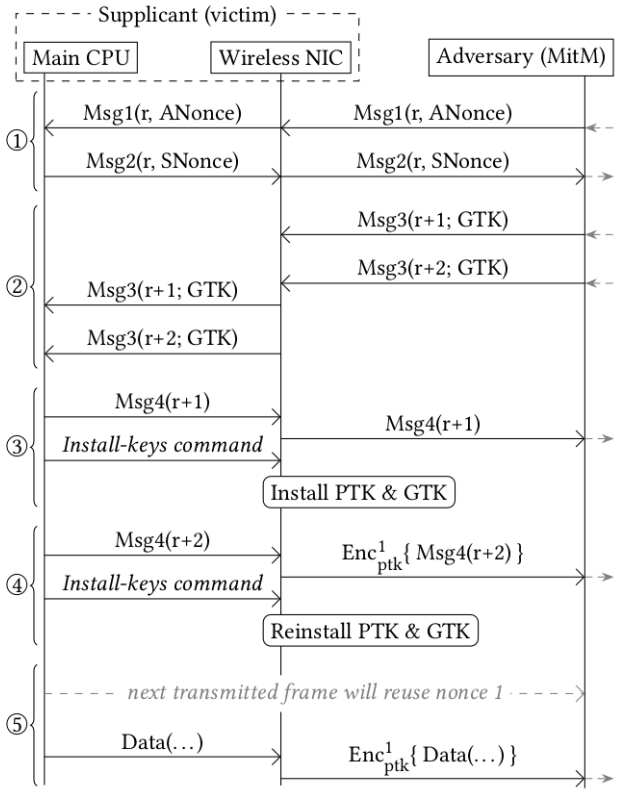

Figure 13: KRACK Attack sur le 4WH. À priori, le client n'accepte pas la retransmission du message 3 en clair. Cas Android/Linux [3]

5.4.3 Vulnérabilité Clé-Zero

L'attaque vers le **4WH** sur les clients qui ont le *wpa_ supplicant* (c'est le composant **IEEE 802.11i** qui utilisent les clients WPA2) a une comportement un peu special. Dans les versions **2.4** et **2.5** du *wpa_ supplicant*, le client va installer une *clé-zero* (une clé où tous les bits son à zero) comme *encryption key* (**TK**) quand il reçoit un message 3 retransmis. Cette vulnérabilité est du à une remarque dans le standard **802.11** qui dit qu'il faut enlever la **TK** de la mémoire après l'avoir installé.

Comme Android utilise intérieurement une version un peu modifié du *wpa_ supplicant* (la version 2.4 et 2.5 ont été plutôt implémenté sur Android 6.0), il est aussi affecté par cette attaque, donc un peu plus de 30% des appareils qui utilisent Android sont vulnérables à l'attaque.

Au moment de faire l'attaque et quand le client a installé une clé-zero pour faire le chiffrement, le déchiffrement des messages devient pratiquement trivial. En effet, comme la **TK** est mis à zéro la part d'inconnus qui génère la keystream disparaît car elle est crée a partir de cette valeur et d'un nonce que l'on peut connaître. C'est pour ça que cette faille a des conséquences catastrophiques.

5.5 Les autres variantes de l'attaque

L'attaque **KRACK** peut être implémenté sous plusieurs variantes, qui vont changer un peu le fonctionnement de l'attaque, mais qui va rester avec le même principe général. Cela varie en fonction du système d'exploitation que l'on veut attaquer. On a déjà vu en détaille l'attaque sur le **4WH**, et on peut l'implémenter dans le cas où la victime accepte la retransmission du message 3 en clair après l'installation de la clé **PTK**. Mais on a l'opportunité d'implémenter le même attaque sur le **4WH** lorsque le client n'accepte que la retransmission du message 3 chiffré. Nous pouvons aussi faire l'attaque sur le *Group Key Handshake* et le *Fast BSS Transition Handhsake*.

5.5.1 Group Key Handshake

Les réseaux mettent à jour leurs *Group Key* (**GK**) périodiquement, pour s'assurer que les clients qui ont été authentifiés puissent l'utiliser. Ce changement est effectué avec le *Group Key Handshake* (**GKH**), dont une preuve formelle assure la fiabilité. Comme on peut l'observer dans la figure 11, le **GKH** est initialisé par le PA quand il envoie un *message de groupe 1* à tous les clients. Le PA va retransmettre ce message s'il ne reçoit pas une réponse approprié du type **ACK**. À noter que le *replay counter* **EAPOL** de ces messages retransmis sont toujours incrémenté de 1.

Lors de l'attaque contre le **GKH**, le but est de capturer un *message de groupe 1* qui est retransmis. Puis on l'empêche d'arriver au client en évitant la retransmission. Pour finir, on peut renvoyer ultérieurement ce même message au client. Ce qui aura pour conséquence de tromper le client pour qu'il réinitialise le replay counter de la GK déjà installé.

Le premier pre-requis pour faire l'attaque est que le client doit réinitialiser le *replay counter* au moment de la réinstallation de la **GK**. Or ce fait est justement effectué par tous les type de clients WiFi (voir tableau 2). Par voix de conséquence, tous les clients WiFi sont vulnérables à cet attaque.

Le deuxième est que l'on doit être capable de trouver un message de groupe que le client est susceptible d'accepter. De plus, celui-ci doit contenir la **GK** qui est actuellement en train d'être utilisé par le PA. Le PA va commencer à utiliser la **GK** neuve immédiatement après avoir envoyer le premier *message de groupe 1.* Sinon il peut attendre que tous les clients aient répondus avec le message 2. Quand le PA installe la **GK**, l'attaque est direct.

Seulement le PA peut envoyer des vrais paquets broadcast et multicast (*group frames*) qui sont chiffrés avec la **GK**. Comme l'attaque se base sur les clients, ça veut dire l'on ne peut pas forcer la réutilisation des nonces pendant le chiffrement. Toutefois, le client réinitialise le *replay counter*

quand il réinstalle la **GK**, ce qui peut être utilisé pour rejouer des messages vers le client.

La plupart des PA vont mettre à jour la **GK** toutes les heures. Certain réseaux la met également à jour chaque fois qu'un client sort de ce même réseau. Après, les clients peuvent commencer un **GKH** en envoyant des messages **EAPOL** possédant des flags spéciaux. Comme certains PA ne vérifient pas l'authenticité de ces messages, l'attaquant peut générer et déclencher une mis à jours de la **GK**. La combinaison de toutes ces situations peut faire que presque tous les réseaux soient vulnérables au mis à jour de la clé, et être vulnérables à l'attaque comme conséquence.

5.5.2 Fast BSS Transition Handshake

Le but de cet handshake est de diminuer le temps de *roaming* au moment que le client change d'un PA à un autre dans le même réseau, en utilisant les clés maîtres générées par le 4WH. Additionnellement, il intègre le 4WH dans les paquets d'authentification et ré association.

À différence du 4WH, cet échange est initialisé par le client, les deux premières messages sont un requête et un réponse pour l'authentification, ils sont l'équivalent du message 1 et 2 du 4WH, et ils vont avoir des nonces qui vont dériver une nouveau clé de session. Après ça, le client envoie un requête de re-association et le PA va lui répondre. Ce sont des messages qui ressemblent aux troisième et quatrième messages du 4WH. On finit la communication et on envoie la GTK au client.

Seulement les messages de re-association sont authentifiés avec le MIC. Après, aucun message dans le FT handshake contient des *replay counters*. Au lieu, le FT handshake repose sur le SNonce et le ANonce pour fournir de protection anti rejeu de messages.

Selon le standard, la PTK doit être installé après le message de réponse pour l'authentification a été envoyé ou reçu, et le port 802.1x va être ouvert après avoir envoyé ou reçu la requête de re-association. Ça va assurer que, même si le PTK est déjà installé pendant le handshake, le PA et le client vont transmettre et accepter des paquets de données après que le FT handshake ait fini. Normalement ça doit protéger les dispositifs de l'attaque KRACK, mais la plupart des implémentations font l'installation de la PTK et la GTK après envoyer ou recevoir la réponse de la re association. Comme résultat, dans la pratique, la plupart des implémentations du FT handshake seront vulnérables à l'attaque KRACK de re installation de clés.

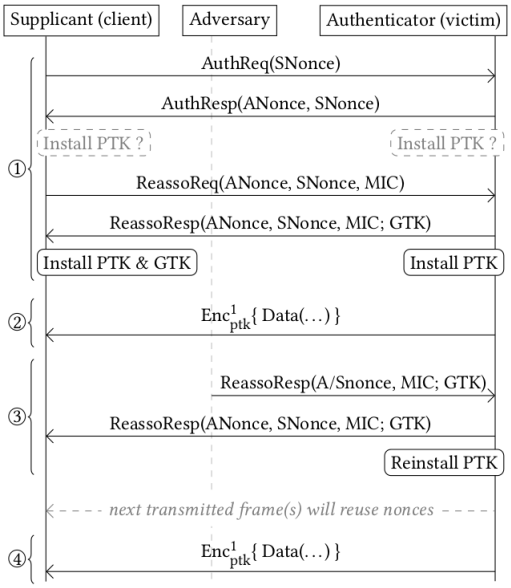

Figure 14: Fast BSS Transition Handhsake. À noter que ici nous n'avons pas besoin d'avoir une position MiTM. [3]

5.6 Impact de la réutilisation des nonces

L'impact de la réutilisation des nonces va dépendre du protocole de chiffrement qu'on utilise. Qui peut être **TKIP, CCMP** ou **GCMP**, tous les trois utilisent le chiffrement par flux pour chiffrer les paquets. Donc, la réutilisation d'un nonce en particulier implique toujours la réutilisation d'un même *keystream*, c'est ce qui nous permet de déchiffrer les paquets. Après, l'attaque re initialise le *replay counter* du client, donc tous les protocoles sont vulnérables aux attaques de rejeu.

Plus spécifiquement, quand on utilise le protocole **CCMP**, qui est le plus utilisé comme nous l'avons expliqué en détaille dans la section 3.6.3, l'attaque ne peut que rejouer et déchiffrer les paquets. On peut revenir au fonctionnement du protocole et du **4WH** pour mieux comprendre comment on peut faire pour déchiffrer les paquets. Il faut se souvenir que :

- Le numéro du paquet (nonce) est incrémenté +1 pour chaque paquet envoyé.
- Le nonce ne doit être jamais répété.
- Si le nonce est répété on aura le même *keystream* et on aura l'opportunité de l'utiliser pour déchiffrer les paquets.

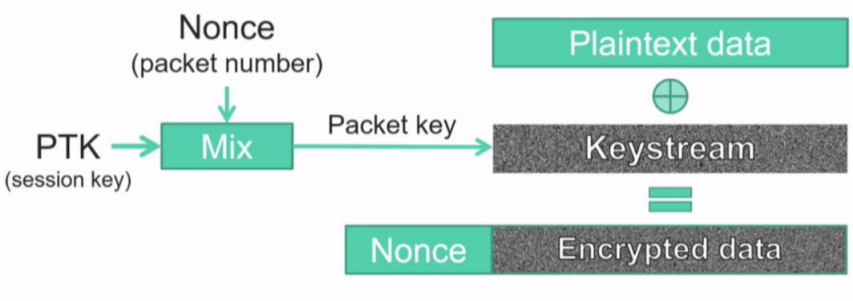

Figure 15: Fonctionnement du chiffrement CCMP dans WPA2 simplifié. [8]

Comme on le sait déjà, l'attaque va réinstaller la **PTK** et cela implique que l'on va ré-initialiser le nonce, et ainsi le réutiliser dans le protocole de chiffrement, comme le montre la figure 16.

- Si le client envoie un paquet après installer la **PTK**, il va avoir la valeur de **1** comme nonce, le prochaine aura la valeur de **2**, etc...
- Si on renvoie le **message 3** au client après ces premières paquets, le client va réinstaller la **PTK** et ré initialiser le nonce.
- Si le client envoie des autres paquets après ça, le première va avoir une autre fois la valeur de **1** comme nonce, le prochaine une valeur de **2**, etc...

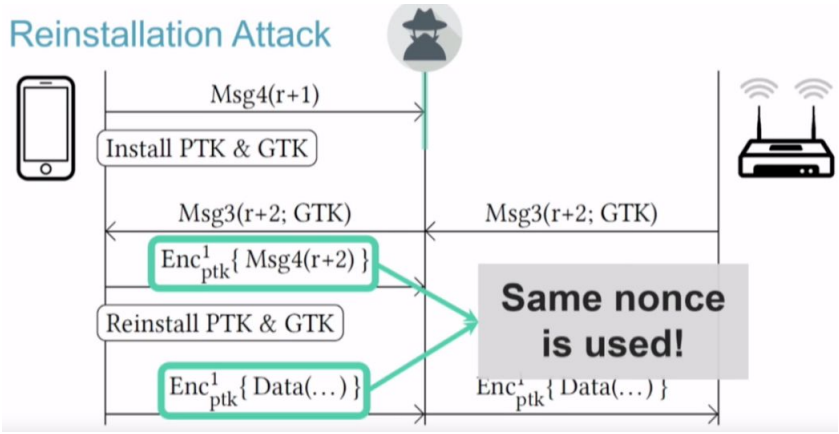

Figure 16: Réutilisation des nonces. [8]

Donc on a vu la réutilisation des nonces, mais pour pouvoir déchiffrer les messages il faut ajouter que :

- Le premier **message 4** du **4WH** a été envoyé en clair. Après avoir réinstallé la **PTK** on va envoyer un autre **message 4**, mais cette fois il va être chiffré avec le nonce **1**. Du coup on a maintenant deux **message 4** l'un en clair et l'autre chiffré.

– Comme les algorithmes de chiffrement, dans ce cas **AES-CCMP**, sont publics, ont sait comment ils fonctionnent. Donc on peut faire un **XOR** entre ces deux **messages 4** et on obtient la *keystream* qui correspond au nonce avec la valeur de **1**.

– Après on renvoie le **message 3**, en ré-initialisant les nonces. On utilise la *keystream* déjà calculé et on faite une autre opération **XOR** avec le paquet envoyé par le client avec le nonce du valeur **1**.

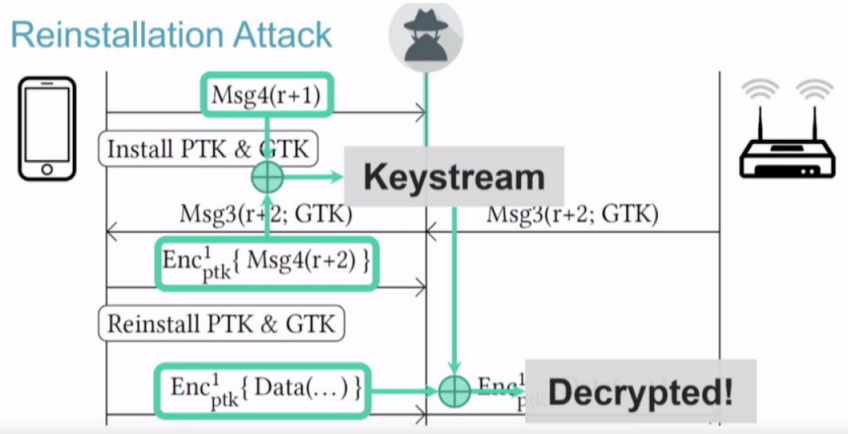

Figure 17: Déchiffrement des paquets avec le même nonce. [8]

Il faut se souvenir que l'attaquant a le droit d'attendre le temps qu'il veut pour renvoyer les **messages 3**, ça veut dire qu'il n'est pas forcément obligé d'utiliser les nonces de valeur **1**, il peut sélectionner les paquets qu'il veut déchiffrer.

En général, l'adversaire peut toujours rejouer, déchiffrer et créer des paquets dans une direction spécifique de la communication. La direction dans laquelle on peut faire ça va dépendre du handshake qu'on veut attaquer. Par exemple, comme l'implémentation sur le **4WH** attaque le client, on peut rejouer des messages unicast et broadcast vers le client; déchiffrer des messages envoyés par le client vers le PA; et créer des paquets depuis le client vers le PA.

	Rejouer	Déchiffrer	Forger
4 Way Handshake			
TKIP	AP -> client	client -> AP	client -> AP
CCMP	AP -> client	client -> AP	
Group Handshake			
TOUS	AP -> client		

Figure 18: Impact de la re-utilisation des nonces et la re-installation de clés.

5.7 Contre-mesures pour l'attaque KRACK

Afin de clôturer une étude sur une faille de sécurité, il est de mise de toujours spécifier les solutions qui ont été proposées. C'est pourquoi nous allons expliciter plus en détail les deux contre-mesures qui ont été suggérés dans un article de référence [3].

La première méthode consiste à ce que l'agent qui assure la confidentialité des données vérifie si une clé est déjà utilisé ou en cours d'installation (par comparaison avec la clé courante). Si c'est le cas, alors on ne doit pas ré-initialiser la valeur du nonce et du conteur de rejeu. Cela prévient nos attaques, du moins si un adversaire ne peut pas temporairement tromper l'implémentation en installant une autre clé (ancienne) avant de réinstaller celle en cours. En particulier, lorsque vous utilisez cette contre-mesure, il est essentiel que seulement le *replay counter* des messages du *Group Key Handshake* reçus augment. Sinon, un adversaire peut utiliser un ancien message de groupe 1 pour que la victime installe temporairement une ancienne clé (différente), pour ensuite réinstaller la clé de groupe actuelle en utilisant un message de groupe 1 plus récent.

La deuxième solution est également simple, lors de la prise de contacte, l'entité implémentant le protocole de confidentialité des données, doit s'assurer qu'une clé particulière n'est installé qu'une seule fois dans la même exécution de la prise de contacte. Comme ça peut être le cas de la PTK dans le four-way handshake. Ce qui veut dire que lorsque le client reçoit la retransmission d'un message 3, il doit répondre, mais ne doit pas réinstaller la clé de session. Pour cela, la version 2.6 et plus de wpa_ supplicant ajoute une variable bouléene à la machine d'état. Elle est initialisé à faux, et mis à vrai lors de la génération d'une nouvelle PTK. Or à chaque fois que l'on demande une nouvelle installation de PTK, on regarde la valeur de ce booléen, ce qui déterminera ou non son acceptation.

6 Vérification expérimental de l'attaque KRACK

Dans un premier temps, pour faire l'implémentation de l'attaque ou une preuve du concept, nous avons contacté à **Mathy Vanhoef PhD.**. En effet il a développé une PoC (proof of concept) en langage python en utilisant des librairies et fonctions *SCAPY* qu'il a partagé librement sur son dépôt *GitHub*. Normalement pour exécuter le programme il faut avoir deux cartes réseaux, pour le clonage des canaux, sur la machine qui effectue l'attaque.

Nous allons maintenant expliquer le fonctionnement du programme, développé par **M. Vanhoef**, à fin de montrer comment l'attaque de re-installation de clés KRACK est réalisé. À noter que cette programme a été rendu fonctionnelle que contre les clients Linux ou Android.

Pour implémenter l'attaque de l'homme du milieu, le programme prend les noms des deux cartes réseaux de la machine attaquant, l'addresse MAC du client et le SSID du réseau à attaquer comme arguments. Une carte réseau sera connecté au vrai point d'accès et l'autre au client. Pour faire ça, il est nécessaire que les deux interfaces soient en mode monitor. Après avoir obtenu la position MiTM, on va laisser passer les paquets entre le PA et le client, mais on va regarder son contenu. Si les paquets sont de type EAPOL, ça veut dire que très probablement le four way handshake est en train d'être exécuté, donc on les analysera de manière plus approfondit.

Si c'est le premier message du 4WH, on enregistre le message. Si c'est le troisième message on le sauvegarde dans une petite liste. Si dans celle-ci nous avons au moins 2 messages 3, on les envoie consécutivement au client. On remarque ici que le code a été développé pour simuler l'attaque lorsque le client n'accepte que les message 3 chiffrés, comme on l'explique dans la partie 5.4.2 du rapport.

Après ce traitement des message EAPoL, on regarde les messages envoyés par le client, qui en théorie doivent déjà être chiffrés, car la clé a été déjà installé. On extrait la partie chiffré du paquet et on calcule une keystream. On extrait aussi le IV du paquet et on regarde si on l'a déjà reçu avant (on enregistre les IV de tous les paquets dans un petit dictionnaire). Dans le cas où le IV a déjà été utilisé, et que la keystream a déjà été reçu avant, on peut dire qu'on a réussi à détecter la réutilisation de nonces et la réinstallation d'une clé qui a déjà été utilisé. Si la keystream n'est pas la même, on peut dire qu'on a détecté seulement la réutilisation de nonces, et que le client a installé une nouvelle clé, très probablement la Clé-Zero.

Suit à cette analyse du fonctionnement du code disponible, nous avons cherché à l'exécuter par nous même afin de confirmer la PoC. Hélas, cela n'a pas put aboutir suit a un problème de compatibilité matériel. Nous avions à notre disposition du matériel en vus de créer la configuration

nécessaire à l'expérience. Un point d'accès *TP-Link* nous avait été prêté par l'université, ce qui nous a permis de configurer une réseau locale dans lequel tous les tests aurait peut être confinés afin de respecter les normes.

Nous disposions également d'un ordinateur personnel qui devait effectuer le rôle d'attaquant et de cible. Dans un première temps , après s'assurer que deux machine client puissent se *pinger*, nous avons été en mesure de confirmer la présence du 4WH et l'échange des quatre messages correspondant. Tout ceci étant effectué grâce au capture réseau de WireShark depuis l'ordinateur attaquant, toutefois il est bon de signaler que pour cela une seul carte réseau suffit. Nous avions donc l'intention d'exécuter sur la machine compromettante (sous Kali Linux) le code crée par M. Vanhoef. Ainsi on aurait du obtenir la position de MiTM, puis l'attaque aurais put être effectué concrètement. Suit à quoi nous aurions pu déchiffrer le trafic en direction de l'ordinateur cible, ce qui nous aurait permis de voir le ping entrant. De ce fait, la technique KRACK attaque se basant sur la ré-initialisation du nonce aurait peut être confirmé.

Venons en au point délicat qui nous a limité lors de la mis en place du test. Rappelons que dans la mesure où la machine attaquant doit être en position de MiTM, il faut quelle ait au moins deux cartes réseau distinctes. Évidemment la première étant celle déjà présente dans l'ordinateur, il n'en manque qu'une. C'est pourquoi nous avons tenté un première essai avec un adaptateur WIFI de la marque *CSL*, mais du fait qu'aucun driver soit créé spécifiquement pour Linux, elle n'a pas pu être détecté par le système d'exploitation. Donc, nous nous en somme procuré une autre, le modèle *TP-Link TL WN722N*, à partir des listes d'adaptateur compatible à Kali Linux.

Toutefois, la version y joue un rôle majeur, car la première est reconnus pour sa compatibilité et sa simplicité alors que le second est réputé incompatible. Or les fournisseurs de matérielle informatique ne donnant que selon leur approvisionnement, nous avons reçu la nouvelle *version 3* qui est à ce jour très peu documenté. Pourtant, à partir d'un code source d'adaptation du driver pour la deuxième version, nous avons réussi à faire en sorte que le système d'exploitation reconnaît l'adaptateur WiFi. Toutefois, l'attaque nécessite obligatoirement le passage en mode moniteur de ces deux interfaces. C'est là que le bas blesse, car malgré différentes méthodes et tentatives, l'adaptateur n'a pu être mis en mode moniteur. Tout t'entative d'exécuter le code de l'attaque a donc échoué. Nous nous somme donc retrouvé dans une impasse, et n'avons pas pu réaliser l'expérience par nous même, bien que la démarche soit présente.

7 Conclusions

Comme le suggérait le fait que la recherche a été fait au préalable, par l'équipe qui a présenté cette faille au *Computer and Communications Security*, notre travail a principalement consisté à effectuer une étude. Toutefois bien que cela semble plus restreint, du fait de l'amplitude du champs de connaissance abordé, la tache est resté complexe. En effet, l'une des principales difficultés que nous avons rencontré, a été la gestion des informations. L'étude se portant sur un point technique dans le fonctionnement des transmissions sans fils, il a été à la fois nécessaire d'avoir un connaissance global de ce dernier et une vu plus fine du protocole qui nous intéresse. Il nous a donc fallut effectuer un trie en permanence entre le superflue, l'utile et l'ésentielle dans un documentation fournie. La synthèse de toute ces informations a donc été la clé de notre travail.

De multiples connaissances ont été acquises au cours de ce TER. Après un bref rappel historique des évolutions technologiques dans ce milieux, nous avons consacré une grande partie à la description général du fonctionnement des protocoles de sécurité des réseaux sans fils. Au sein de cette vision global, suit à un petit récapitulatif des attaques déjà fortement documentée, nous avons consacré la deuxième grand partie au détail du fonctionnement de la méthode d'exploitation KRACK. Qui plus est, nous avons également tenté de reproduire la preuve de concept, et à défaut de l'expliquer. Du fait de sa découvert récente, cette étude permet d'avoir une vulgarisation du principe et des manières d'utiliser de cette attaque. Qui plus est, couplé avec la première partie, nous avons réunis au sein d'un seul et même document toutes les connaissances nécessaires, pour qu'une personne qui n'ait que quelques bases en réseau, puisse comprendre cette faille, son contexte et plus généralement le fonctionnement de la sécurité dans les réseaux sans fils.

En prenant du recule, il est intéressant de voir comment un certain degré de liberté dans la norme d'un protocole peut aboutir à une faille majeur de sécurité. Ce qui tant à montrer que malgré la présence de preuve sur la protection cryptographique des données, des failles peuvent tout de même être exploité en périphérie. C'est donc dans sa totalité qu'il faut concevoir et vérifier les protocoles que l'on veut sécuritaire. Mais malgré tous les efforts investit, comme c'est le cas dans notre étude, certaines failles peuvent être découverte que bien plus tard. Il est donc primordiale de continuer à effectuer des recherches, et à divulguer leur information afin que les méthodes d'attaques soient comprises en vue dis prévenir.

8 Bibliographie

References

[1] Aurélien Géron. *WIFI Professionnel: La norme 802.11, le déploiement, la sécurité*. DUNOD. 2009.

[2] Matthew S Gast. *802.11 Wireless Networks: The Definitive Guide*. 2nd Edition. O'Relly. 2005.

[3] Mathy Vanhoef and Frank Piessens. *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2*. [*Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*]. 2017. ACM.

[4] Samia Alblwi and Khalil Shujaee. *A Survey on Wireless Security Protocol WPA2*. Int'l Conf. Security and Management, SAM'17.

[5] Paul Arana. *Benefits and Vulnerabilities of Wifi Protected Access 2 (WPA2)*. INFS 612. 2006.

[6] Devin Akin. *802.11i Authentication and Key Management (AKM)*. CWNP. 2005.

[7] Web site - Key Reinstallation Attacks Breaking WPA2 by forcing nonce reuse, `https://www.krackattacks.com/`

[8] Video - Key Reinstallation Attacks: Breaking the WPA2 Protocol, `https://www.youtube.com/watch?v=fZ1R9RliM1w`

[9] Video - KRACK Attacks: Bypassing WPA2 against Android and Linux, `https://www.youtube.com/watch?v=Oh4WURZoR98`