WI-FI AND SECURITY

Kevin Hagner
jsaipakoimetr@spyzone.fr
June 5, 2013

When Internet started to become popular, and that our knowledge in electronic was good enough for allowing us to build light and easy to move computers, we started to need more connectivity. The need was even increased when real laptops started to be sold to mainstream users. Very quickly, it was uncomfortable to search one Ethernet port and be always connected to Internet by a wire.

The *Wi-Fi* was born in 1997 with *IEEE 802.11* that allowed 2 MBits transfers speed. It was definitely the beginning of a new era. People will quickly become more an more connected. Now, 15 years later, we are connected all the time, and even often with more that just one connection because we also have our smart phones on Internet now...

But what about security? It's easy for a system administrator to check who is connected to a wired network, but what about wireless ones? And for the user, how to be sure that we are really connected on the network that we think, and that nobody is spying us? It will be at this questions that we will try to answer.

CONTENTS

BEFORE STARTING

ABOUT

Disclaimer Please keep in mind that this document was written by a student without any kind of expertize in this domain. You shouldn't blindly trust his content. Even if I tried to do my best, it won't surprise me if I misunderstood something.

For this purpose, don't hesitate to contact me if you notice a mistake, or something that can be enhanced. For the same reason, check sometime if a new version of this document is available here: `http://share.spyzone.fr/projects/wifi_security.pdf`.

Thanks

Philippe Bordron for giving to me advices about how to write a good paper.

COMMON TERMS

Access point Two kinds of *Wi-Fi* network exists. The first one is called *ad-hoc* and represents a network without center: all users communicate directly with each other. We won't talk about this kind of network because it's not the most representative with securities issues, but encryptions systems that we will deals with works in the same way in this kind of network.

The second is *structured network* and are the most used (mainly because it's the only one that allow quite big network, and so, also an Internet connection). However, in this mode, we have to notion of client, and *access point* (abbreviated *AP*). One client always has to connect it-self to one *AP* that will do the bridge between him and the rest of the network. Two clients can't share data directly between them in this mode, but has, at least to pass by one *AP*.

***MAC* addresses** The *MAC* address is the physical and unique "name" of each network interfaces, in the second layer of the *OSI* stack.[1] It's the address used at the *Wi-Fi* level for setting the receiver and source of each packets.

Initialization vector It's a block of data usually randomly chosen for adding entropy in the keys used for the encryption or the authentication process. This initialization vector (abbreviated *IV*) is usually set by the emitter of the packet and is send with this one to the receiver in plain text.

His important characteristic is to be always different, and if possible, never used before during the communication.

Integrity Check Value One transmission of a packet can be altered in two ways:

The legitimate one When an interference just occurs and so, create some noise during the transmission of the packet. It can result some reception errors. It's very common in wireless communication.

The less legitimate one One cracker can also "stole" a packet, altered his content, and send it back to the legitimate receiver. It's much more vicious because we can't just use *checksums* [2] controls for fighting against forgery attacks. We have to use other integrity check that like encryption, will use a secret key necessary for the good computation of this value.

The integrity check can so be a method that fight just against natural alteration of data, or also against forgery attacks. It's depends of the algorithm. Here *WEP* is only useful for the first issue, and *WPA-TKIP* and *WPA-CCMP* are protected against both.

MSDU* and *MPDU When an operating system wants to send data trough the network, it sends this data to the *network* layer that will perform the fragmentation operation for splitting the big packets given by the upper layer in smaller ones that will actually be sent on the air.

The big packet is called *MSDU*, and the smalls ones *MPDU*. This segmentation mainly occurs for allowing us to resend only the *MPDU* if one transmission problem occurs. We have to be careful because some check or encryption are done to the *MPDU*, and some others ones to the entire *MSDU*.

HOW *Wi-Fi* WORKS

Like talkie-walkie, radio, or phone cells, Wi-Fi use radio waves for sending informations between two antennas, through the space. So one antenna is used by your computer or your smart phone, and the other one are a bridge that make the link between the wired, and the wireless part of the network.

A network is never only wireless. Usually, you have a wired network with some antennas (access points) that will emit a Wi-Fi signal at some places. We can identified those wireless outputs as kind of "air wires". Because they are invisible and that everybody could plug this air wires to each computers they want, it was necessary to create some kind of logical wires that we can protect again undesired users.

We can mainly get tree kinds of problems that occurs with wireless network that don't on physical ones.

- Interferences and lost of quality. So we need to check that the data are received correctly.

[1] Read about the *OSI* stack for more details: `https://en.wikipedia.org/wiki/Osi_stack`

- Security. Because it's much easy to connect to a wireless network and to hide our real identity, we need an authentication and confidentiality method.
- Automatically use the best access point if one network has more that one (like at university).

We will only deal in this report about the security issue. Thus, it will also introduce some knowledge that fight against interferences because both of them affect authenticity of the data, but other tools exists for managing quality that we won't introduce here.

How security works

Now, it could be interesting to introduce the fundamentals that have to be respected in order to claim that we are using a secure communication protocol.

Authentication The first step is to be sure that we are really communicating with the user/server that we think. No one should be able to use a fake identity, and each users should be distinctly identified.

Authenticity Each message sent have to come to the receiver with the exactly same content. We should be in measure to check if the content was altered or not.

Confidientiality A message should only be readable for the receiver. A cracker that intercept the messages shouldn't be able to understand it.

If this tree rules are respected with strong systems, we can pretend to have a quite relevant level of security for our communication process. This rules should be applicate also in the *Wi-Fi* environment.

Wired equivalent privacy

At the beginning, the *wired equivalent privacy (WEP)* algorithm was created directly with *802.11*, but weaknesses quickly appears [7].

Technologies used

The encryption key In *WEP* the encryption key is composed of two things:

The private key that is 5 or 13 bytes long, and that will be shared between all users of the access point.

The initialization vector that is a 3 bytes pseudo random number, that will changes for each packets.

The secret key is so build with the *IV* (initialization vector) concatenated with the private key. For example, if the key is *games*, and the *IV* for the first packet is *ske*, the encryption key that will be used for this packet is *skegames*. For the second one, the private key is still *games*, but we generate a other *IV*: *pks*. So now, the encryption key will be *pksgames*.

XOR The exclusive or boolean function allow us to know if other two booleans, A and B are similar, or not. If $A \neq B$ the result will be 1, otherwise, the result will be 0.

The function *XOR* is defined such as:

$$XOR(A,B) = \begin{cases} 0 & \text{if } A = B \\ 1 & \text{else} \end{cases}$$

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: *XOR* comparison table

Keep in mind that *XOR* is a weak cipher [6]. If you know two numbers, you will always be able to find the last one. This is an important part of how and why it's so fast to crack *WEP*.

RC4 is the encryption algorithm that is used with *WEP*. The power of it is that it's something very simple thus very fast for encrypting data. *RC4* is actually split in two parts: the key scheduling *(KSA)*, and the pseudo random generation *(PRGA)*.

KSA The key scheduling algorithm has the aim to generate a pseudo random array. We will explain how it's working:

- Let's say that $N = 256$. It represents the length of the pseudo random array (the array will have 256 elements). It's the case with *WEP*.
- We also have *S*, the array that has the length of *N*. Its default content equals to the position of the element in the array. It means that *S* will looks like: *S[0]=0*, *S[1]=1*, *S[2]=2* $\cdots$ *S[255]=255*.
- Assume also that *K* is an array that contains the encryption key, so the *IV* + the private key.

Now, the goal is to use the encryption key *K* for scrambled the ordered suite contains in *S*. *KSA* do it like algorithm 1:

Algorithm 1: *KSA*

Data: The encryption key K
Data: The linear S array
Result: The swaped S array
$j \leftarrow 0$
for $i \leftarrow 0$ **to** $N - 1$ **do**
  $j \leftarrow (j + S[i] + K[i])\ mod\ N$
  swap(S[i], S[j])
end

The loop assure us that all elements in *S* will at least be swaped one time. The two variables i and j are indexes for the S array. We can also say that j is an

monotonic increasing function[2] that will have a value between [0, $N-1$], and that i is a linear one that will pass trough all S values. Finally, we swap the current $S[i]$ with the pseudo-random selected $S[j]$.

The swap process just invert the values in both element. If you have `swap(A, B)`, you will have a permutation of the values stored in A and B.

As you can see, it's not a terribly complex process... The only element that assure the randomization is a swap process between two elements in function of the value of the encryption key.

PRGA The pseudo random generation algorithm is almost the same that the previous one. Let's assume that S is still our array with N value that was previously created and scrambled (so the output of *KSA*). We will just do a loop like the one bellow for all bytes in the packet (named *data* in the algorithm) that we want to encrypt:

Algorithm 2: *PRGA*

Data: Plain text data
Data: Pseudo random array S
Result: Encrypted data
$i \leftarrow 0$
$j \leftarrow 0$
while *generation loop* **do**
  $i \leftarrow (i+1) \bmod N$
  $j \leftarrow (j + S[i]) \bmod N$
  swap(S[i], S[j])
  $z \leftarrow S[(S[i] + S[j]) \bmod N]$
  return XOR(z, plain text data)
end

At the two last lines, we select the *S* value with which we will do an *XOR* on the plain text data. After this process, the output will be the encrypted byte that will be send through the air.

CRC Because data will fly in the air, the probability that interferences appears during the transfer are high. The cyclic redundancy checksum (*CRC*) is a checking value that is included in the package with data for being sure that we don't have a corruption.

Concretely, the *CRC* is computed by the emitter, and added at the end of the package, before the encryption and the sending. When the receiver get the package, he will decrypt it, and after separating the real data and the *CRC* computed by the sender. He will then compute his own *CRC* with the data, and compare it with the one sent by the emitter. If both of them has the same value, the packet is considered as clean. Otherwise, it is considered at corrupted and is dropped [2].

AUTHENTICATION

The authentication with *WEP* works in a wired way. You actually have the choice between two modes that are both of them based on the same pre-shared key system.

Open System Authentication is the default protocol, and just allow each connection request to have access to the access point. The authentication handshake is done in plain text, but *WEP* has to be used for transferring data. It means than everybody can have a working authentication, but only people that really know the *WEP* key can use this connection.

Shared key authentication is a system that will send a challenge to the user that wants to be authenticated on the access point. This challenge consists of encrypting a random message sent by the access point. If the client encrypts it correctly, it means that he really know the *WEP* key, so the client is authenticated. Otherwise, the authentication failed. With this system, the user needs to know the key both for authentication and transmission purpose.

We can imagine that the shared key authentication system is more secure that the open one, but actually, no. Both of them need the *WEP* key for being usable, even if with the open system authentication each clients can be connected the access point. But shared key authentication has a strong weakness: the plaintext and encrypted challenge are sent in the air, so one cracker that sniff them can just bruteforcing the challenge for finding the key. It's really not a big deal with a key that is five characters long...

ENCRYPTION PROCESS

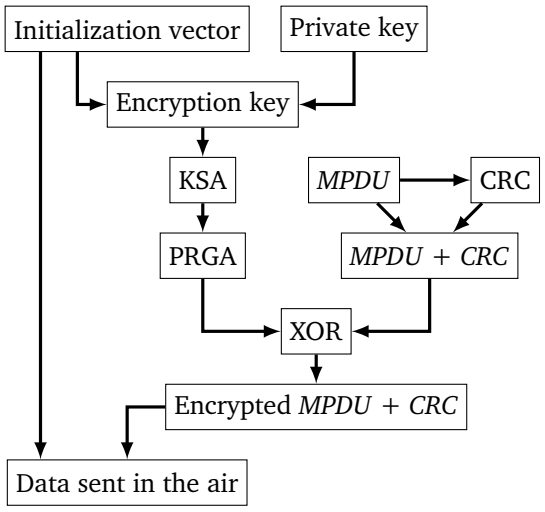

Figure 1: *WEP* encryption diagram

We will now use an concrete example. Imagine that we have the letter *h* to encrypt. The *ASCII* representation of *h* is 68_{16} in hexadecimal. We did the *KSA* step, and the PRGA one for finding the value of *S* that we will use for the *XOR*. In the example, we will say that it is *S[43]* = 84_{16}. We can so deduce the encrypted value by *XOR*:

[2]Monotonic increasing function: `https://en.wikipedia.org/wiki/Monotonic_function`

bit	h	S[43]	encrypted h
1	0	0	0
2	0	0	0
3	0	1	1
4	1	0	1
5	0	0	0
6	1	0	1
7	1	0	1
8	0	1	1

Table 2: Encryption of the *h* letter

The diagram showed in Figure 1 summarize how *WEP* encrypts data. The things to understand is that the encryption key is not the same for each packets. Some bytes changes (the initialization vector) for making in fact that a cracker can't decrypt all data if he find how to decrypt a single packet.

So finally data sent will be the plain *IV* (that the receiver will need for *KSA* and *PRGA* plus encrypted data that represents real data (the *MPDU* and the checksum (*CRC*) for checking if the packet is corrupted or not.

As you can see, the *CRC* checksum only covered the *MPDU* content. This mean that if the *IV* is altered, we can not trust on it for noticing the modification. It's not a big deal because with a wrong *IV*, the decryption will produce an invalid packet that will be drop afterward.

Decryption process

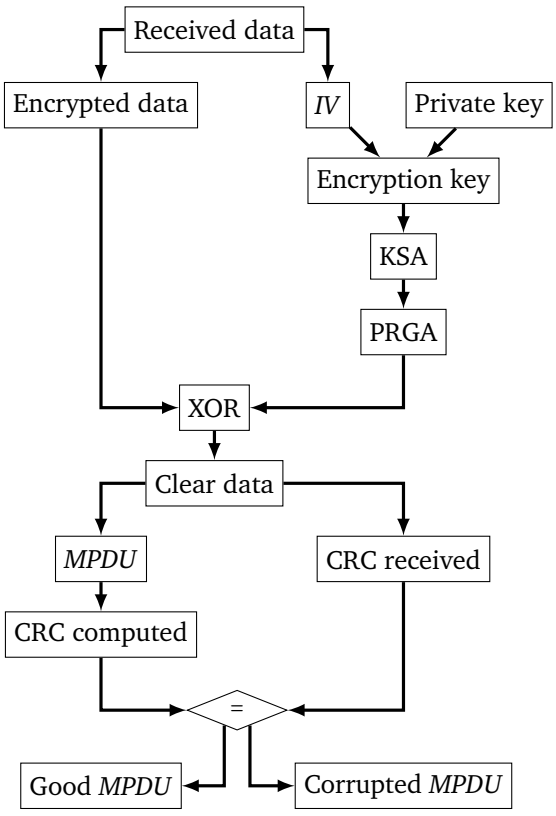

Figure 2: *WEP* decryption diagram

Now that we know how to encrypt data, it's time to learn how to decrypt them. It's almost the same, but in the inversed order: the receiver will have to generate the key with *KSA* and *PRGA* for *XORing* the encrypted data, and so find the plain text value, plus the *CRC*.

The new step that will have to do now the receiver is to compute his own checksum of the plain text data, and compare the result with the *CRC* received by the emitter. If both of them are equal, the *MPDU* will be considered as valid and will be sent to the network layer. Otherwise, the package is considered as corrupted and is droped.

Breaking the encryption

Because *IVs* are small and send in plain text, the result is still weak. Especially because we just use the *XOR* solution, and that we also use a weak checksum technology[3]. We will here introduce one of the first cracking method that was created for *WEP* keys that use *RC4* weaknesses.

Now, severals methods exists that are much more powerful, but also more complex. We won't learn those ones here, but you are encouraged to learn how they works when you will understand this one [12, page 3].

Finding the 3 first *S* values It's mean that we have to find *S[0]*, *S[1]*, *S[2]* after the *KSA* proccess. So the values that will be used by *PRGA* for *XORing* data. This step is very easy to do because the *IV* represents the 3 first byte of the encryption key, and they are send in plain text at the beginning of the packet.

For this example, we will take an initialization vector that looks like that: *IV = (3, 255, 7)*. After the four first iteration of the *KSA* loop, we get that[4] $S[0] = 3, S[1] = 0, S[2] = 12, S[3] = 1, S[12] = 2$

The new thing to know is that some *IVs* are more weak than other ones, one pattern of weak ones is: $IV = B + 3, N - 1, x$ with *B* as the secret key bit that we want to crack, and *N* the length of the *S* table, so 256 for *WEP*. With this weak packet, we have 5% of chance that *S[0]* to *S[2]* will remains the same after the 3 first iterations of *KSA*. Yes, our example is here a weak *IV* for guessing the value of the first byte of the secret key. ☺[13, page 53]

XORed* of *PRGA Usually data sent on the air by *WEP*, *WPA* or any other technologies that are used for building computers networks encapsulates *SNAP* packets.[5] Those headers always start with the hexadecimal value: AA_{16}. So we know now that the first encrypted byte of the packet is AA_{16}, that allow us to find relevant information by reverse engineering of the *PRGA* function.

As we previous say, at the first loop in the *PRGA* algorithm, we have: $i = 1$, $j = S[1]$, $z = S[1 + S[S[1]]$. And, we have 5% of chance with this weak *IV* that

[3] Read about the *Chopchop attack*[14] if you want to learn how to crack *WEP* thank to the *CRC32* checksum on *MPDU*.

[4] The previous explanation should be enough for finding the result. If you need more details in the computing process, please read [7].

[5] *WEP* is working in the link *OSI* layer, and *SNAP* is in the network one, so the one upper. More info on *Wikipedia*: `https://en.wikipedia.org/wiki/Subnetwork_Access_Protocol`

$S[1] = 3$, after the first swap (*S[0]* swap with S[1]): we can admit that $z = S[3]$ for the first iteration of *PRGA*.

With the previous example, we sniffed the value $A5_{16}$, so we just need to *XORed* with the *SNAP* header value for finding the first *z* value:

Sniffed byte	1	0	1	0	0	1	0	1
SNAP header	1	0	1	0	1	0	1	0
XORed z value	0	0	0	0	1	1	1	1

So we have $z = 15$.

Cracking our secret byte We know that *S[S[0] + S[1]]* = 15, at the end of the first *PRGA* loop.We also compute that $S[0] = 3$ and $S[1] = 0$, so we know that at the end of the loop, we have $S[3] = 15$.

We just have to come back to the *KSA* algorithm and continuing the loop like we kept it in stand-by two paragraphs before, and keeping in mind that at the end of this one, we will have $S[3] = 15$. This mean that the actual value of *S[3]* will be swaped with *S[15]*, because at the beginning, all values of *S* are ordered. We can thus say that we will have $j = 15$.

We had: $i = 3$, $j = 12$ and $S[3] = 1$, so the new value of *j* will be: $j = j + S[i] + K[i] = 12 + S[3] + K[3] = 12 + 1 + K[3] = 13 + K[3]$ And because we know that $j = 15$, we have: $K[3] = 15 - 13 = 2$. We now have 5% of chance to successfully cracked the first byte of the secret key.

This solution look like something impossible to use in practical because of the small probability of chances that we really found something. But keep in mind that $\frac{1}{255}$ *IVs* are weak, and that 5% of weak *IVs* will reveal one byte of the secret key that is only 5 or 13 bytes long... In average, we need 7 GB of data for cracking a password with this attack.

A STEP DEEPER IN THE SECURITY

WEP WEAKNESSES

As we saw, *WEP* has some important problems that make it dangerous:

1. The *IV* is too short, and unprotected from reuse. It means that lots of packets will have exactly the same encryption key.
2. The encryption key is too weak: it's just a concatenation of the (small) *IV* and the (small) secret key!
3. No keys management: all users use the same one, and we directly use the master key for encrypting. This means that we can't generate new encryption keys: if this key is cracked, the attacker has access to all the network and can decrypt all data sent by legitimate users.
4. We don't have any protection for message integrity: a cracker can forge new packets without being detected by the legitimate receiver.
5. We don't have any protection against message replay: with *WEP* a cracker can save a packet and send it back later to the access point. The packet will still be valid.
6. The authentication part is missing. How can we be sure that we are connected with the access point that we think?

So, you can guess that it was necessary to find a new solution for assuring the transmission security trough *Wi-Fi*. The problem with *WEP* was that the algorithm wasn't implemented in cooperation with security experts, and didn't have any interest by the cryptographic community. When the algorithm started to have some attention, it was when wireless network became common and the result was only humiliation.

A new talk group was created, and named *802.11i*. The only aim that this group had was two find a solution that can replace *WEP*. They created *WPA* and this time, with much more support and cooperation from experts.

WPA exists in two solutions, why? *WPA-TKIP* is a less secure one, but that still can work with old *WEP* designed hardware. This solution allowed users to keep their old material instead of throw it away. *WPA-CCMP* is a stronger solution, but that old material can't support[17].

THE MASTER KEY

Authenticate the user The biggest problem with *WEP* was that we don't have any security context. A security context, is the step before the start of sharing data: being sure about the identity of the correspondent. How can we be sure that we (as a client) are really connected to the access point of the network, and not to a cracker's one? And how an access point can be sure of the identity of all clients? That each of them are really allowed to connect them to the network?

In real life, we can take the example of an university that allow each students to access to all classrooms during all the day, and night. How can this university build a security context that allows only students to access to it? The common way to to that is to procure a student card to students that will have to be used for unlocking the university's door. So the authentication process can be your inscription in a class, that proves to the administration that you are really a student. Now, the administration can give you a card that you can freely use for going inside and outside the university whenever you want.

The master key Security relies heavily on secret key, and security is completely lost if this key is stolen. Indeed, the better way to be sure of the identity of someone is to share a secret with this person, and only this one. The key by itself doesn't mind at all, the only important thing is that they are the same. If we take the previous example with the university, we can considerate the card as the master key given to you by the

upper-level authentication server (the school administration).

A good way to do so, is to use a *master key* for proving your identity, and after that, use an ephemera key for the encryption process. Theses ephemera keys lives usually not longer than one connection, and can even be updated during the security context. They are called *session keys*, or *temporal keys*.

Using the master key in a way that protect it from discovery is very important. In a general rules, the master key is used the less possible, and never directly. It should only be used for generating sessions keys that will be used for encryption instead. *WEP* don't respect this rule: the master key is used during all the encryption process. It's a main reason about why we just need a single attack for compromising the entire network, and this as long as the administrator doesn't change this master key.

We can assume in IT that two kinds of master keys exist[5]:

Preshared keys means that the secret key will be shared between users and the access point with an external way. You need to share a password (the secret key) between all your devices and your access point. With this method, the authentication is totally bypassed because an user only needs this key for being allowed to accessed to the network. We will never really know the identity of the user, and the user will never know if he is connected to the real access point or to a cracker's computer that also know the secret key. In this case, we can admit this preshared key to be our master one, and it's with this method that all home wireless network works by default.

Server-based keys requires an upper-level authentication. Like we presented it previously, this solution enable the user authentication. It's something much more secure than simple preshared key because this method allow administrator to correctly manage a big network. With a single authentication server for all access point, we can easily grant or revoke access to each users. When an user is successfully authenticate by the server authentication, this one generate the master key that will be used for only this single connection. We will talk about upper-level authentication with more details afterward.

Access control

The main purpose of access control is to manage authentication. In other words: separate the world into "good guys" and "bad guys". Before starting, we will introduce some vocabulary:

- The *supplicant* is an user that want to have access to the network.
- The *authenticator* is an entity that control the "access gate".
- The *authentication server* decides who is good or bad.

IEEE 802.1X In 2001, the security standard *IEEE 802.1X* was released. The aim of this standard was to procure an authentication method for wired access to a network. In short, the aim was to have more security that just hide Ethernet ports and hope that only allowed persons will find it. Before *IEEE 802.1X*, you just had to plug your computer in a Ethernet port for having a full access in the network. With this standard, system administrators can manage access control.

When weaknesses in the *WEP* protocol appears, we started to think that it could be a good idea to extends *IEEE 802.1X* also for wireless access: *IEEE 802.11i* was highly inspired by it [1].

IEEE 802.11i *802.1X* was designed for wired network. Even if the authentication process is really great, one huge weakness exists with this standard in a wireless context. Indeed, it's really difficult to "stole" a connexion on a wire: we have to cut it and to install our computer between the node of the network (usually a switch) and the victim. If we admit that we can do that, the cracker just have to wait until a legit user authenticate himself on the network for stole the connection by sending new packets with the victim's *MAC* address.

In a wireless network, anyone near to an access point can sniff the air and wait until someone get authenticated on the network for stole his *MAC* address. For this purpose, *IEEE 802.11i* needs to implement an authentication process that will certify than we are really conversing with the user that we think. *WPA* use *EAP*[6] for this authentication protocol.

Access control and wireless With all home wireless access, the authentication server and the authenticator is actually the same device: your router. It means that we don't need a specific protocol for sending authentications messages before that the user is allowed to use the network. It also means that the user can't choose the authentication protocol that he wants, but as we say: we don't really care about all this stuff for domestic network.

Problems occurs when we start to be a quite big company than need to synchronize all access points. Here, we have to use an external and central server that the authenticator (the access point) will ask for granting access or not to each users. *IEEE 802.11i* uses the *RADIUS*[7] protocol for encapsulating all messages used for authenticating an user over *LAN*. It means all "conversations" between the upper-layer authentication server and the user that wants to access to the

[6] *EAP* was designed for working over *PPP* network instead of *LAN* like it's the case here. So the real name of the *LAN* version is *EAPOL* (*EAP* Over Lan).

[7] If you want to learn more about *RADIUS*, you are welcome to read [8, Chapter 8 – *RADIUS* part].

network, like the sending of the shared key, or other server-based systems like *TLS*.

Upper-layer authentication

The access control system is something implemented in the network layer that is considered as a low-level one because it depends on the *LAN* technology. Here, we will talk about others systems that are localized in the session layer, it's those ones that the access control system deals with.

We can so use a lot of several upper-layer authentication softwares, or also not at all. As we said previously: for home network, we usually just need a preshared key for granting access to the network, and it's enough. We will talk about *TLS* because it's an upper-layer authentication system that has to be supported by all devices that pretends to be *WPA* compatible, but others are quite popular, like Kerberos or *PEAP*.

TLS The main problem with pre-shared key is that you don't have any method for being sure that you are really connected to the access point that you think. What happen if one day another user that also know the preshared key decides to create a fake access point? You won't have any solution for being sure about who is who because both of them will be able to compute the session key...

A good solution at this problem is to use a asymmetrical cipher. With symmetrical ones, like *RC4*, you need exactly the same key for encrypting and decrypting a message. We saw that it is something dangerous, at least, during the authentication because everybody that know this key can pretend to be someone else that is also supposed to know it. Asymmetrical ciphers use the notion of *public* and *private* keys. The public one is use for encrypting, and the private for decrypting. This is a really nice feature because now, if you encrypt something with the public key of your access point, only this one will be able to decrypt it because he will keep this secret key secret.[8] [4]

As we said it, *TLS* is an upper-layer authentication system. This means that *WPA* will use the lower-layer *EAP* and *RADIUS* protocols for dealing during the authentication process: *EAP* messages will be send between the user and the authenticator (the access point), and *RADIUS* ones between the authenticator and the authentication server for encapsulating *TLS*. When the authenticity of everybody is proved, the authentication server generate a master key that will be send to the access point and the client. This master key will be the one use by both *TKIP* or *CCMP* encryption solution that we will discuss about now instead of a simple preshared one.

The session key

Because it's never a good idea to use directly the master key for encryption purpose, we will need to find a way for generating others keys that are based on this one. The secret key have to be 32 bytes long with *WPA*, and we need to derivate two other ones[9]:

- The *Data encryption key* that is 16 bytes long, will be used for encrypting data.
- The *Data integrity key* that is also 16 bytes long, and used for checking if the authenticity of the message is correct.

Those sessions keys have to change at every connection. It's something quite hard to find a way for generating unique keys with a one that can stay the same. A solution to this problem was found by generating two *IVs*. One is generated by the client, and the other by the access point. For being really sure that sessions keys will be unique, we also add *MAC* address from both devices. The final session keys depends of a mix of all this elements [3].

In summarize we have:

Master key *MAC* client *MAC AP* Nonce client Nonce *AP*	Data encryption key Data integrity key

The sharing operations of the data for creating those sessions keys is done via the *EAPOL* protocol. We won't go into details here [3, page 6].

Message integrity check This function has the same meaning that hash ones. It is used for checking if the integrity of the package is good or not.

The difference between hash and *MIC* is that a secret key is needed for computed the value. Thus, a hacker that doesn't know this key can't forge new packets. It was the solution implemented for trying to solve the 4^{th} problem of *WEP* that we enumerate.

TKIP

Temporal Key Integrity Protocol *TKIP*[10] still use *RC4* because we was looking for a solution that don't need to replace all hardwares equipments (encryption is done directly on hardware for performance purpose).

In a nutshell, *TKIP* is the same that *WEP*, but with a much stronger *RC4* key and with a *MIC* function that can certificate data. With *WEP*, the key is 5 or 13 bytes long, and only the initialization vector that is 3 bytes long is different in each encryption keys. Here, the key is 16 bytes long, and each of them is different for each packets.

[8] Well, *TLS* is more that only this. This solution can of course, also authenticate the identity of users, and we also don't talk here about certificates that is a quite cool and big part of *TLS*.

[9] We also generate some *EAPOL* keys, but we won't talk about that here.

[10] *TKIP* is usually called *WPA* for mainstreams utilisations.

RC4 MODIFICATIONS

The initialization vector *WEP* uses 3 bytes as randoms one, that are just concatenated before the master key. *TKIP* uses 6 bytes, that are mixed with the encryption key (more details in the next paragraph). The value of the *IV* will also be a monotonically increasing value for avoiding reuse. So now, it's not anymore a random value, but an incremental number. It's not a problem for the security because this *IV* is totally mixed with the encryption key, not just concatenated. Thank to his big length and the way that they are chosen, the first *WEP* problem that we introduce can be avoided.

The *IV* has also a second role, that actually justify why we need a incremental value: to be a sequence counter that indicate "where" we are in the transmission of packets. Concretely, because one *IV* will always be greater than the previous one, we can reject packets with a lower value, and in this time, avoid the 5^{th} problem of *WEP*: the packets reuse.[11]

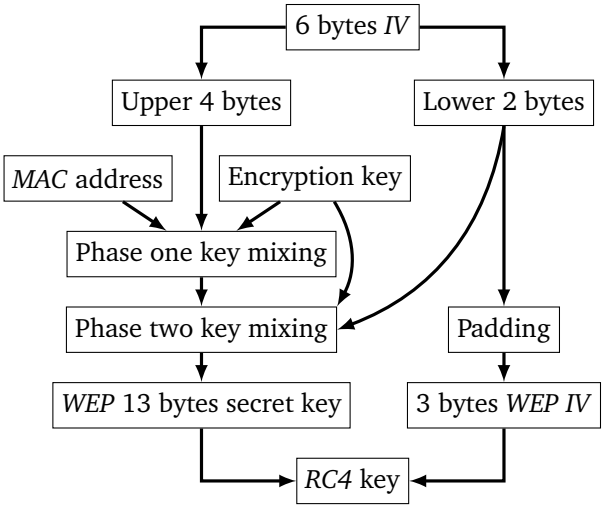

Figure 3: *IV* usage by *TKIP* for *RC4*

Generation of the encryption key As we already said it: *TKIP* was designed for working on slow computers and devices. It's definitely a good idea to always use an *IV* that is twice the length of the previous *WEP* one for enhancing his security, but the mixing operation with the encryption key has a big performance cost...

So for avoiding to have a too slow network, a new trick was found. The creation of the *RC4* key will be done in tree steps:

- The four last bytes of the *IV* are mixed with the session key and the sender *MAC* address of the network interface for producing a quite static key that only have to be recomputed each 2^{16} packets.
- We take the result of the first mixing phase, and we mixed it in a different way with the two lasts *IV* bytes and again with the session key. This step has to be done for each packet but is a way less power consuming than the first one. We now have a 13 bytes key that can be considerate as the *WEP* secret key for *RC4*.
- The first 2 bytes of the *IV* are padded out to 3 bytes in a way that avoids know weak key, and are used as the "usual" *RC4* initialization vector. This part of the key is also computed for each packets.

Adding the *MAC* address in the key generation is useful for being sure that the process will be different with each computers, even if the encryption key is the same. If you are wondering why we can use the key generated in the first step during 2^{16} packets, just think that *MAC* address and sessions keys are fully static, and also about the fact that now the *IV* is just an incremental number. The first step need the lasts bytes of the *IV*, not the two firsts ones: we obtain 2^{16} values, with the two bytes not used ☺.

We won't go in the details of the key mixing implementation because again, it's a quite complex process [16, page 54] but in summarize, we have:

$$\text{TTAK} \leftarrow Phase1(\text{TK}, \text{TA}, \text{TSCU})$$
$$\textit{RC4 key} \leftarrow Phase2(\text{TTAK}, \text{TSCL}, \text{TK})$$

With:

TK	Session key (Temporal key)
TSCU	Upper 4 bytes of the initialization vector
TSCL	Lower 2 bytes of the initialization vector
TA	*MAC* transmitter address
TTAK	Output of first phase

MICHAEL ALGORITHM

The *MIC* key is computed for each *MSDU* thanks to *Michael*'s algorithm and is added at the end of those ones and is also encrypted. Notice that the "old" *WEP CRC* is still computed and added at the end of each *MPDU*.

The data integrity key So as we said it, *Michael* need a secret key, that is 8 bytes long for computing a message that has an arbitrary size. After, the key is splited in two words that are each 4 bytes long. We will do the same treatment to the message: we will split it in blocks of 4 bytes. The message is padded at the end with the value $5A_{16}$, and between 4 and 7 zero bytes. The number of zero bytes is chosen so that the overall length of the message in a multiple of 4.

Implementation Let's assume that our 8 bytes key is splitted in two parts: k_0 and k_1, and that our message is splitted in m packages of 4 bytes, we have the algorithm 3.

[11] It's little bit more difficult in practicals because we usually send more that one single packet in the same time. Read [8, Chap 11, *TKIP* overview, *IV* as a sequence counter] for more details.

Algorithm 3: *Michael*

Data: Key(k_0, k_1)
Data: Padded message (m_0, m_1, ..., m_{n-1})
Result: *MIC* value(L, R)
(L, R) ← (k_0, k_1)
for $i \leftarrow 0$ **to** $n-1$ **do**
  $L \leftarrow$ XOR (L, m_i)
  (L, R) ← B(L, R)
end
return (L, R)

B function used in the algorithm 3 won't be presented here because it's a quite complex one that I can't fully understand now. Bits shifting are used, modulo additions and swapping operation are done. The important thing that we can saw in this algorithm is how the data are XORed with the secret key [15].

Michael's weaknesses Remember that we said that *TKIP* was implemented for allowing to old hardwares to still be usable. A problem with this hardware was that we have to use *RC4* for the encryption, but another one was that an algorithm *B* can't be very powerful because for that, we need to do lot of heavy computation like multiplications. Multiplications really need a lot of power to be done, and with old computers or network hardwares, it was something that we can't imagine because the debit would pass to 11Mbps down to less that a single one...

We finally find a method with only bits shifting, but the cost of this simplicity was that *Michael* become vulnerable to force brute attack. One solution against brute force was to introduce the concept of countermeasures.

Countermeasures Remember that we still have the *CRC* check during the *WEP* decryption process. So if the packet is altered by a transmission error, it is extremely unlikely than those randoms errors will generate a result that will pass this checksum test. Thus, we can be quite sure that if a *MIC* test failed, it's because a cracker is forging some custom packets.

Thus, the idea implemented to avoid bruteforcing on the integrity key was to oblige the attacker to considerably slowing down his attack speed. If more than one *MIC* failure is noticed during a single minute, the access point will be turned off during one other minute, and creating news connections afterwards with news sessions key. Even if *Michael* is quite weak, you need probably some months just for generating one valid packet with no more than one try per minute.

Maybe you think that an attacker can do a denial of service attack on the network by triggering countermeasures in a voluntary way? Actually, we just need to send a wrong *MIC* value to the access point every 59 seconds[12] to for having this one down indefinitely... In practical, it is more difficult than that because before the *MIC* check, the cracker will have to pass previous steps:

- The *IV* have to be a valid one, so one that are not too old.
- By modifying the *IV* we also need to find the good new *CRC* value for passing the (unsecure but still difficult) *WEP* authenticity test.
- Because the *IV* is directly used for encryption, the package still have to be valid after the decryption with the *IV* and content that we forged.

Yes, you probably have better tool at disposition if you want to *DDoS* a wireless access than *Michael* countermeasures.

THE ENCRYPTION PROCESS

Both of the encryption and integrity keys are generated during the initialization of the connection. For being able to generate them, we need the master key of the network, *MAC* addresses of the access point and the user, and two nonces: one produced by the access point, and the second one by the user. We won't go too deep in details [3, page 6].

When those keys are generated, we are able to start to encrypt packet with *TKIP*. When we got a packet from the operating system[13], we start to computed the *MIC* key. Because this *MIC* key needs the integrity key to be computed, it's a good way for authenticating the packet: only devices that know this key should be able to compute it correctly, so only the access point and the user.

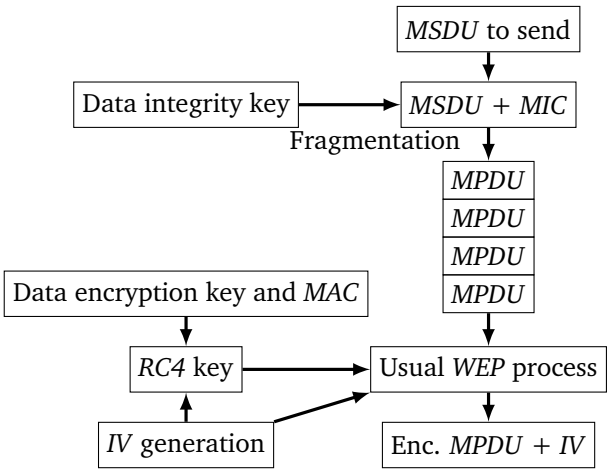

Figure 4: *TKIP* encryption

Now, we pass to the link layer of the *OSI* stack, so we need to fragment the *MSDU* in smaller packets: we will produces the *MPDUs*. It's each of those *MPDUs* that will be encrypted by the same way that with *WEP* and send on the air. The thing that changes is that here the *RC4* key is totally different for each packet (remember

[12] Actually not every 59 seconds because you will have at least to wait until a new connection is established for starting again to explode original *MICs* and provoking a new countermeasure, but it's for the style.

[13] Actually, we should say "when we got a packet from the transport layer of the *OSI* stack".

the key mixing steps). The data sent are so the encrypted *MPDU* with the *WPA* 6 bytes *IV* in plain text before it.

The decryption process

The decryption process is little bit more complex than the encryption one. At the beginning, we split encrypted data and the *IV* as we did with *WEP*, but after, we did the sequence test for checking if the packet is and old one or not. Remember that with *TKIP*, the *IV* is a monotonically increasing function for avoiding the reuse attack: if the new *IV* is superior than the old one[14] we accept the *MPDU*, otherwise not.

When the *IV* value is tolerated, we have again to compute the *RC4* key with the same data than for the encryption: the emitter *MAC* address, the shared encryption key, and the *IV* that we found before the *MPDU*. When the computation of the key is done, we use the *WEP* protocol for decrypting the *MPDU*.

When all *MPDUs* for assembling one *MSDU* are correctly received, it's time for doing the last verification: if the integrity of the package was preserved thanks to the *Michael* algorithm. If the comparison fails, we have a very high probability to be under attack, so the countermeasures will be applied if two fails occurred in the same minute. Countermeasures will shutting down the entire *TKIP* service during one minute before reinitializing each connection with new sessions keys. If everything is great, the *MSDU* is kept.

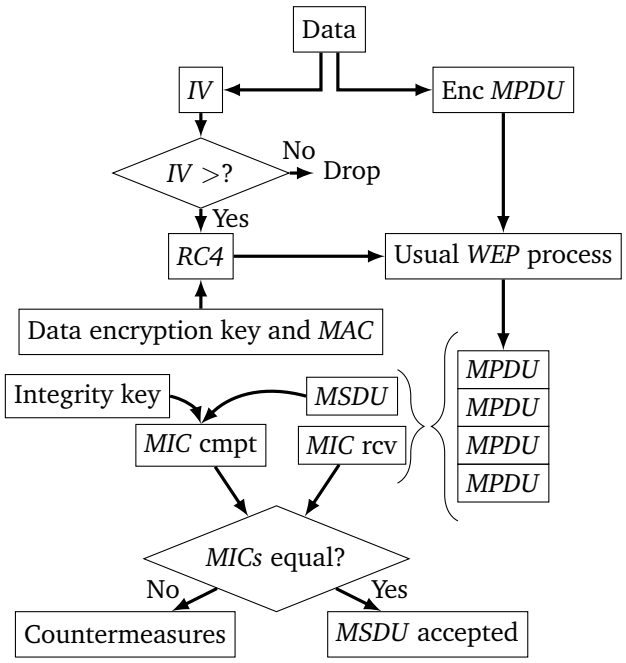

Figure 5: *TKIP* decryption

Breaking *TKIP*?

TKIP is a mastercase of retro-ingeniring. We have to use the *WEP* skeleton that have a lot of weaknesses for doing something that resists against weak keys attacks, replay attacks and packets reforge. As crazy at it could seems, *TKIP* do it a good way. Obviously, *TKIP* is a compromise between security and retro-compatibility. So ever if the solution is quite OK for the moment, it's not the most secure one that we could do.

Chopchop is already back In 2008 some persons already found a way to counter the *MIC* and replay protection for allowing an attacker to decrypt a packet with the old *WEP* Chopchop attack[14], and send forged packets that they want afterward. We just need an inactive channel that doesn't send and receive any packets during the attack[15] for bruteforcing the integrity key.

You just need to bruteforce some *MPDUs* that you "stole" on an other channel (with a higher *IV*, because if not, the packet will be rejected by the replay filter) by playing with the *CRC*. When the attacker will receive a *MIC* failure frame, he knows that the new *CRC* was good, and can wait one more minute for trying to decrypt another byte.

We need little bit more than 3 minutes for decrypting the *CRC* value of the *MPDU*. When we get the plain text *CRC*, it's not a big deal to furcebrute the content of packet until we match with the *CRC*. One *MPDU* is now decrypted, we just have to do the same with all others *MPDU* in the *MSDU*.

When the entire *MSDU* is decrypted, we have the plain text *MIC*, and the plain text content. We just have to forcebrute the *MIC* value now for finding the data integrity key. [12]

CCMP

CCMP[16] was built in the same period than *TKIP* without any started requirement: we don't have to use the same cipher than *WEP*, for example. All the mains features that was implemented for *TKIP* are obviously also implemented here. We have a sequence checker and a data integrity checker. The main difference between *TKIP* and *CCMP* is that now, we can use the bests solutions that exist for this purpose.

AES

Advanced encryption security In 1997, the *NIST* launched a contest for finding a new encryption standard that will substitute to the old *data encryption security* (*DES*) that started to be quite weak because of

[14] Actually, it's working with the notion of windows: more than one packet is send on the same time for being faster, so you can technically get a little bit smaller *IV* in a legitim context [8, chapter 11 *TKIP – IV* as sequence counter].

[15] *IEEE 802.11e* implement a 8 channels network that allow users to do quality of services by prioritizing packets. Unfortunately, almost all constructors don't use this functionality by default so even with a very used network, you have good chances to find 7 inactive channels on it.

[16] *CCMP* is usually called *WPA2* for mainstreams usages.

the small size of the keys that it uses. In 1999, five ciphers was selected for having a more deep inspection, for finding the best one.

Finally, the *Rijndael* cipher won this contest, and will be used by the *AES* standard in 2001. It's a totally free encryption system, open-source, approved by the *NSA* for protecting sensitives informations, low resources consumer and quite easy to implement. It's probably because of all those advantages that it became a very common and trusted encryption system.

AES "kernel" will replace the *RC4* cipher for the encryption process with *CCMP*. If you want to learn more about how *Rijndael* works, don't hesitate to look around [10], We won't explain here how it work because it's a way too big for this paper.

CCM

The mode *AES* is only the very center of the cryptographic process: the algorithm executed for transforming one byte in one other. We need something that send to *AES* the data to encrypt in a specific order, with a specific content, or to do more operations also on it. We call this function: the *mode*.

Each time that we use a encryption method, we also use a mode with it. Usually, the name of the entire encryption process will take a name composed with both the "kernel" function and the mode. Here, with the *AES* encryption and the *CCM* mode, the name of the full encryption process is *AES-CCM*.

Avoiding repeatability pattern *AES* will encrypt the message by blocks of 16 bytes. We have to be careful about one thing: if two of those blocks are exactly the same, the cipher text that will be produce will also be exactly the same. It is something that can be dangerous because we can guess thank to those repeatability patterns what kind of data we are transmitting (a lot of kind of packet have the same header value for example).

WEP and *TKIP* use the initialization vector for this task. Here, we will have to find a mode that supports also this kind of random value used for avoiding to encrypt totally the same values.

The *MIC* test The authenticity of the transmitted data isn't checked with *WEP*. We already explained why it was a critical issues, and how *TKIP* solved the problem thanks to *Michael*'s algorithm.

With *CCMP*, we also have to find a way. Remember that *AES* just encrypt blocks of 16 bytes one after another. If one attacker wants to change the content of one block in the middle of the packet, it's not *AES* that will have a problem because of that.

For performance purpose, the designers of *CCMP* decided to include this functionality directly inside the *AES* mode instead of having two totally separated parts like it is for *TKIP*.

AES-CCM Finally, the name of the mode that was chosen is *CTM + CBC*, shortened *CCM*. It's a fusion of two others mode: *CTM* and *CBC*.

The counter mode: *AES-CTM* It works like the encryption part of *TKIP*: we encrypt the plain text by *XORing* it with a random key (so even if we want to encrypt the same value, both results will be different).

The difference between *CCMP* and *TKIP* is that the first one uses *AES* instead of *RC4* for encrypting. We also encrypt the initialization vector that will be *XORed* with the plaintext data.

Note that the receiver has to know the value of the initialization vector. With *CCMP*, the value is send in the *CCMP* header of the packet, and the increment step has also to be know (as with *TKIP*, we just increment the *IV* for next ones) [9].

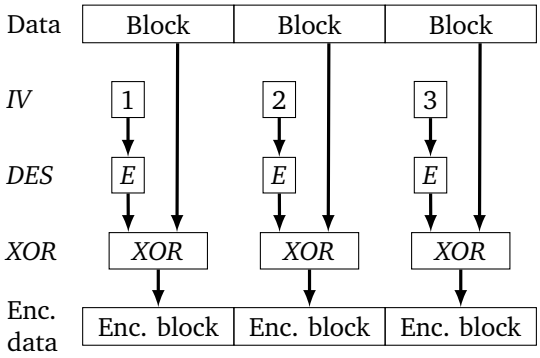

Figure 6: How *AES-CTM* works

The *MAC* computing mode: *AES-CBC* It is a *MIC*[17] international standard. In summarize, *CCMP* uses the algorithm in this way:

- We take the first *AES* block and we encrypt it. The first block will be a initialization vector.
- We *XORed* the result with the second block and encrypt again the result.
- We repeat this operation until the last block.

The main difference with *CCM* mode is that in this one, all blocks are connected. If we modify a single bit in the stream, the next encrypted data will have a totally different value than the one that they should have. Yes, your guess is right ☺ we can use the last encrypted block as the *MIC* value. It's actually precisely for that that we will use *AES-CTM* in *CCMP* [11, Chapter 4]. We will only include the last 8 bytes in the packet instead of the full 16 bytes block, but it still enough for having a strong key.

[17] In the cryptographic jargon, *MAC* is the same *MIC* in the networking one. We use *MIC* in the networking one because *MAC* was already "took".

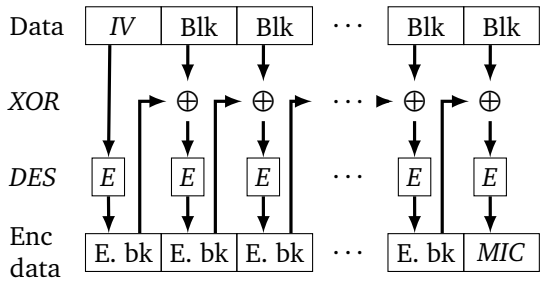

Figure 7: How *AES-CBC* mode works

AES-CCM will so use this two modes for producing an encryption content that will be always different (thanks to the *CTM* mode and the initialization vector) and *CBC* for computing the *MIC* value. But some hacks was also added:

- The possibility to change the *IV* suite for allowing us to use a totally different *IV* generation pattern for each packets.
- Both modes use the same session key.
- Setting *CBC* for making in fact that it authenticate more bytes that encrypted ones, for being able to also authenticate the *MAC* and *CCMP* headers.

As a general use, it's never a good idea to use the same key for encryption and for authentication. But even if we use the same session key with *CCMP*, it's not the case for the *IV* [8, Chapter 12 – *AES* overview].

Implementation

Authenticating data Authenticated data are not the same that encrypted one. Of course, encrypted data are authenticated, but not only. Indeed, we also authenticate the *MPDU* header that has to be in plain text for networking purpose (it contains the *MAC* addresses), and also the *CCMP* header that contain the packet number necessary for decrypting the content. We also have to note that *CBC* needs 16 bytes blocks. So if some blocks are too small (the packet is not a multiple of 16), a virtual[18] pad of zeros is added. We have:

$$\left.\begin{array}{r} \textit{MPDU}\text{ header} \\ \textit{CCMP}\text{ header} \\ \text{Plaintext data + padding} \end{array}\right\} \text{ MIC value}$$

The *CCMP* header We need a header that will contains a 6 bytes "packet number" that will be used for fighting against replay attack, and for computing the *IV* used in the encryption and *MIC* process. The header also contains which key was used for the encryption in case of a multicast message and the priority of the packet if 802.11e *QoS* is enabled on the network. Finally, we have a flag that are not so important (the value is fixed as 59_{16} for *CCMP*).

Initialization vectors As we said it, *IV* used in the encryption and authentication are not the same, but both of them are derived for the same packet number.

CTM IV	*CBC IV*
Flag	Flag
Priority	Priority
Source *MAC*	Source *MAC*
Packet number	Packet number
Counter value	Data length

Table 3: *IV* used for the *CTM* and *CBC* modes

As we can see on Table 3, content of *IVs* are almost the same. Note also that each values except the *CTM* counter used are contained in the *CCMP* or *MPDU* header.

The encapsulation In the jargon, this name is used instead of "encryption" but it's exactly the same. As we already said it, the complete encryption process is done in two step, plus a last one for assembling all elements:

1. We use *AES-CBC* for computing the *MIC* of plain text data, *MPDU* and *CCMP* header.
2. We use *AES-CTM* for encrypting data and the *MIC* value.
3. We forge correctly the packet as showed in the figure 8.

When it's done, we just have to send the encrypted packet to the receiver.

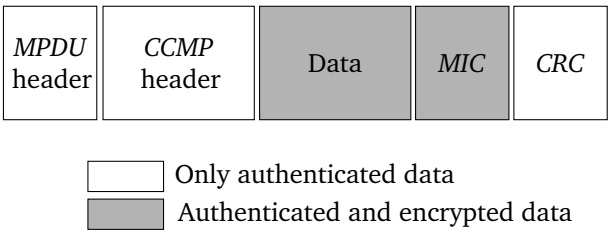

Figure 8: The *CCMP* packet

The decapsulation When a receiver receive a packet, the first thing he does is to do the sequence check. As we do it with *TKIP*, the packet will be drop if the packet number is equal or less than the previous one.[19]

When this test will be done, we just have to do exactly the same things than for the encapsulation, but in the inverse order:

1. Computing of the *CTM IV*.
2. Decryption of the encrypted datas with *CTM*.
3. Computing of the *CBC IV*.

[18] *Virtual* means that zeros are not really added inside the packet, and thus, sent to the receiver, but just used by *CCM* for computing the *MIC*

[19] Again, we have the same problem if more than one packet is sent in the same time. So as for *TKIP* we use the notion of windows for allowing the sender to still send some "not to old packets

4. Authentication of all data.
5. Sending the decrypted *MPDU* to the upper *OSI* layer.

Today, *CCM* is still really strong, and the only way to break it is by bruteforce, but with big keys like the ones used here, we can still sleep quietly.

I hope you enjoyed this small introduction on the *Wi-Fi* technology ☺.

REFERENCES

[1] 802.1x and network access control. Technical report, ForeScout Technologies, Inc., 2012. http://www.forescout.com/wp-content/media/802-1XTechNote.pdf.

[2] Wesley Addison. *Hacker's Delight second edition – Chapter 14: Cyclic Redundancy Check*. 2012. http://www.hackersdelight.org/crc.pdf.

[3] Devin Akin. 802.11i authentication and key management (*AKM*). Technical report, Certified Wireless Network Professional, May 2005. http://www.cwnp.com/wp-content/uploads/pdf/802.11i_Key_Management.pdf.

[4] Nassar Ikram Athar Mahboob. Transport layer security (*TLS*) – a network security protocol for e-commerce. Technical report, National University of Science & Technology. http://www.researchgate.net/publication/216485703_Transport_Layer_Security_(TLS)--A_Network_Security_Protocol_for_E-commerce/file/79e415093d5e82cbb2.pdf.

[5] Technischer Bericht. Comparison studies between pre-shared key and public key exchange mechanisms for tls. Technical report, January 2006. http://www.net.informatik.uni-goettingen.de/publications/1281/ptls-ifi-tb-2006-01.pdf.

[6] Cybergibbons. Why *XOR* alone is an incredibly bad encryption technique, April 2013. http://cybergibbons.com/2013/04/12/why-xor-alone-is-an-incredibly-bad-encryption-technique/.

[7] Seth Fogie Cyrus Peikari. Cracking *WEP*. Technical report, Airscanner, 2003. http://www.airscanner.com/pubs/wep.pdf.

[8] etutorials.org. 802.11 security, wi-fi protected access and 802.11i. http://etutorials.org/Networking/802.11+security.+wi-fi+protected+access+and+802.11i/.

[9] R. Housley. Using aes counter mode with ipsec encapsulating security payload. Technical report, January 2004. https://tools.ietf.org/html/rfc3686.

[10] Vincent Rijmen Joan Daemen. *AES* proposal: Rijndael. Technical report, September 1999. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[11] Yehuda Lindell Jonathan Katz. *Introduction to modern cryptography*. August 2007. http://www.cs.umd.edu/~jkatz/crypto/f12/chap4.pdf.

[12] Erik Tews Martin Beck. Practical attacks against *WEP* and *WPA*. Technical report, TU-Dresden, TU-Darmstadt, November 2008. http://dl.aircrack-ng.org/breakingwepandwpa.pdf.

[13] Forgie S Peikari C. *Maximum Wireless Security*. 2002. http://flylib.com/books/en/4.234.1.53/1/.

[14] Vivek Ramachandran. Korek's chopchop attack. http://wipkip.nikhef.nl/events/securitytube/DVD/unpack/pages/018.html.

[15] Jennifer Seberry. Security analysis of michael: the ieee 802.11 message integrity code. Technical report, 2005. http://www.uow.edu.au/~jennie/WEB/WEB05/Michael.pdf.

[16] IEEE Computer Society. Ieee standard for information technologyi – part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. Technical report, 3 Park Avenue, New York, NY 10016-5997, USA, July 2004. http://standards.ieee.org/getieee802/download/802.11i-2004.pdf.

[17] Erik Tews. Attacks on the *WEP* protocol. Technical report, TU Darmstadt, 2007. http://eprint.iacr.org/2007/471.pdf.