WLAN Security Analysis and Construction (Laboratory Report)

Group 18: Matteo Ridolfi, Enrico Ciccarelli, Antonio M. Franques Garcia
matteor@stud.ntnu.no, enricoc@stud.ntnu.no, antoniof@stud.ntnu.no

October 12, 2014

1 Challenge 1: WEP analysis

1.1 Introduction and theoretical background

Wired Equivalent Privacy (WEP) is a security algorithm for wireless networks. Roughly summarizing, the standard describes two parts of WEP security: the first one is the authentication phase and the second is the encryption. In the authentication phase no secret tokens are exchanged and that makes this phase quite useless. Even worse, if an attacker is eavesdropping it can help him breaking the security measures (see [1]). Regarding the second encryption phase, the standard uses the stream cipher RC4. A simple rule is that the same key stream (used for encrypting the data packet) shouldn't never be used twice (see [2]). But even with all the possible precautions that the IEEE 802.11 Standards Committee took at the moment of its conception, WEP became and remained a very easily security protocol to attack. In this challenge we will show how the WEP key can easily be obtained and which weaknesses we exploited for fulfilling that aim.

1.2 Equipment used

To be able to perform the attack we used the following instruments:

1. Two Desktop PC with Ubuntu Linux
2. One Wireless Network Interface Card in each PC (NIC)
3. `Kismet` tool
4. `aircrack-ng` suite

1.3 How we performed the attack

The first tool we used was `Kismet`, a wireless network detector program. We needed it because the network was initially hidden (no SSID broadcast). After typing the command `kismet_server` in one shell and `kismet_client` in another, a list of the received networks showed up and hence we were able to find (and get information about) our target. The name of the target network is `ttm4137_lab` (SSID). Before going on with the attack we checked if our software and hardware were properly set up. There are three different modes for our NIC: `Managed`, `Master` and `Monitor`. It must be in `monitor mode` before start listening to traffic. Using `airmon-ng` we created a new interface (`mon0`) listening to a specific range of frequencies (channel 6, the one that our target network used): `airmon-ng start wlan0`, then `iw dev mon0 set channel 6`. Next step was simply sniff the traffic with `Airodump-ng` (the result was saved in a `.cap` file):

```
airodump-ng --write filesniff.cap --channel 6 --bssid
  98:FC:11:B2:D2:67 mon0
```

This kind of file can be fed and analyzed with `aircrack-ng` or `Wireshark`. The basic idea behind this first step was to capture as more packets as possible in order to be able to perform the attack. Obviously what we captured was needed to be saved somewhere, that's why we run `Airodump-ng`, for saving the traffic in a file named `filesniff.cap`.

After a certain time we felt that `airodump-ng` had gathered enough packets so we tried the first `PTW` attack (see [4]) by running `aircrack-ng`. Our first attempt failed. We had only 511 IVs and that was not enough (not even close). At this point we needed to figure out something to speed up the process. There are several techniques available for this, those we tried are:

Fake authentication That's the first step in order to avoid that the AP drops the packets that we are going to inject. If there would be a legitimate client already associated to the AP we could spoof his MAC address and inject packets in his name (the AP would not drop the packet because he would see that the source MAC is associated). However, if there is no legitimate client associated we should use this technique (fake authentication) for associating with the AP (see [3]). Notice that the fake authentication attack does not generate any ARP packets. For its performing we typed the following command:

```
aireplay-ng --fakeauth <delay> -a <bssid> -e <essid> <interface>
aireplay-ng --fakeauth 1000 -a 98:FC:11:B2:D2:67 -e ttm4137_lab mon0
```

ARP-request reinjection This is the tool we used after having performed the fake authentication. It simply sniffs for ARP-request packets and *injects* them into the networks. It exploits the weakness of WEP against a replay attack. The procedure is to just capture an ARP packet sent by a legitimate client and resending it to the AP in order to capture the AP responses to these (with a new IV every time) (see [3]). We first had to perform a fake authentication because the access point will only accept and repeat packets where the sending MAC address is *associated*. For performing this reinjection we typed the command:

```
aireplay-ng --arpreplay -b 98:FC:11:B2:D2:67 mon0
```

where the MAC address specified is the BSSID of the target network and `mon0` is the interface used. Afterwards we performed again the `PTW` attack with `aircrack-ng`. We succeeded in this second attempt, the found key was: `B++>v\/\l\#[Uj`. It used 111296 IVs (although we know that even with less could be also possible) and it did it in less than 2 minutes.

Deauthentication The aim of this attack is to disassociate the client who is currently associated to the AP. One of the main reasons to do that is to generate new ARP-requests (Windows clients sometimes flush their ARP cache when disconnected) and hence use them for performing an ARP-request reinjection (see above paragraph and [3]).

Interactive packet replay It's used to inject a packet of your own choice (not necessarily ARP). We first typed the following command for capturing traffic:

```
aireplay-ng -2 -b 98:FC:11:B2:D2:67 -d FF:FF:FF:FF:FF:FF -t 1 mon0
```

After having read around 100 packets it asked us to choose the packet to be injected and so we did. But since only certain packets can be replayed successfully, we used the Modified Packet Replay:

```
aireplay-ng -2 -b 98:FC:11:B2:D2:67 -t 1 -c FF:FF:FF:FF:FF:FF
  -p 0841 mon0
```

where `-p 0841` sets the Frame Control Field such that the packet looks like it is being sent from a legitimate client. If it works, the AP will answer with a reply and thus, with a new IV.

Fragmentation attack It cannot be performed if we don't receive at least one data packet. Let's assume we are in the condition to perform this attack: it attempts to send ARP and/or LLC packets with known contents to the AP. It exploits the WEP weakness against plaintext-ciphertext attack, which let us obtain the key stream used for a certain IV and hence letting us forge our own packets (see [1]). The scope of this is to obtain as many answers as possible from the AP. This cycle is repeated until more or less 1500 bytes of PRGA are obtained. This was the command line used for this first step:

```
aireplay-ng -5 -b 98:FC:11:B2:D2:67 -h 00:18:DE:A9:77:3E mon0
```

Pretty soon it asked us to use a packet, that resulted in a saved key stream in a `.xor` file. So we could use now `packetforge-ng` to build our own packet with the key stream obtained. To be more specific `packetforge-ng` creates encrypted packets that can subsequently be used for injection. There is the possibility to create different types of packets like ARP-requests, ICMP and also custom packets. We decided to use the first type. Following the instructions found on [3] we typed the command,

```
packetforge-ng -0 -a 98:FC:11:B2:D2:67 -h 00:18:DE:A9:77:3E -k
  192.168.1.100 -l 192.168.1.1 -y fragment-1003-142737.xor -w arp-request
```

Now we could inject (as many times as we wanted) this ARP-request packet using the command,

```
aireplay-ng -2 -r arp-request mon0
```

KoreK chopchop The principle behind this attack is the same of the previous one (we obtain the key stream associated to a certain IV). It additionally checks if the checksum of the header is correct after guessing the missing parts of it (see [3] for more information). First:

```
aireplay-ng -4 -b 98:FC:11:B2:D2:67 -h 00:18:DE:A9:77:3E mon0
```

this means that all the packets will be sent with the source MAC specified by `-h` and the destination MAC will vary with 256 combinations. In 21 second it created the resulting file `replay_src-1003-152718.xor` (containing the key stream), afterwards we used `packetforge-ng` in the same way as in the previous paragraph. This attack took much more time than the previous one.

1.4 Answers to the questions

Q1.

- SSID=`ttm4137_lab`
- BSSID=`98:FC:11:B2:D2:67`
- Channel=6 (2437 MHz)
- Encryption=WEP
- Associated clients=They varied in the lab
- WEP key=`B++>v\/\l#[Uj`

- Packets needed=52836 (after succeeding in the first attempt with 111296 IVs we tried it again with the other injection methods and we saw that with only 52836 IVs we could also succeed)
- Web page client browsing: after getting the WEP key we fed the initial traffic captured with `airodump-ng` to `Wireshark`, we indicated him the WEP key used for decrypting the packets and we could see in plaintext the kind of traffic that the users were generating at the moment of our attack (although it was diverse we could see some HTTP traffic on it)

Q2. Yes, but it probably will take more time. The completely passive `PTW` attack can only be run if the clients associated to the AP are generating enough traffic by themselves.

Q3. A fast answer would be that it is because WEP doesn't have any protection against a packet replay attack. We captured one encrypted ARP-request packet (how did we know that it was an ARP-request packet? Because the structure of an ARP-request packet is always the same, including the length) and we just kept resending it to the AP (the AP didn't drop it because the source MAC address was already associated).

Q4. Deauthenticate them (the legitimate associated clients) in order to make them generate some traffic, hopefully also an ARP-request that hence will let us inject as much traffic as we want.

Q5. If not clients are associated but we see (for example with `airodump-ng`) that the AP is sending some valid data, we can use an attack like `KoreK chopchop` or `Fragmentation attack` in order to get the keystream used for encrypting those packets (with their corresponding IV) and use that keystream (with that corresponding IV) obtained for forging our own packets for injection.

Q6. No protection against a Replay attack (the same IV can be used as many times as desired), so we could reinject packets for generating data or reuse a keystream (and its corresponding IV) for forging our own packets. About the RC4 cipher, certain key values (combination of weak IVs and password) do not produce sufficiently random data for the first few bytes, hence a PTW attack can be performed for easily obtaining the password (based on the FMS attack). The PTW attack can also be performed due to the fact that the length of the IV is too short and hence the IVs are reused after some time at high speed bitrate. Since the key used as a seed of the RC4 cipher is directly the master password (instead of deriving a temporary key every time from it) the PTW attack results on obtaining the master password by itself (if we would have used temporary keys we could only obtain de temporary key but not the master password where it was derived from). No encrypted integrity protection (message modification flaw), this lets an attacker modify part of the packet and change its integrity value in order for the AP to do not initially drop it (weakness used for instance in the `chopchop` attack, which results on obtaining the keystream used for encrypting that packet).

Q7. We cannot consider a network using WEP as a secure network, the only way that we can provide more safety to the encrypted data is by using other security protocols in the upper layer, such as SSL. By the way, there's no reason today for using WEP (in case that we have to buy a new device) since other much reliable security systems are available (see [5]).

Q8. No, TKIP changes the RC4 seed key for every packet, has replay attack protection and the amount of IVs that produce the weak keys has been reduced. Also the length of the IV has been increased in a way that now they are not reused (at least not in a reasonable time). RC4 is reused in WPA in order to keep compatibility with older systems and at the same time solve some of the WEP problems (by the use of TKIP).

2 Challenge 2: Password-based PSK analysis

2.1 Introduction and theoretical background

In this activity we set up a WPA-PSK protected WLAN and performed a password dictionary attack against it. WPA (WiFi Protected Access) was intended to replace WEP. The WPA protocol implements part of the IEEE 802.11i standard. Specifically, the TKIP (Temporal Key Integrity Protocol) which employs a per-packet key, meaning that it dynamically generates a new 128-bit key for each packet (instead of using the same every time). PSK (Pre-Shared Key) mode is designed for small networks that don't require the complexity of an 802.1X authentication server. Each wireless device belonging to the same network encrypts the network traffic using a Pre-Shared 256 bit key (that is the same for every device of the network), which can be entered either as a 64 hexadecimal digits, or using a passphrase from 8 to 63 ASCII characters. We will use the second approach.

2.2 Equipment used

To be able to perform this activity we used the following:

1. Two desktop PC with Ubuntu Linux
2. Personal WLAN-enabled device to connect to the AP
3. `aircrack-ng` suite
4. `John the Ripper` tool

2.3 Setting up the Access Point

We used the server tool `hostapd` in conjunction with a wireless NIC in `master mode` to implement the AP. First we set up a bridge between the Ethernet and the wireless interface of the AP by creating the file `brup.sh` (with the content specified in the lab instructions), then making it executable, and finally running it by typing `chmod +x brup.sh` and `sh brup.sh` respectively in the terminal. We also created the file `brdown.sh` (also from the content of the lab instructions) for stopping the bridge when needed. Then we used the `hostapd` tool to set up the WPA access point and to set the wireless NIC into `master mode`. It is a user space *daemon* for access points and authentication servers which implements: IEEE 802.11 access point management, IEEE 802.1X/WPA/WPA2/EAP authenticators, RADIUS client, EAP server and RADIUS authentication server. All required configuration of `hostapd` is based on the file `hostapd.conf`, so we set some parameters on it in order to reach our purpose (see Appendix A).

After having edited the file, we started the daemon in `debug output mode` by typing in the terminal `hostapd -d <pathToConfigFile>`. Then we used another device to cennect to the just created AP in order to be sure that it worked.

2.4 How we performed the Password Dictionary attack

The first thing that every password dictionary attack must have is a good dictionary (list of possible passwords). In this case we guessed (actually we knew) that the password could be

an english word in conjunction with some numbers and regular symbols. For obtaining a good dictionary we first need a list of possible words, that in this case we downloaded from internet (for example from [6]), and a good way of mixing this words with some numbers and symbols. For this last purpose we used John the Ripper (`john`), which we fed with the initial list of words as well as with some rules about how to apply different variations to our dictionary words (see, for example [7]). The command used for this last was:

```
john -wordlist=words.lst -stdout -rules > extendedwords.lst
```

where, as we said, we extended the dictionary contained in the file `words.lst` into a new one contained in the file `extendedwords.lst`.

Once we got our dictionary ready we used `airodump-ng` to sniff the packets from the WLAN in order to capture the 4-way authentication handshake of some legitimate client. Also in this case we can use an active attack tool to speed up the process, such as the deauthentication attack of `aireplay-ng` (see [3]). After supposing that we already got a 4-way handshake in our dump (because sometimes it doesn't show up on the screen), we were ready to run the attack using `aircrack.ng`. For this we executed the command:

```
aircrack-ng -a wpa -b Try to hack me!!
  -w extendedwords.lst filesniff-01.cap
```

where `-a wpa` means WPA attack mode. Notice that in this step we could have run `john` with other rules to maybe have a better dictionary and hence speed up the process, but we didn't do it because the password was found with the default rules in around 5 minutes. The speed of this attack will directly depend of how good we are creating a good dictionary.

2.5 Answers to the questions

Q9. TKIP adds a series of corrective measures around the existing hardware to solve the problems of WEP. It adds a message integrity protocol (MIC) field to prevent tampering (this way we solve the `bit-flipping attack` problem), that can be implemented in software on a low-power microprocessor (such as the WEP devices have). It changes the rules for how IV values are selected (it solves the weak keys problem) and use the IV as a replay counter (for avoiding replay attacks). It changes the encryption key for every frame. It increases the size of the IV to avoid ever reusing the same IV.

Q10. In the PSK scenario the PMK (Pairwise Master Key) is derived directly from the passphrase. Specifically, the PMK is generated using the PBKDF2 (Password-Based Key Derivation Function) whose input information parameters are the passphrase, the SSID, the SSID length, the number of function iterations (4096) and the length of the desired output key (256 bit). The PTK (Pairwise Transiet Key) is then derived from the PMK using the 4-Way Handshake with other information such as the MAC of the authanticator, the MAC of the supplicant, an authanticator's nonce (ANonce) and a supplicant's nonce (Snonce). A brute-force attack is possible because both MAC addresses and both nonces travel in a plaintext. Hence, since the MIC (Message Integrity Code), which is calculated with the PMK, is included in the last three messages of the 4-Way Handshake, we can try with a brute-force attack which key is used to create that MIC value. This key is the one we are looking for. Notice that once we have captured the whole 4-way handshake we can perform the attack completely offline (because we don't need anything else from the network).

Q11. Yes, if the attacker captured the encrypted traffic from some session, also captured the Anonce and Snonce used for deriving the PTK in that session and the MAC addresses of both Authenticator and Supplicant of that session and then he got somehow the password used to derive the PTK, the intruder can now know what PTK was derived from the password in that session (because he knows both mac addresses and both nonces of that session) and hence decrypt the messages encrypted with that PTK in that session.

Q12. No, because both WPA2-PSK and WPA-PSK use the same way of obtaining the PTK from the PMK, which is by the 4-way handshake. It is clear that AES is a better cipher than RC4 but the encryption process comes later, on the part where we encrypt the data. Remember that for the 4-way handshake the only field that brings some protection to the conversation is the MIC, and for obtaining this it doesn't matter if the later cipher is going to be RC4 or AES.

Q13. First of all, what is a precomputed database of PMKs? It is a database that specifies, for a given passphrase, SSID (name of the target network), SSID length, number of function iterations desired and length of the desired output key, the corresponding PMK. That way we can precompute the PMKs database that we are going to try by brute force in our attack. Computing the PMK corresponding to a passphrase is what takes most part of the time during the attack and this is the reason why we would like to have it in advance (and also because that way we can reuse it in another moment). However, the disadvantage of this is that every database can only be used for one SSID, so even if two different networks use the same passphrase they will need different databases because (probably) the SSID will be different.

Q14. The WPA passhphrase should be at least 14 ASCII characters long. See calculations on Appendix G.

3 Challenge 3: Construction of an RSN BSS

3.1 Introduction and theoretical background

In this activity we are going to set up a very Robust wireless Security Network (RSN). The authentication is going to be deployed by EAP-TLS over an external Radius server and the encryption is going to be done by CCMP.

For accessing to the network the supplicant will have to demonstrate somehow that he is an allowed user, in this case he is going to do that by providing his public key/certificate signed by the CA (Certification Authority). Why the certificate has to be signed by the CA? Because the CA is the entity that both supplicant and AS (Authentication Server) trust. Also the supplicant will be able to confirm the authenticity of the AS due to the fact that the AS will also provide his certificate signed by the CA. We have mutual authentication.

If it would be the case of a real company or office and we would like to add a new user (for example a new employee), we would probably already have an AS created and a CA designated, so we should only have to ask the CA to confirm our identity by signing our certificate and then connect to the network by providing our signed certificate to the AS. However in this laboratory we are going to start from scratch and hence we are going to create the CA that both AS and supplicant are going to trust, then we are also going to create the AS and make his certificate get signed by the CA and finally we are going to create the supplicant's certificate and also get it signed by the CA.

At the same time we will need an authorizer, which will be the dealer between the AS and the supplicant. This dealer is going to be the AP (Access Point), it is the guard that will

only let the supplicant speak with the AS and not with the rest of the network (not until the supplicant accomplishes his authentication process). In this laboratory we are also going to set up this AP.

Note: although the AS and the authorizer (AP) will act as a two different entities we are going to configure them both in the same machine, it is something usual in small environments.

3.2 Procedure to follow

3.2.1 step1 - Create the CA's private key and self signed certificate

the tool that we are going to use for this is called `openssl`, but before we can start using it we have to make some preparations:

1. create a folder structure where the CA material is going to be organized (the structure and names of the folders are arbitrary, we use it in order to separate the different elements that we are going to create). The structure will have a root folder called `CA`, and hanging from that we are going to have `newcerts` and `private`. `newcerts` will contain the certificates that the CA signs and private will contain the private keys that we create (from the CA, AS and supplicant); these private keys are going to be protected with a password (they are not going to be stored just in a plaintext)

2. initialize the files that `openssl` need to lay on for creating the certificates, they are `serial` and `index.txt` (`serial` is going to be initialized to `01` and every time we create a certificate it is going to increase by one unit; it is used for following a sequence)

3. prepare the `openssl` configuration file (`openssl.cnf`) and store it in `CA` folder. For doing this we have downloaded the `openssl.cnf` file from [12] and modified some of the lines (just in order to avoid type them in every certificate). See Apendix B.

Once we have performed these preparatives we can create the CA's self signed certificate with the following instruction:

```
openssl req -new -x509 -eyout private/cakey.pem -out cacert.pem
  -config ./openssl.cnf
```

where `-new -x509` creates a new self signed certificate, `-keyout private/cakey.pem` specifies the path where the CA's private key is going to be stored, `-out cacert.pem` specifies the path where the CA's certificate is going to be stored and `-config ./openssl.cnf` specifies the path where to find the openssl configuration file. The CA's certificate (`cacert.pem`) is the file that we will have to distribute to the AS and supplicant/s (although in this case the AS is in the same computer than the CA and there is only one supplicant).

The password that this command execution asks to us is used to protect the CA's private key. Let's suppose that somebody manages somehow to get into the system and find the private key, if this private key would be in a plaintext this intruder could use it for signing certificates without authorization, this is why we use a password to encrypt the private key and at least, in case that the intruder would find it, he would not be able to use it without this password (and this password is something that the CA administrator should store in a safe place).

3.2.2 Step 2 - Create the AS's private key and certificate and make it signed by the CA

This procedure has to be performed in three sub-steps. Ffirstly, the AS will create his pair of keys (private and public); he will also prepare and give the certificate to be signed to the

CA (this certificate includes the AS public key and also some data about him). Secondly, the CA will check somehow that the data given by the AS is correct and hence he will sign AS's certificate. Lastly the CA will send/give back somehow the signed certificate to the AS. These are the two instructions used for the first two sub-steps, correspondingly:

```
Openssl req -new -out asreq.pem -keyout private/askey.pem
   -config ./openssl.cnf
Openssl ca -out newcerts/ascert.pem -config ./openssl.cnf
   -infiles asreq.pem
```

where `askey.pem` is the generated AS's private key, `asreq.pem` is the certificate before being signed and `ascert.pem` is the signed certificate

3.2.3 Step 3 - Create the supplicant's private key and certificate and make it signed by the CA

Analogously to the previous step we initially create the supplicant's pair of keys, prepare the certificate to be signed by the CA, then make it signed by the CA (once he has check our identity) and finally give it back to the supplicant. The instructions used in this case are also analogous to those two used in the previous step, as it follows,

```
Openssl req -new -out supplicantreq.pem -keyout private/supplicantkey.pem
  -config ./openssl.cnf
Openssl ca -out newcerts/supplicantcert.pem -config ./openssl.cnf -infiles
  supplicantreq.pem
```

3.2.4 Step 4 - Setting up the AS (Authentication Server)

As we have mentioned in the introduction of this activity, the AS is the responsible for authenticating every user (supplicant) before letting him access to the network. The AS will talk to the supplicant, ask him to prove that he has access to the network and finally let the authorizer know if the supplicant is finally allowed or rejected. Our job in this step is to configure the Radius server, that is, to prepare the AS to receive, understand, treat and reply the requests from the authorizer (sent by the supplicant). Why Radius? We are setting up a Radius Server because the communication between the authorizer and the AS is going to be over Radius (TLS over EAP over RADIUS). The tool that we are going to use for this is `Freeradius`. It is already installed in the computer and almost ready to be used, we only have to modify a couple of its configurations files (`eap.conf` and `clients.conf`, both allocated in `/etc/freeradius/`). See Appendix C. Once the configuration is done we can run and monitor the server by the instruction `freeradius -X` (notice that the server activity starts to show up on the screen).

3.2.5 Step 5 - Setting up the authorizer

As we have also mentioned in the introduction, the authorizer plays the role of the guard, letting the supplicant only speak with the AS and forbidding him the access to the rest of the network until he proves his authenticity. In this case the authorizer will be the AP, an AP that (as same as in the Challenge 2) we are going to configure with the hostapd tool. Just in order to summarize we recall that this AP will deploy EAP-TLS with an external Radius authentication server (the AS) and CCMP as the encryption protocol.

Let's be sure then that the `hostapd.conf` file contains the lines mentioned in the Appendix D. Notice that the shared secret is the same than the one configured in the AS (`/etc/freeradius/clients.conf`).

3.2.6 Step 6 - Connecting a user to the network

Now that both AS and AP are set up and all certificates are created, signed and distributed, let's configure the supplicant and make him ready to access to the network. For doing this the supplicant can use a tool called `wpa_supplicant` that will automate the process of authentication, association and IP-address assignment. Since the supplicant has already installed this tool on his computer we will only have to execute the following instruction (the result of the process will show up then on the screen and if we succeed we are completely ready to access to the network),

```
wpa_supplicant -D nl80211 -i wlan0 -c path_to_config_file -d
   -K dhclient wlan0
```

Notice that the instruction requires us to indicate some data about the AP that we want to connect with. This configuration file must contain,

- the SSID of the network
- path to the CA's certificate (that will use for checking the legitimacy of other certificates)
- path to its own certificate (signed by the CA)
- path to the file that contain his private key, and of course know the password used for encrypt it

The structure that this file must follow can be seen in the Appendix E.

This is probably the moment when we wonder ourselves what happened internally in that `wpa_supplicant` tool. How both AS and supplicant managed to authenticate each other? On Appendix F we can see a possible pseudo-dialog between the supplicant and the AS (where the AS is authenticating the supplicant) that helps us understanding it better.

3.3 Answers to the questions

Q15. In a small environments, such as our homes or small offices with no many users. In general all those places where we can easily control the amount of allowed users/clients.

Q16. A Fake AP attack. If we would trust and connect through a fake AP it could eavesdrop all the content sent by the attacked client (although the upper layers will probably also have their own encryption but it is not good from a security point of view that an attacker can perform eavesdropping)

Q17. First of all it must be said that by WPA2-Enterprise we refer to the version with 802.1X/EAP authentication (in contrast to the PSK method). About the all possible EAP authentication protocols we can enumerete the followings ([1, 8]):

- EAP-TLS (the one we used in this activity)
- EAP-TTLS/MSCHAPv2 (first, a secure connection is established, then the server authenticates the client)
- PEAPv0/EAP-MSCHAPv2 (also use a secure tunnel to protect the authentication transaction)
- PEAPv1/EAP-GTC
- EAP-SIM (a SIM card is used instead of a pre-established password between the client and the server)
- EAP-AKA (authentication over UMTS)

- EAP-FAST (designed as a replacement to the weak LEAP, it uses a protected access credential to establish the secure tunnel)

Q18.

- TLS: protocol used for authenticate both Supplicant and AS by the use of signed certificates.
- EAP: is the protocol that performs the role of a guard, only letting pass the upper-layer protocol (TLS in our case) messages between the supplicant and AS. Blocking the rest of the traffic coming from the supplicant (until it correctly authenticates).
- Radius: Is the encapsulation protocol used between the Authorizer and the AS, it will encapsulate the EAP messages (that at the same time contain the TLS messages). Let's notice that while we use Radius between the Authorizer and the AS, we use EAPOL between the Supplicant and the Authorizer.
- CCMP: in contrast to TKIP, in this activity we have used AES-CCMP for encrypting our data packets. It uses AES, which is a very good cipher.
- WPA2/RSN: It is the standard that contain all the new security measures. Designed for replacing the old and weak WEP and the temporal WPA. It also contains TKIP as an optional mode but its main purpose is to define AES-CCMP and stronger access control (like the 4-way handshake).

Q19. We think that taking into consideration the time available for this laboratory it has covered all our expectations and hence we have no suggestions about how to improve it. In our opinion it has been a really good experience for understanding the differences, advantages and disadvantages of using one or another method for protecting a WLAN.

References

[1] *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*, by Jon Edney and William A. Arbaugh, Addison-Wesley Professional

[2] *Attacks on the RC4 stream cipher*, Andreas Klein, 16 April 2008

[3] Aircrack Tutorial, `http://aircrack-ng.org`

[4] *Breaking 104 Bit WEP in Less Than 60 Seconds* by Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin

[5] Wi-Fi Alliance. Wi-Fi Protected Acccess (WPA), `http://www.wi-fi.org/discover-wi-fi/security`

[6] OpenWall. English wordlists. `ftp://ftp.openwall.com/pub/wordlists/languages/English/`

[7] OpenWall. John the Ripper syntax and rules. `http://www.openwall.com/john/doc/RULES.shtml`

[8] Wikipedia. WPA article. `http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access`

[9] NTNU. The Norwegian University of Science and Technology, buy new supercomputer. `http://www.ntnu.edu/news/new-supercomputer-to-be-installed`.

[10] Wikipedia. Password Cracking. http://en.wikipedia.org/wiki/Password_cracking

[11] Wikipedia. FLOPS. http://en.wikipedia.org/wiki/FLOPS

[12] Marcus Redivo. Creating and using SSL certificates. http://www.eclectica.ca/howto/ssl-cert-howto.php

Appendix A: hostapd.conf file used in Challenge 2

```
interface=wlan0
bridge=br0 (the bridge previously created)
driver=nl80211
ssid=Try to hack me!!
wpa=1 (WPA enabled)
wpa_passphrase=Elephant1
wpa_key_mgmt=WPA-PSK
```

Appendix B: openssl.cnf file used in Challenge 3

```
0.organizationName_default = NTNU
organizationalUnitName_default = ITEM
localityName_default = Trondheim
stateOrProvinceName_default = Sor-Trondelag
countryName_default = NO
```

Note: these lines provide information to the certificate, thus it depends of the owner of the certificate. In this case we have used these data because all CA, AS and supplicant belong to the same institution.

Appendix C: Freeradius configuration files (Challenge 3)

eap.conf In this file we make sure that,

- `default_eap_type = tls`
- `private_key_file =` path to the AS private key
- `private_key_password =` password chosen in step 2 for protecting the AS private key
- `certificate_file =` path to the AS's signed certificate
- `CA_file =` path to the CA's signed certificate
- `dh_file =` path to a file containing parameters for a Diffie-Hellman key exchange
- `random_file =` path to the file with random data

In this case we have used `/dev/urandom` as a random file and created the file with Diffie-Hellmann parameters by the instruction `openssl dhparam -out dh 1024`. Why do we need this file? Because Diffie-Hellman is going to be used for exchanging the cryptographic keys.

clients.conf In this file we must specify the client/s that are going to speak with the Radius server. In this case we only have one authorizer and actually it is placed in the same computer than the Radius server so the only client will be the localhost and thereby we must be sure that the `client localhost` paragraph is the only one uncommented. Also inside that paragraph we must make sure that `ipaddr = 127.0.0.1` and that `secret = shared secret with the`

`authorizer`. This secret will be the key that is going to be used for encrypting the packets between the AS and the authorizer (AP) and hence the authorizer will also have to know it (but be sure that nobody else knows it because the security of the Radius protocol depends completely on this secret and it is necessary to prevent the installation of wild authorizers).

Appendix D: hostapd.conf file used in Challenge 3

- `interface=wlan0`
- `bridge=br0` (we reuse the bridge set up in the Challenge 2)
- `driver=nl80211`
- `ssid=EMA_AP` (the name of our new AP, EMA is the acronym of the first letters of our names)
- `channel=6` (we have selected the channel where our AP will operate, usually we should select the less crowded channel in order to avoid interferences)
- `ieee8021x=1` (802.1X enabled)
- `wpa=2` (WPA2 enabled)
- `wpa_key_mgmt=WPA-EAP` (key management algorithm)
- `wpa_pairwise=CCMP` (Pairwise cipher)
- `auth_server_addr=127.0.0.1` (RADIUS authenticattion server address)
- `auth_server_shared_secret=SharedSecret99` (RADIUS secret)
- `eap_server=0` ("integrated EAP server" disabled)

Appendix E: wpa_supplicant configuration file

```
ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="EMA_AP"
    key_mgmt=WPA-EAP
    proto=WPA2
    pairwise=CCMP
    group=CCMP
    eap=TLS
    ca_cert="./cacert.pem"
    client_cert="./supplicantcert.pem"
    private_key="./supplicantkey.pem"
    private_key_passwd="SharedSecret99"
    identity="EMA_AP"
}
```

Appendix F: Pseudo-dialog of AS authenticating the Supplicant

-Hello supplicant, how do I know that you are the owner of the certificate (public key) that you are sending me?
-Because I answered you correctly the message that you sent me encrypted with my public key. I am the only one that could unencrypt it because I am the only one that have the corresponding private key.
-Ok, now I now know that you are really you, but how do I know that you are

allowed to access the network?
-Because the CA put his signature on my certificate.
-Ok, but how do I know that it is the signature of the CA and it is not you who signed it?
-Because you also have the CA's certificate and hence you can confirm that this signature corresponds to a legitimate CA signature of my certificate.
-Ok, but how do I know that the CA certificate that I have is legitimate?
-Because you put it there personally (probably the CA in person gave it to you manually).
-Ok supplicant, you can get it, I will let it know to the authorizer (AP)

Note: With the analogous process the supplicant also authenticates the AS.

Appendix G: Calculations for Question 14

The WPA passphrase should be from 8 to 63 ASCII characters long. In WPA, 95 different ASCII characters are allowed (from 32 to 126, decimal index), so our passphrase could take one of the 95^x different combinations, where x is the length of the passphrase (from 8 to 63).

Now, let's consider the new supercomputer at NTNU (see [9]), which has a capacity of 275 TFLOPS ($275 * 10^{12}$ Floating-point Operations Per Second). With this computer, if we assume that we require 10 Floating-point Operations per checked key (see [10]), we can check $\frac{275*10^{12}}{10} = 275*10^{11}$ keys per second. After 30 days (2592000 seconds) checking keys ceaselessly, we will have checked $2592000 * 275 * 10^{11} = 7,128 * 10^{19}$ keys.

If the probability of succeeding goes from 0 to 1 and can be expressed as $p = \frac{keyschecked}{totalofpossiblekeys}$, and this must be such that $p \leq 2^{-20}$, we can obtain x (the minimum length of the passphrase) in the following way:

$$p = \frac{keyschecked}{totalofpossiblekeys} = \frac{7,128 * 10^{19}}{95^x} \leq 2^{-20} \longrightarrow \frac{7,128 * 10^{19}}{2^{-20}} \leq 95^x$$

then,

$$\log\left(\frac{7,128 * 10^{19}}{2^{-20}}\right) \leq \log(95^x) \longrightarrow \log\left(\frac{7,128 * 10^{19}}{2^{-20}}\right) \leq x * \log(95)$$

and now,

$$x \geq \frac{\log\left(\frac{7,128*10^{19}}{2^{-20}}\right)}{\log(95)} = 13,08$$

which means that at least we need our passphrase to be 14 characters long (because we have to round up the result to the next integer).

If instead of using the new NTNU supercomputer we would have used a regular desktop PC (which usually performs around 7 GFLOPS (see [11]), for the same working time (30 days) and for the same probability (2^{-20}), the minimum required length for the passphrase would be of 11 characters long (it makes sense because the computer would have checked less keys and hence we wouldn't need that much key robustness).

Another way of performing this calculations (instead of assuming that we require 10 Floating-point Operations per checked key) would have been the following: during the lab, the computer

used to perform the brute-force attack, showed in the shell (while it was checking the keys) the keys per second (k/s) ratio. Hence, knowing that such a computer performs around 7 GFLOPS, we could have extrapolated the k/s ratio of the new NTNU supercomputer as it follows,

$$NTNUsupercomputer\frac{keys}{sec} = (275 * 10^{12}) * (DesktopPC\frac{keys}{sec}) * \frac{1}{7 * 10^9}$$

and now, knowing the NTNU supercomputer k/s ratio, repeat the last calculations of the previous method.