NTNU

Laboratory Report

WLAN Security Analysis and Construction

TM4137 Wireless Security

Group 3
Rikard Eide, Eyvind Gjertsen, Steffen Birkeland

September 27, 2015

Preface

This report is based on the laboratory assignment for the TTM4137 course. As the assignment itself, it is split into three main parts. Each part contains a short introduction, the lab set-up and the attack/configuration procedure(s). The last section of each part contains answers to the questions from the lab description. As a part of these answers, we include our own reflection on the technologies we tested, our results, as well as some more detail on the attacks/protocols we used. Please refer to our glossary in Appendix A for any unclear abbreviations.

1 WEP Penetration and Cryptanalysis

In this section we describe how to crack a WEP password and successfully decrypt all traffic on the network. WEP is one of the first encryption schemes developed for wireless communication. After being both theoretically and practically broken in a number of publications and software implementations, it is strongly advised to refrain from using WEP to secure wireless networks.

1.1 Lab Set-up

The lab was already set up with an AP running WEP, and our task was to find the secret key. All we needed to perform the attack was a normal PC with a wireless network card (NIC), and some free software (`kismet`, `aircrack-ng` suite).

1.2 Attack Procedure

Hidden SSID: The AP was configured to hide its SSID (called "network cloaking"), meaning that it did not appear in the public list of networks. However, even though the SSID is not broadcast publicly (in the beacons), it is still possible to acquire it by looking at other packets transmitted across the network (probes, associations), which always contain this information. Free software like `kismet` accomplishes this task in just a few seconds (see figure 1).

Monitor mode: In order to be able to capture traffic on any channel, we had to set our NIC to "monitor mode". This was done with the tool `airmon-ng`, by running one simple command: `airmon-ng start wlan0`.

Capture traffic: As our NIC now was receiving traffic from all APs and stations within range, we ran `airodump-ng` to capture the traffic we needed for the attack. Using the information obtained with `kismet`, we ran the following command: `airodump-ng -c 6 -w dump --bssid 98:FC:11:B2:D2:67 mon0`, initiating capture on channel 6, focusing on traffic to/from the AP we wanted, expressed by the BSSID.

Packet injection: Since only one legitimate client was connected to the AP, the traffic rate was rather moderate. In order to speed things up, we used `aireplay-ng` to inject packets into the network, more specifically ARP packets. *ARP Replay* will sniff a legitimate ARP-request packet off the network and replay this, spoofing the legitimate clients MAC address, thus causing the AP to generate traffic (ARP-responses). To do this, we executed the following command: `aireplay-ng -3 -b 98:FC:11:B2:D2:67 mon0`, where `-3` means *ARP replay* and `-b` denotes the BSSID of the access point. Still running `airodump-ng`, we were capturing the ARP-responses, building our IV library.

Cracking the key: After collecting a sufficient amount of IVs [3], we could leverage our attack, using `aircrack-ng`. We executed the following command: `aircrack-ng dump-01.cap`, where `dump-01.cap` is our captured traffic. Almost immediately, the key was found (see figure 2).

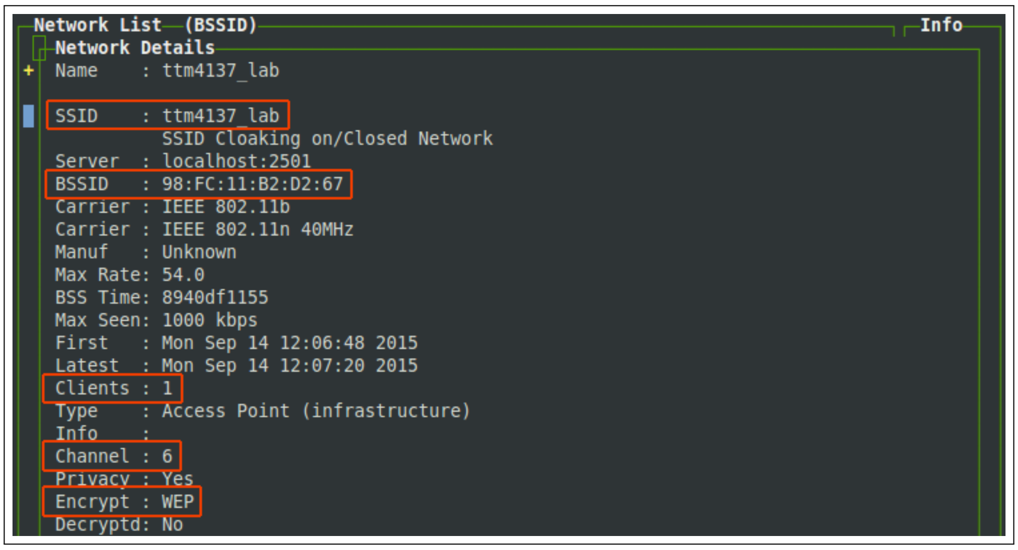
Decrypting the traffic: Upon retrieving the secret key, we had everything needed to generate keystreams (as the packet IV is sent in the clear) and decrypt any packet on the network. In Wireshark it is possible to enter the key in order to read encrypted packet data in clear text.

1.3 Results

By using freely available software, we were able to crack the secret key in no-time, and with almost no processing power. It is therefore safe to say that WEP is a very insecure encryption scheme. Please refer to the "questions" section (1.4) below for more detailed results.

1.4 Questions

Q1. *Describe the parameters of the AP under analysis. How many packets did the PTW attack require? Which web page was the legitimate client browsing?* AP paramters (SSID, BSSID, # of associated clients, channel and encryption mechanism) found with `kismet` are displayed in figure 1 below. The PTW attack requires on average 40,000 packets [3]. We forgot to stop our ARP-request injection, so we ran our attack with **81,129** packets, which successfully uncovered the key, as shown in figure 2. The client was browsing *www.vg.no*, which we were able to see from the decrypted HTTP GET packet.



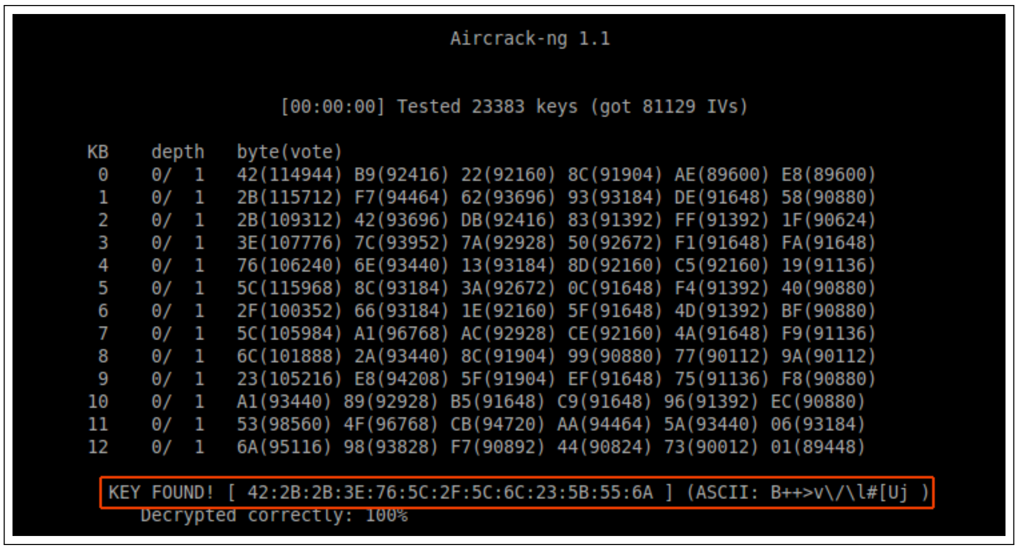
```
Network List  (BSSID)                                              Info
 Network Details
 Name     : ttm4137_lab

 SSID     : ttm4137_lab
            SSID Cloaking on/Closed Network
 Server   : localhost:2501
 BSSID    : 98:FC:11:B2:D2:67
 Carrier  : IEEE 802.11b
 Carrier  : IEEE 802.11n 40MHz
 Manuf    : Unknown
 Max Rate: 54.0
 BSS Time: 8940df1155
 Max Seen: 1000 kbps
 First    : Mon Sep 14 12:06:48 2015
 Latest   : Mon Sep 14 12:07:20 2015
 Clients : 1
 Type     : Access Point (infrastructure)
 Info     :
 Channel : 6
 Privacy : Yes
 Encrypt : WEP
 Decryptd: No
```

Figure 1: Access Point details found with `kismet`

Q2. *Is it possible to run a completely passive PTW attack on 104-bit WEP? Why and under which circumstances?* Yes, it is possible to run a passive attack. This requires that there are some client(s) associated with the AP, such that there is some traffic to be captured. It may however take a long time until enough ARP-packets are transmitted across the network. In [3], it is described how IPv4 packets can be used.

Q3. *Why is it possible to send an arbitrary amount of ARP-requests to the AP without knowing the WEP key?* ARP is a link-layer protocol, and (MAC) addresses are public and can easily be spoofed. Furthermore, WEP does not have a mechanism to detect replay of packets. By capturing one legitimate ARP-request (identified by its characteristic structure), one can replay this multiple times (using e.g. `aircrack-ng/aireplay-ng`). The AP will accept this packet because the IV/keystream pair is still valid.



```
                                 Aircrack-ng 1.1


                    [00:00:00] Tested 23383 keys (got 81129 IVs)

   KB    depth   byte(vote)
    0    0/  1   42(114944) B9(92416) 22(92160) 8C(91904) AE(89600) E8(89600)
    1    0/  1   2B(115712) F7(94464) 62(93696) 93(93184) DE(91648) 58(90880)
    2    0/  1   2B(109312) 42(93696) DB(92416) 83(91392) FF(91392) 1F(90624)
    3    0/  1   3E(107776) 7C(93952) 7A(92928) 50(92672) F1(91648) FA(91648)
    4    0/  1   76(106240) 6E(93440) 13(93184) 8D(92160) C5(92160) 19(91136)
    5    0/  1   5C(115968) 8C(93184) 3A(92672) 0C(91648) F4(91392) 40(90880)
    6    0/  1   2F(100352) 66(93184) 1E(92160) 5F(91648) 4D(91392) BF(90880)
    7    0/  1   5C(105984) A1(96768) AC(92928) CE(92160) 4A(91648) F9(91136)
    8    0/  1   6C(101888) 2A(93440) 8C(91904) 99(90880) 77(90112) 9A(90112)
    9    0/  1   23(105216) E8(94208) 5F(91904) EF(91648) 75(91136) F8(90880)
   10    0/  1   A1(93440) 89(92928) B5(91648) C9(91648) 96(91392) EC(90880)
   11    0/  1   53(98560) 4F(96768) CB(94720) AA(94464) 5A(93440) 06(93184)
   12    0/  1   6A(95116) 98(93828) F7(90892) 44(90824) 73(90012) 01(89448)

     KEY FOUND! [ 42:2B:2B:3E:76:5C:2F:5C:6C:23:5B:55:6A ] (ASCII: B++>v\/\l#[Uj )
        Decrypted correctly: 100%
```

Figure 2: Discovering the key with `aircrack-ng`

Q4. *What can you do to strengthen your attack when there are some clients, but hardly any traffic at all?* As stated in the previous question, one can capture one ARP-request and replay this. Alternatively, is is possible to trick a client to believe that they have been de-authenticated (by sending a *State1*[1] packet), making them re-authenticate with the AP, and thus generate traffic.

Q5. *How can you obtain packets for injection if no clients are associated with the AP?* If no clients are associated with the AP, one can use the following procedure to conduct the attack [4]: 1. Fake-authenticate with the AP (using `aireplay-ng`). 2. Use KoreK's "ChopChop attack" or the "Fragmentation attack" to decrypt a packet in order to obtain the keystream[2] (ciphertext XOR plaintext). 3. Craft an ARP-request (e.g. using `packetforge-ng`) and encrypt this with the keystream and IV from the previous step. 4. Inject this multiple times into the network and capture the responses from the AP. 5. Run `aircrack-ng` when enough packets (IVs) are collected.

Q6. *Which weaknesses of WEP are we taking advantage of in our attack, and why?* `aircrack-ng` runs the PTW attack by default [5]. This attack extends Klein's attack on RC4, using correlations between the secret key and the generated keystreams to statistically determine the secret key. In addition to this, we take take advantage of WEP's inability to detect packet replays, such that we are able to replay packets at high rates in order to collect large amounts of IVs.

Q7. *Under what circumstances would you consider WEP to be sufficiently secure?* WEP is generally not secure under any circumstances. Only by applying other encryption mechanisms at higher levels (e.g. VPN/IPSec or HTTPS/SSL), we can be confident that our data is not being disclosed to outsiders. Maybe by deploying an advanced IDS "in-front" of the AP, we can reduce the attack surface (e.g. by detecting ARP injections), but due to the many flaws in WEP, it can never provide sufficient security.

Q8. *Would these attacks be effective against a network deploying WPA with TKIP? If the RC4 cipher used in WEP is considered to be insecure, why is it reused in WPA?* WPA-TKIP is not vulnerable to the same attacks we leveraged against WEP. For one, the *Root/Master key* is not used to encrypt traffic. A new key is generated for each packet. Also, the IV is used as a packet

[1]If a station is prepared to receive data packets (*State3*), but receives a control packet, it will re-authenticate.
[2]Even though no clients are associated, the AP will still transmit some encrypted packets [4].

counter, prohibiting replay of packets. The reason RC4 is still used in WPA is that it was only developed as a so-called "stopgap" protocol in order to patch the most critical vulnerabilities of WEP [2], as the hardware in the existing APs did not support any other encryption algorithms.

2 Password Dictionary Attack

The second part of the lab consisted in running a brute-force dictionary attack against a WLAN running WPA-PSK, in order to determine the secret password. This password is the basis for all confidentiality in the WPA-PSK protocol, thus discovering this could be considered a major security breach. In order to check if we are guessing correctly, the password and other network specific parameters must be fed into two different key derivation functions (parameters listed in *Q10* below). Two of these parameters can only be obtained during the authentication procedure (4-way handshake) between the supplicant and AP.

2.1 Lab Set-up

We used the following set-up for this part:

Lab Computer 1 Using `hostapd`, we set up a WLAN AP, running WPA-PSK, on the wireless NIC. We also we set up a bridge between the NIC and the Ethernet port for Internet access. SSID: *TheMatrix*, password: *Garden01*.

Lab Computer 2 Running packet capture with `airodump-ng` to catch the 4-way handshake, and then execute the brute-force dictionary attack.

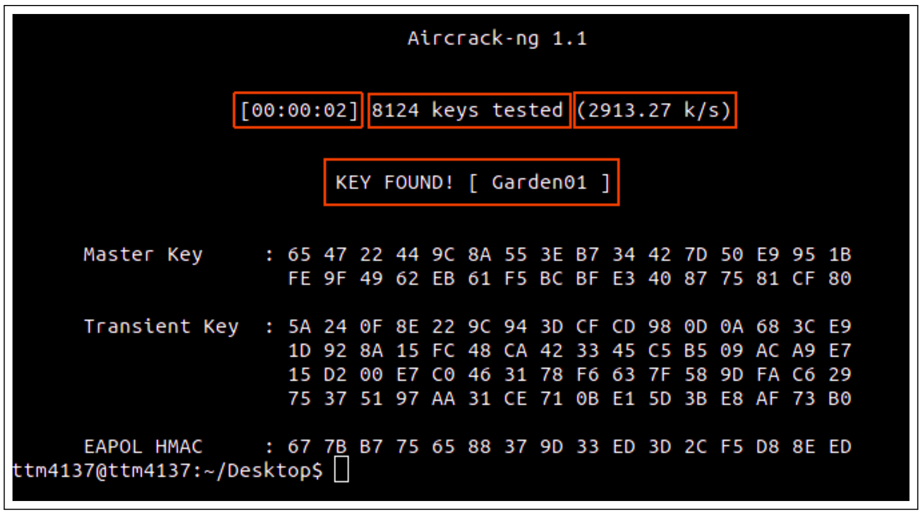
See Appendix B for all the changes we made in the `hostapd.conf` file.

2.2 Attack Procedure

Password dictionary: To set up our dictionary for the attack, we first needed a list of words that we could extend to match typical password patterns. We chose a word list with approximately 28 000 English words [7]. To create different variations of each word we used the tool `John the Ripper`. We modified the `john.conf` file to adapt to the rules to our case [6]. The following command takes in a list `lower.lst` that contains the words we want to use as a basis for our dictionary. `-stdout -rules` applies the rules we defined in john.conf to all the words in lower.lst and outputs it to `extendedwords.lst`: `john --wordlist=lower.lst --stdout --rules > extendedwords.lst`

De-authentication: We then connected a smartphone to our network. We started `airodump-ng` to capture the traffic on our network. We used `aireplay-ng` to de-authenticate the connected device so we could capture the 4-way handshake once it reconnected. This was necessary in order to get the `ANonce` and `SNonce`. It is important to note that this step is not required, as you may just simply wait until a client connects to the network to capture these parameters, making this attack completely passive.

Cracking the password: The final step was to run `aircrack-ng` with our captured file, password dictionary and the SSID. Thus, the only unknown value was the password. We ran `aircrack-ng` with the following parameters: `-a wpa`, meaning that we are attacking WPA, `-b` denoting the `BSSID`, `-w` for the word list we used, and fileCapture.cap was our traffic dump from `airodump-ng` [5]: `aircrack-ng -a wpa -b TheMatrix -w extendedwords.lst fileCapture.cap`



```
                          Aircrack-ng 1.1

          [00:00:02] 8124 keys tested (2913.27 k/s)

                      KEY FOUND! [ Garden01 ]

   Master Key     : 65 47 22 44 9C 8A 55 3E B7 34 42 7D 50 E9 95 1B
                    FE 9F 49 62 EB 61 F5 BC BF E3 40 87 75 81 CF 80

   Transient Key  : 5A 24 0F 8E 22 9C 94 3D CF CD 98 0D 0A 68 3C E9
                    1D 92 8A 15 FC 48 CA 42 33 45 C5 B5 09 AC A9 E7
                    15 D2 00 E7 C0 46 31 78 F6 63 7F 58 9D FA C6 29
                    75 37 51 97 AA 31 CE 71 0B E1 5D 3B E8 AF 73 B0

   EAPOL HMAC     : 67 7B B7 75 65 88 37 9D 33 ED 3D 2C F5 D8 8E ED
ttm4137@ttm4137:~/Desktop$
```

Figure 3: Result from running the dictionary attack with `aircrack-ng`

2.3 Results

We experienced the big drawback with brute-force attacks. After 30 minutes of wrong guesses, we decided to remove some rules from our `john.conf` file, in order to make it feasible to complete the lab within our lab hours, and confirm that our setup worked. In the second run, we were "lucky" to find the key after **8,124** tested keys. As shown in figure 3, the lab computer was able to test approx. 3,000 keys per second, and so the key was found within **3 seconds**. Considering that we had drastically reduced the password space for the second run, we acknowledge the fact that brute-force attacks may be very time-consuming.

2.4 Questions

Q9. *Why is TKIP a more secure encryption mechanism than the one-key WEP alternative?* TKIP was introduced as a intermediate solution in the mitigation process from WEP to what is now known as WPA2. WEP's major drawback is the fact that all devices on the same network use the same key to encrypt their data. WPA introduced improved key management through TKIP. It scrambles the keys with a hashing algorithm and adds a MIC. With TKIP, each wireless client have unique encryption keys, which are constantly changed[16].

Q10. *Which information is used to compute the Pairwise Master Key (PMK) and the Pairwise Temporary Key (PTK) in the password-based PSK scenario? Why is the offline password brute-force attack possible?* The PMK is calculated as follows:

$$PMK = PSK = PBKDF2(\,password, SSID, SSID_{length}, Hash_{count}, Output_{length}\,)$$

In this example the pairwise master key is equal to the pairwise session key which is calculated from the password and the SSID of the wireless network. The last two parameters determine the number of hashes/rounds and the length of the output. The PTK is calculated as:

$$PTK = f(PMK, MAC_{AT}, MAC_{STA}, Nonce_A, Nonce_S)$$

The offline brute force attack is possible because the initial 4-way handshake is done in plain text. This makes it possible for an attacker to pick up MAC addresses and nonces. These values are all that is needed in order to start guessing the PSK offline.

Q11. *Does the compromise of password imply the disclosure of traffic from previous sessions? Justify your answer.* Under the circumstances that an attacker have been capturing all traffic since the initial handshake (which can be provoked by a deauthentication attack), he or she will have all the required information in order to calculate the PTK, which is used to encrypt the traffic within the current session. However, if the captured data spans multiple sessions, they may have to re-calculate the PTK for each session (with unique nonces for each session).

Q12. *Does the use of AES in WPA2-PSK give an advantage against a password dictionary attack, as compared to RC4 in WPA-PSK?* No, the encryption algorithm does not matter because the dictionary attack only targets the 4-way handshake, where encryption is not yet applied.

Q13. *What would be advantages and disadvantages of using a pre-computed database of PMK in a PSK password dictionary attack?* The PMK is computed as PMK = PSK. The PSK is a key derived (through PBKDF2) from the password and the SSID of the network. Because SSIDs usually are different, a so called "rainbow table" of pre-computed hashes wouldn not give any advantage, as you would have to pre-compute these tables for every SSID. What you could do however, is to pre-compute these hashes for common SSIDs, but that would require a tremendous amount of time/resources, which might turn out to be a waste of time.

Q14. *Suppose an attacker employs 30 days of processing time on the new supercomputer at NTNU. Which minimum requirements would you put on a WPA password to ensure that the probability of successful attack does not exceed 2e-20?* In our lab we put pretty hard constraints on our password, restricting it to only consist of upper- and lowercase letters and numbers. This is makes up 62 unique ASCII characters, and for this calculation we wanted to find out how long our key had to be in order to be within the an acceptably low probability of being compromised. We concluded that our password had to be **at least 15 characters long**. A justification and the full calculations can be found in Appendix C.

3 Setting up RSN-EAP Wireless Access Point

This challenge was not about *attacking* an insecure network, but rather *defending* by setting up a secure network. The main objective was to set up a wireless access point and configure it's security features following industry standards. Part 2 of the lab demonstrated the weaknesses with the handshake in WPA-PSK. In order to mitigate this and establish mutual authentication of both the supplicant and the authentication server we employed CCMP, the EAP-TLS authentication protocol and digital certificates based on public key encryption. The following section will address the the necessary steps in order to achieve this, before discussing how and why this is a secure setup.

3.1 Lab Set-up

For challenge 3 the we used three computers which were allocated the following responsibilities:

Lab Computer 1 Certificate Authority (CA), responsible for generating and signing certificates for all entities.

Lab Computer 2 Authentication Server (AS) running FreeRadius and serving as an AP and Authenticator configured with `hostapd`.

Personal Computer Supplicant with a personal certificate. Sole responsibility was to connect to the secure access point.

3.2 Configuration Procedure

Setting up the CA: Starting with the certificate authority, we used the OpenSSL software in order to create a private key and certificate for the CA. This certificate was self-signed, meaning this CA now would be a *Root Certificate Authority* [13] for everyone it should issue certificates to. We downloaded an `openssl.conf` template file [9] and changed default values to something suitable for our little CA organisation, as well as the key length (2048 bit) and message digest algorithm (sha256) to values that are considered secure at the moment [10].

Creating and self-signing a certificate can all be done with the following one-liner: `openssl req -new x509 -keyout cakey.pem -out cacert.pem -config ./openssl.cnf`. The `req -new -x509` flags tell OpenSSL that we want to create a X.509 Certificate Signing Request (CSR) and sign it. The output is two `.pem` files, which contain the self-signed (root) certificate (see Appendix D) and the CA's private key. This key is protected by a password, created at the same time. We then went on and created private keys and certificates for the supplicant and AS: `openssl req -new -out supplicantreq.pem -keyout supplicantkey.pem -config ./openssl.cnf` (repeated for the AS with new file names).

Note: The lab notes instructed us to create all certificate related files (including public/private key pairs) on one machine and then transfer it to the other actors later. No one should *ever* generate nor handle your private key. It is important to note that outside the lab, both keys and certificates should be generated locally, and then the certificate can be sent to the CA for signing.

The two certificates were then signed by the CA, once again with the OpenSSL terminal program: `openssl ca -out supplicantcert.pem -config ./openssl.cnf -infiles supplicantreq.pem` (and again the same for the AS). We then transferred the files to the respective computers with a USB stick.

The authentication server: In order to set up the AS, we used the FreeRadius program. The program has a lot of default settings, and we modified two files, `eap.conf` and `clients.conf`, located in `/etc/freeradius`. In the `eap.conf` file (Appendix E) we specified the absolute path to where our certificates and keys were located. Another input for the AS is a file containing parameters for the Diffie-Hellman key exchange, along with a file containing random bytes. These two files can be created with the following commands: `openssl dhparam -out dh 1024` and `openssl dhparam-out random 256`. We also made sure that the default EAP method was set to TLS.

In the `clients.conf` file we had to specify who was going to communicate with the AS. In our case the AP and AS were located one same machine. Therefore, we made sure `localhost` was the only client that was "uncommented". We also set a shared secret between the AS and the AP. We then started the server with the `freeradius -X` command.

Note: We encountered some permission problems when first firing up the FreeRadius server. The server is configured to run with it's own user, which in our case did not have read access to the certificate files. We solved this by changing ownership of all the certificate files `chown -R freerad:freerad ../CA/`.

Access point: The AP would act as a mediator between the supplicant and the AS, as well as handling encryption CCMP for the wireless communication. These configurations were done in the `hostapd.conf` file (see Appendix F).

Connecting to the network: The easiest way of connecting to the network was to use our operating system's built in network connection manager. After finding the wireless network in the list of available networks, we had to locate the certificates we received from the CA via USB and specify both the CA certificate and our own along with our private key with its password. Any value is accepted in the the identity field, so after this we were able to connect to the RSN with link layer encryption (see Appendix G).

3.3 Results

We successfully set up an RSN with AES-CCMP link encryption and mutual authentication.

3.4 Questions

Q15. *When would it be appropriate to use WPA2-PSK instead of WPA2-EAP (as a key management scheme)?* It would be better to use in small environments where there are not too many users, where it is easier to keep an overview over the people on the network and thus individual authentication can seem unnecessary. The issue with using a PSK is that everyone use the same key, and that increases the risk of the key being compromised. This is why it should be used with few users to limit the chances of this happening, and make it easy to change if it would happen.

Q16. *EAP-TLS deploys mutual authentication of two communicating parties. What kind of attack is possible against authentication protocols lacking such authentication?* MITM attack could be possible if there is no mutual authentication. The typical scenario would be that the user authenticates to the server, but the server does not authenticate to the user (as in e.g. GSM). This could allow for an attacker to set up a fake AP and possibly disclose sensitive information, hijack the traffic, or tamper with the packets. Considering if the client is not authenticated, an attacker could impersonate a legitimate client act maliciously on their behalf.

Q17. *Give an overview of the EAP authentication protocols which can be used in WPA2-Enterprise WLANs.* The following protocols can be used [8]:

EAP-TLS The original standard in wireless security.

EAP-TTLS/MSCHAPv2v2 Creates a secure tunnel before starting authentication.

PEAPv0/EAP-MSCHAPv2 Also uses a secure tunnel to protect the authentication.

PEAPv1/EAP-GTC Uses a text challenge and securtiy token for authentication.

EAP-SIM A SIM-card is used for authentication.

EAP-AKA Used in UMTS.

EAP-FAST Uses PAC to create a secure TLS tunnel where a client can be verified.

Q18. *List the security-related protocols used in your RSN-EAP-TLS setup and explain their purpose.* **EAP** - Provide functions and methods for authentication protocol transport, without IP connectivity. **TLS** - Used to negotiate a symmetric key for transport between the communicating endpoints. It ensures confidentiality and integrity. **RADIUS** - Is used for the communication between the Authenticator (AP) and AS. It runs on the application layer. It is used to grant or deny access to the network. **CCMP** - Is used to encrypt the data sent over the wireless network. It guarantees authenticity, integrity and confidentiality. It uses AES with counter mode cipher block chaining. It provides data confidentiality, authentication. **RSN/WPA2** - The latest security standard in wireless security. Its purpose is to secure wireless computer networks. It is the replacement for WPA that used temporal keys. WPA2 has mandatory support for CCMP mentioned above. What separates it from WPA is the encryption and key management.

Q19. *How can this lab project be improved?* We were satisfied with how the lab was organized, and the lab description was very helpful. There was only one issue with the configuration in part 3, where some parameters for `eap.conf` and `client.conf` were left out, and this did cause some confusion to why the setup did not work at first. Also we would have preferred if the answers to the questions could be ”merged” into the report, instead of being a separate part. In some situations, we had to choose between repeating an answer to a question, or leaving it out where it seemed natural to mention it.

References

[1] Jon Edney and William A. Arbaugh. *Real 802.11 security: Wi-Fi protected access and 802.11i*, Boston, MA: Addison-Wesley 2004 ISBN: 0-321-13620-9.

[2] Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc. ”Wired Equivalent Privacy”, *en.wikipedia.org/wiki/Wired_Equivalent_Privacy.* Visited 21 September 2015.

[3] Erik Tews, Ralf-Philipp Weinmann and Andrei Pyshkin. *Breaking 104 Bit WEP in Less Than 60 Seconds*, TU-Darmstadt, FB Informatik.

[4] Aircrack-ng.org. ”Tutorial: How to crack WEP with no wireless clients”, *www.aircrack-ng.org/doku.php?id=how_to_crack_wep_with_no_clients*, Version 1.16 (darkAudax). Visited 22 September 2015.

[5] Aircrack-ng.org. ”Aircrack-ng Description”, *www.aircrack-ng.org/doku.php?id=aircrack-ng.* Visited 22 September 2015.

[6] OpenWall. ”Rules for john the ripper config file” *www.openwall.com/john/doc/RULES.shtml.* Visited 24 September 2015.

[7] OpenWall. ”List of English words in lower-case (lower.gz)”, *ftp.openwall.com/pub/wordlists/languages/English/1-tiny/.* Visited 19 September 2015.

[8] Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc. ”Wi-Fi Protected Access”, *en.wikipedia.org/wiki/Wi-Fi_Protected_Access.* Visited 21 September 2015.

[9] Eclectica. ”Creating and Using SSL Certificates”, *www.eclectica.ca/howto/ssl-cert-howto.php.* Visited 23 September 2015.

[10] Ivan Ristić. OpenSSL Cookbook / SSL/TLS Deployment Best Practices, Feisty Duck, Qualys. Available at *https://www.feistyduck.com/books/openssl-cookbook/.* Visited 26 September 2015.

[11] Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc. ”ASCII Printable Characters”, *en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.* Visited 23 September 2015.

[12] NTNU HPC GROUP / Vilje. ”About Vilje”, *www.hpc.ntnu.no/display/hpc/About+Vilje.* Visited 23 September 2015.

[13] Microsoft Technet. ”Certification Authority Trust Model”, *technet.microsoft.com/en-us/library/cc962065.aspx.* Visited 23 September 2015.

[14] NTNU. ”The Norwegian University of Science and Technology, met.no buy new supercomputer”, *www.ntnu.edu/news/new-supercomputer-to-be-installed.* Visited 23 September 2015.

[15] Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc. ”Floating Point Operations Per Second”, *https://en.wikipedia.org/wiki/FLOPS.* Visited 26 September 2015.

[16] Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc. ”IEEE 802.11i-2004”, *https://en.wikipedia.org/wiki/IEEE_802.11i-2004.* Visited 27 September 2015.

Appendix A

Glossary

AKA Authentication and Key Agreement. 9

AP Access Point. 2

ARP Address Resolution Protocol. 2

AS Authentication Server. 7

BSSID Basic Service Set Identifier. 3

CA Certificate Authority. 7

CCMP Counter Mode Cipher Block Chaining Message Authentication Code Protocol. 9

EAP Extensible Authentication Protocol. 9

FAST Flexible Authentication via Secure Tunneling. 9

GTC Generic Token Card. 9

IDS Intrusion Detection System. 4

MAC Medium Access Control. 3

MIC Message Integrity Code. 6

MITM Man-in-the-Middle. 9

MSCHAPv2 Microsoft Challenge Authentication Protocol. 9

NIC Network Interface Card. 2

PAC Protected Access Credential. 9

PBKDF2 Password-Based Key Derivation Function. 6

PSK Pre-shared Key. 5

PTK Pairwise Temporary Key. 6

PTW Pyshkin, Tews, Weinmann. 3

RADIUS Remote Authentication Dial-In User Service. 9

RSN Robust Security Network. 9

SIM Subscriber Identity Module. 9

SSID Service Set Identifier. 3

TKIP Temporal Key Integrity Protocol. 4

TLS Transport Layer Security. 9

TTLS Tunneled Transport Layer Security. 9

UMTS Universal Mobile Telecommunications System. 9

WEP Wired Equivalent Privacy. 2

WPA Wi-Fi Protected Access. 5

Appendix B

Changes made to `hostapd.conf` file to set up our WPA-PSK WLAN AP in part 2 of the assignment.

`ssid = TheMatrix`	Setting the SSID
`wpa = 1`	Enabling WPA
`wpa_passphrase = Garden01`	Setting the secret key (to be cracked)

Appendix C

In order to calculate the probability of a successful attack, we first have to look at NTNU's supercomputer and it's capabilities. The article referenced [12] in the Lab Description says Vilje will be able to do 275 trillion floating point operations per second. Updated numbers [14] reveal that the hardware has been upgraded since 2011, and Vilje should now be capable of 397 Teraflops/s. There are multiple approaches as to how to determine the number of keys that can be tested per second given a number of FLOPS. Our lab machine peaked at 3000 guesses per second, and they run at 2.5Ghz processors. However, it is reasonable to assume that Vilje is optimised for operations like this both in hardware and in OS/software, so for the sake of the calculation we assume that Vilje would require 10 floating-point operations to check one key (see [15]). This lets us test

$$\frac{397\,e^{12}}{10}\,flops\;\times\;2592000\,seconds = 1.02\,e^{20}$$

number of keys in 30 days. The WPA passphrase is limited to 63 printable ASCII characters [11], and has to be at least 8 characters long. Further, the IEEE Standard (802.11i-2004, Annex H.4.1) specifies the characters to have an encoding in the decimal range of 32 to 126. Given a uniform random distribution among all of the characters, this gives us 95 different symbols to work with on up to 63 characters. In our lab we put pretty hard constraints on our password, restricting it to only consist of upper- and lowercase letters and numbers. This is makes up 62 unique ASCII characters, and for this calculation we want to find out how long our key has to be in order to be within the an acceptably low probability of being compromised:

$$p = \frac{keyschecked}{num\,of\,possible\,keys} = \frac{1.02\,e^{20}}{62^{\,x}}\;\le\;2^{-20}$$

this gives us,

$$log\left(\frac{1.02\,e^{20}}{2^{-20}}\right)\;\le\;log(62^{x})\;\;which\,is,\;\;log\left(\frac{1.02\,e^{20}}{2^{-20}}\right)\;\le\;x\times log(62)$$

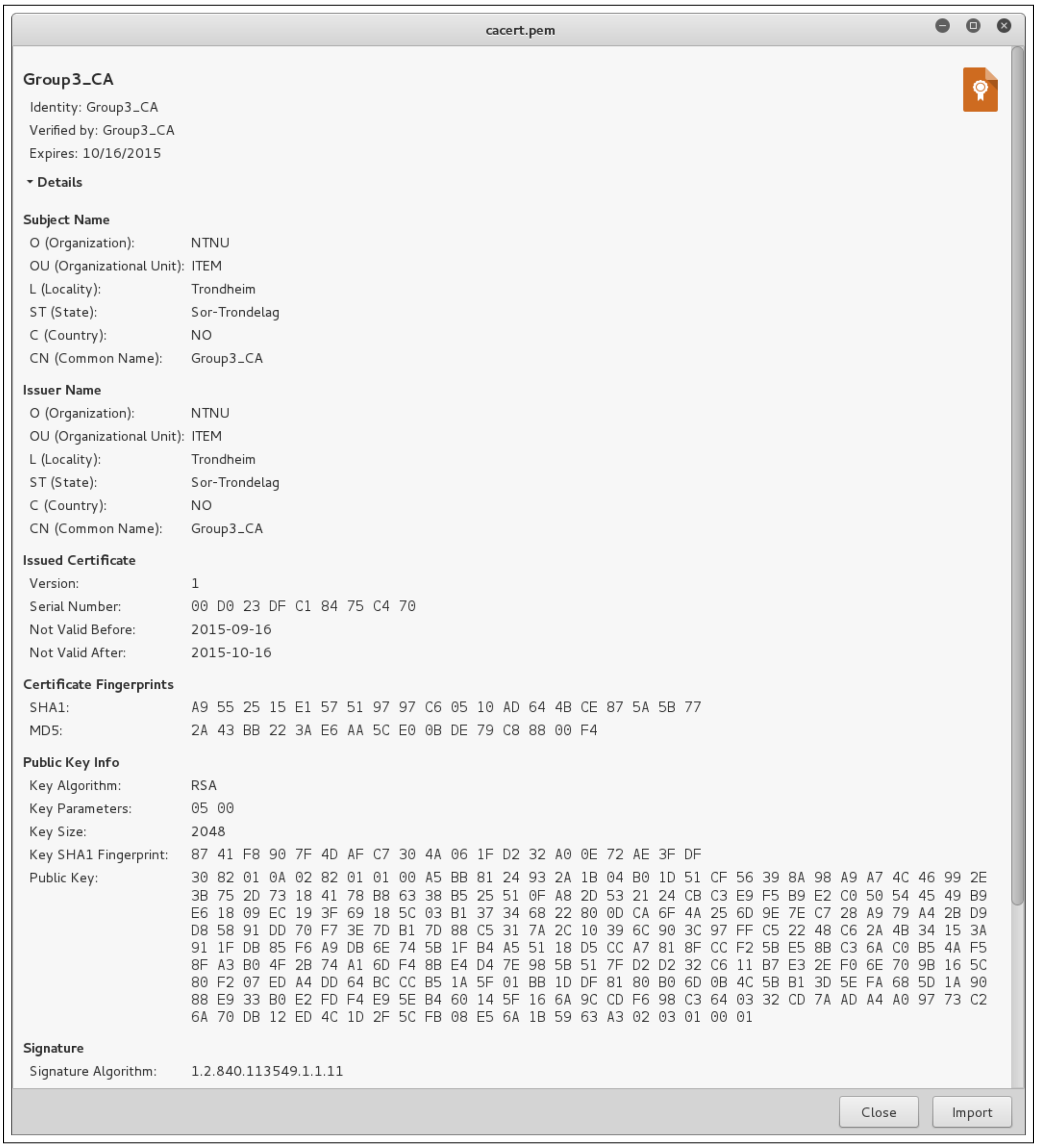
and finally,

$$x \ge \frac{log\left(\frac{1.02\,e^{20}}{2^{-20}}\right)}{log(62)} = 14.522 \approx 15$$

This means our password has to be **at least 15 characters** long, if we are only using upper and lowercase characters and numbers.

Appendix D

The root certificate used in part 3 of the assignment:



```
Group3_CA
  Identity: Group3_CA
  Verified by: Group3_CA
  Expires: 10/16/2015
 ▾ Details

Subject Name
  O (Organization):            NTNU
  OU (Organizational Unit):    ITEM
  L (Locality):                Trondheim
  ST (State):                  Sor-Trondelag
  C (Country):                 NO
  CN (Common Name):            Group3_CA

Issuer Name
  O (Organization):            NTNU
  OU (Organizational Unit):    ITEM
  L (Locality):                Trondheim
  ST (State):                  Sor-Trondelag
  C (Country):                 NO
  CN (Common Name):            Group3_CA

Issued Certificate
  Version:                     1
  Serial Number:               00 D0 23 DF C1 84 75 C4 70
  Not Valid Before:            2015-09-16
  Not Valid After:             2015-10-16

Certificate Fingerprints
  SHA1:                        A9 55 25 15 E1 57 51 97 97 C6 05 10 AD 64 4B CE 87 5A 5B 77
  MD5:                         2A 43 BB 22 3A E6 AA 5C E0 0B DE 79 C8 88 00 F4

Public Key Info
  Key Algorithm:               RSA
  Key Parameters:              05 00
  Key Size:                    2048
  Key SHA1 Fingerprint:        87 41 F8 90 7F 4D AF C7 30 4A 06 1F D2 32 A0 0E 72 AE 3F DF
  Public Key:                  30 82 01 0A 02 82 01 01 00 A5 BB 81 24 93 2A 1B 04 B0 1D 51 CF 56 39 8A 98 A9 A7 4C 46 99 2E
                               3B 75 2D 73 18 41 78 B8 63 38 B5 25 51 0F A8 2D 53 21 24 CB C3 E9 F5 B9 E2 C0 50 54 45 49 B9
                               E6 18 09 EC 19 3F 69 18 5C 03 B1 37 34 68 22 80 0D CA 6F 4A 25 6D 9E 7E C7 28 A9 79 A4 2B D9
                               D8 58 91 DD 70 F7 3E 7D B1 7D 88 C5 31 7A 2C 10 39 6C 90 3C 97 FF C5 22 48 C6 2A 4B 34 15 3A
                               91 1F DB 85 F6 A9 DB 6E 74 5B 1F B4 A5 51 18 D5 CC A7 81 8F CC F2 5B E5 8B C3 6A C0 B5 4A F5
                               8F A3 B0 4F 2B 74 A1 6D F4 8B E4 D4 7E 98 5B 51 7F D2 D2 32 C6 11 B7 E3 2E F0 6E 70 9B 16 5C
                               80 F2 07 ED A4 DD 64 BC CC B5 1A 5F 01 BB 1D DF 81 80 B0 6D 0B 4C 5B B1 3D 5E FA 68 5D 1A 90
                               88 E9 33 B0 E2 FD F4 E9 5E B4 60 14 5F 16 6A 9C CD F6 98 C3 64 03 32 CD 7A AD A4 A0 97 73 C2
                               6A 70 DB 12 ED 4C 1D 2F 5C FB 08 E5 6A 1B 59 63 A3 02 03 01 00 01

Signature
  Signature Algorithm:         1.2.840.113549.1.1.11
```

Figure 4: Screenshot of our self-signed root certificate

Appendix E

Changes made to `eap.conf` in `/etc/freeradius/` in part 3 of the assignment.

```
#TLS
certdir = /home/ttm4137/Desktop/CA/newcerts
cadir = /home/ttm4137/Desktop/CA
private_key_password = birkeland
private_key_file = $cadir/private/askey.pem
certificate_file = $certdir/ascert.pem

#Trusted Root CA list
CA_file = $cadir/cacert.pem
dh_file = $certdir/dh/dh
random_file = $certdir/dh/random
```

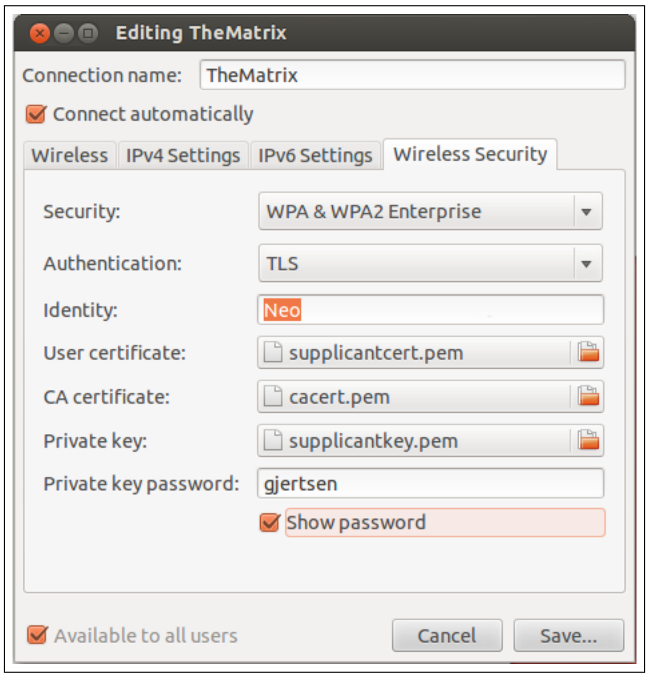
Appendix F

Changes made to `hostapd.conf` file to set up our WPA2-EAP-TLS WLAN AP in part 3 of the assignment.

`ieee8021x = 1`	Enabling 802.1X
`wpa = 2`	Enabling IEEE 802.11i/RSN (WPA2)
`wpa_key_mgmt = WPA-EAP`	Key management
`wpa_pairwise = CCMP`	Encryption mechanism
`auth_server_addr = 127.0.0.1`	AS (RADIUS) address, localhost
`auth_server_shared_secret = gunnar`	Shared secret between AP and AS.

Appendix G

Connecting to the RSN.



Editing TheMatrix

Connection name: TheMatrix

☑ Connect automatically

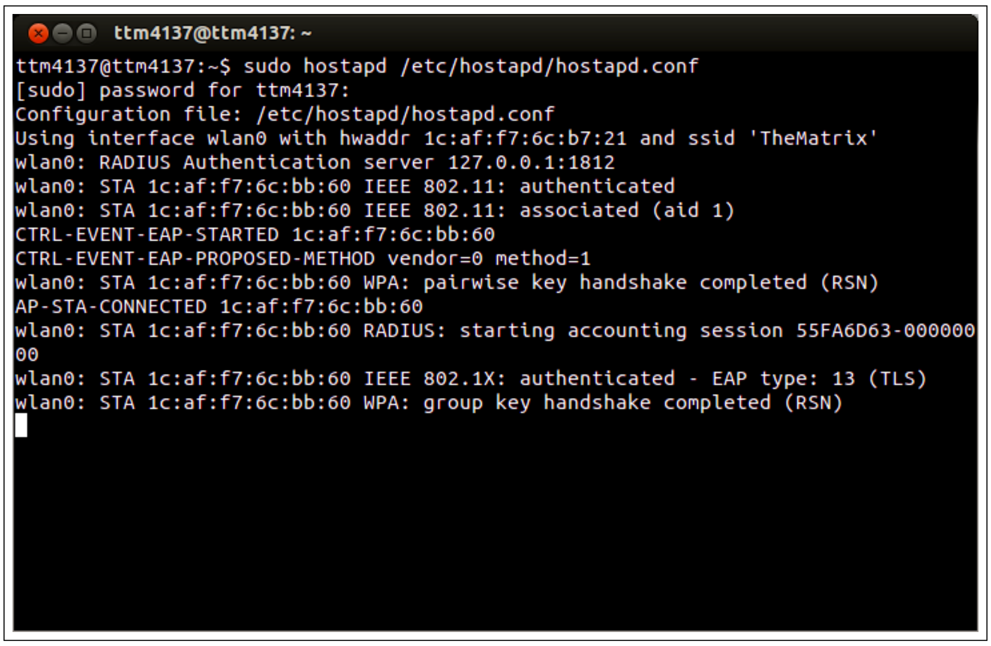
Wireless | IPv4 Settings | IPv6 Settings | Wireless Security

Security:	WPA & WPA2 Enterprise
Authentication:	TLS
Identity:	Neo
User certificate:	supplicantcert.pem
CA certificate:	cacert.pem
Private key:	supplicantkey.pem
Private key password:	gjertsen
	☑ Show password

☑ Available to all users

Cancel | Save...

Figure 5: Client-side connection interface.



```
ttm4137@ttm4137: ~
ttm4137@ttm4137:~$ sudo hostapd /etc/hostapd/hostapd.conf
[sudo] password for ttm4137:
Configuration file: /etc/hostapd/hostapd.conf
Using interface wlan0 with hwaddr 1c:af:f7:6c:b7:21 and ssid 'TheMatrix'
wlan0: RADIUS Authentication server 127.0.0.1:1812
wlan0: STA 1c:af:f7:6c:bb:60 IEEE 802.11: authenticated
wlan0: STA 1c:af:f7:6c:bb:60 IEEE 802.11: associated (aid 1)
CTRL-EVENT-EAP-STARTED 1c:af:f7:6c:bb:60
CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan0: STA 1c:af:f7:6c:bb:60 WPA: pairwise key handshake completed (RSN)
AP-STA-CONNECTED 1c:af:f7:6c:bb:60
wlan0: STA 1c:af:f7:6c:bb:60 RADIUS: starting accounting session 55FA6D63-000000
00
wlan0: STA 1c:af:f7:6c:bb:60 IEEE 802.1X: authenticated - EAP type: 13 (TLS)
wlan0: STA 1c:af:f7:6c:bb:60 WPA: group key handshake completed (RSN)
```

Figure 6: Screenshot from the AP. Authentication successful.