

TTM4137 Wireless Security Lab Assignment 2019

NTNU, Department of Information Security and Communication Technology
August 25, 2019

Contents

0	The Lab	3
0.1	Objectives	3
0.2	Structure	3
0.3	Organisation	3
0.4	Acceptance and Evaluation	5
1	WEP Penetration and Cryptanalysis	8
1.1	Objective	8
1.2	WEP Attacks Historical Background	8
1.3	Objectives and Work Flow	9
1.4	Obtaining a Hidden SSID	10
1.5	Tools and Programs	10
1.6	Questions	16
2	Password Dictionary Attack	18
2.1	Background on a Pre-Shared Key Vulnerability	18
2.2	Objectives and Work Flow	20
2.3	Access Point Set Up with Pre-Shared Key	20
2.4	The Password Dictionary Attack	22
2.5	Questions	23
3	Setting up RSN-EAP Wireless Access Point	25
3.1	Objectives	25
3.2	Elements of Construction	25
3.3	Tools and Programs	27
3.4	Questions	31

4 Mobile Network Security	32
4.1 Objective and Work Plan	32
4.2 Tools and Programs	32
4.3 The GSM Network Setup	34
4.4 GSM Security Analysis	37
4.5 Questions	40
Appendices	42
A ARFCN allocation per group	42
B Programmable SIM Cards	43

0 The Lab

0.1 Objectives

This lab assignment explores many security mechanisms and protocols used for WLANs, and new this year, you will be able to explore mobile network security protocols by setting up and experimenting with a GSM basestation.

The lab work will be both on the analysis side and the constructive side. You will perform a security analysis and assessment of operative WLANs, do some password cracking, and then configure and secure your own IEEE 802.11 RSN access network. Moreover, you will set up a one-cell GSM mobile network based on a software-defined radio device (USRP), and analyse authentication, encryption, and anonymity mechanisms in the GSM system.

The overall objective of this lab assignment is that you will get a better understanding of the weaknesses and the strengths of wireless and mobile network security mechanisms, and experience the challenges of security management.

0.2 Structure

The lab project is divided into four stages, each stage described in one chapter. Chapters 1, 2, and 3 in this document are related to WLANs, while Chapter 4 relates to mobile networks.

First stage will assess the security of the Wired Equivalent Privacy (WEP) protocol. This gives you the experience of how much effort it actually takes to bypass WEP protection by eavesdropping on the radio communication. The second stage will put your hands on the Pre-Shared Key (PSK) mode of Wi-Fi Protected Access (WPA), and ask you to try to find the password by an intelligent exhaustive search method. The third stage asks you to configure your own Extensible Authentication Protocol - Transport Layer Security (EAP-TLS) between the stations and the network, employing the mechanism and protocols of Robust Security Network (RSN). This is the current state-of-the-art security that the IEEE 802.11-2012 standard [14] can offer. Finally in stage four, you will build your own one-cell GSM network and experiment with access control mechanisms (Open Registration, Cached Authentication) and radio channel encryption using your own smartphones.

Proper understanding of information security depends on understanding the threats, which is why a significant part of the lab is devoted to breaching security mechanisms. Please keep in mind that such attack skills can be foul play and may be illegal to apply outside this lab environment.

0.3 Organisation

0.3.1 Work Load

The better part of one week is allocated for the preparations and the lab work. Ideally, there will be three students in each lab group, thus enabling cooperation, sharing of the work, and maybe assign different responsibilities.

Do not postpone but get started as soon as possible on Monday! It is a good idea to start by reading the background material referred to in this document several days before your lab week starts. The group members should agree on a reasonable work schedule (see lab milestones in Section 0.4.2).

A basic knowledge of the Linux environment will be a prerequisite.

0.3.2 Supervision

The teaching assistants will be available to help you and check your work. If you're stuck with a problem, try to solve it in the following order:

1. Consult with the manuals, help pages, or search the web.
2. Ask a group working next to you.
3. Ask one of the teaching assistants during the assistance hours.
4. If all else fails then send an email to `ttm4137@item.ntnu.no`.

0.3.3 Tools

Each group will be working with these devices:

- Two Intel NUC PCs.
- Two wireless network interface cards (NIC).
- One USRP (Ettus B200-mini).
- Two smartphones enabled for 2G access.

The PCs are preconfigured with Ubuntu Linux, and all the drivers and software tools necessary to carry out the lab. Your job will be to perform the correct configuration and use the tools.

When you enter the lab for the first time, choose two computers standing close to each other. The username and password is `ttm4137`. **Change the password** so that other groups don't use your computers throughout the week. Remember to **save all your work** (screenshots, capture files, etc.) **on your personal storage devices**, because the computers will become unavailable to you after the lab completion (most probably on Friday evening).

Disclaimer

We have verified that all of the lab objectives can be carried out on the computers you will be using during the lab. However, if you are to try this out on your own computers and networks, we make no guarantees that the commands in this lab description will suffice. In particular, you are most likely going to have a different NIC chipset than what we have

used in this lab, so the commands pertaining to our specific NIC chipset and drivers, may not work on your machines. Below we describe how the laboratory PC's were prepared for this lab project.

The following packages were added to a blank 32-bit Ubuntu 16.04 LTS Desktop Edition installation: `aircrack-ng` (1:1.2), `kismet` (2013.03.R1b), `john` (1.8.0-2), `wireshark` (2.2.6), `hostapd` (1:2.4), `bridge-utils`, `openssh-server` (1:7.2p2-5ubuntu1) and `freeradius-2.2.8`, supporting `openssl` and EAP-TLS.

OpenBTS suite and all the prerequisites were installed and configured to work with the USRP B200mini. We recommend you to use the computers in the lab and do not try to install the application suite on your own laptops because of incompatibilities that might arise.

0.3.4 Your Creativity

The assignment is formulated as a set of minimum requirements, but we encourage you to let your creative engineering power take you beyond the path we have been staking! For example, try other attacks (like [26, 35]) or modify the ones described here, experiment with your setup, try other EAP protocols, etc. Describe your results in the lab report.

0.4 Acceptance and Evaluation

0.4.1 Submission

The assignment submission consists of two parts:

1. Demonstration of achieved milestones (pass/fail).
2. Lab report (grade).

0.4.2 The Milestones

We have set a few milestones for you, where we will check your work:

Challenge 1: WEP analysis. Demonstrate a terminal window with the calculated WEP key and a working Internet connection over the wireless network.

Challenge 2: Password-based PSK analysis. Demonstrate your WPA/WPA2-PSK setup, a terminal window with the calculated password and a wireless Internet connection.

Challenge 3: Construction of an RSN BSS. Demonstrate your RSN-EAP setup and a working wireless Internet connection.

Challenge 4: 2G mobile networks. Demonstrate the GSM configuration (GSM band, ARFCN allocation, GSM network identity), then enable different types of access (Open Registration, Cached Authentication). Experiment with the programmable SIM cards.

A student assistant must be notified and acknowledge your work progress at these checkpoints. The milestones must be approved by a student assistant by the end of the week.

0.4.3 The Report

Your report will be assessed and count as 20% of the final grade. The report must be submitted via *Blackboard* **by the following Sunday** (one week after you are done with the practical part) for evaluation. Remember to add all group members on the form when uploading the report to *Blackboard*.

A laboratory report is a structured document, usually written and polished after the work is completed. The report is based on notes recorded during the course of experimentation (laboratory journal) and other sources.

The recommended structure of the report:

- Title Page (or heading on the first page).
- Introduction describing the objectives of your work, the lab set-up and some theoretical background.
- Experimental Procedure that *only describes occasions when you did not follow the lab description procedure*.
- Results.
- Answers to the Questions (marked with **Q** in this lab description).
- Discussion.
- Conclusion. Were the primary goals fulfilled? What should have been done differently, and why?
- References.
- Appendices.

The evaluation criteria:

- Format
 - The report should not be longer than 12 A4 pages including text, references and figures, but excluding the title page and appendices.
 - Use 11pt font size, normal line separation, one-column layout and standard margins.
 - The submitted file format must be pdf.

- You should write either in English or in Norwegian, with good grammar, syntax, spelling, and punctuation.
 - Title page (or heading on the first page) should include names of all group members, the group number, date and title.
 - Rules for proper citation must be followed throughout.
 - The reference list should be formatted according to the common rules for citation [38].
 - We recommend using L^AT_EX when writing your report.
- Content
 - Clear and logical structure.
 - Clear identification of problems and objectives.
 - Precision of facts.
 - Presentation, your own analysis and evaluation of the results.
 - Logical discussion part with reasoning that clearly shows that you have understood what you have been doing.
 - Answers to the **Q**-questions (include question numbers and text when answering). All answers should be justified, i.e. no simple yes/no answers.
 - Quality of references. As a starting point you could refer to the course textbooks, but other references are required as well. When you find information on the Internet, it is important that you are careful about its quality. We recommend that you search for information in the university library databases [23, 25], and that you choose references to published articles and books.
 - Your level of understanding of the material.
 - Effort beyond the lab requirements. Presence of new ideas.

Citation and plagiarism It is very important to cite all ideas, quotations, and technical details that you borrow from another source – including course materials. Failure to do so is plagiarism, which is considered cheating, and can have serious consequences on your essay grade, course grade, and academic status. See <http://www.ntnu.no/viko/english/papers/plagiarism> for rules on citations so as to avoid plagiarism.

Remember to cite the source number of at time of reference. If you are referring to an idea, quote, diagram, or technical detail retrieved from another source, the reference number should be present; it is not sufficient to simply have the reference listed in a bibliography. Any citation number should also appear within the sentence that refers to the source. Placing a citation at the end of a paragraph is not acceptable, as it does not distinguish between ideas in the paragraph that are borrowed and that are your own. In such cases, it will be assumed that *none* of the ideas, descriptions, analyses, etc., which are present in the paragraph are your own.

1 WEP Penetration and Cryptanalysis

1.1 Objective

In this first assignment your task is to acquire Internet connectivity via an access point (AP) deploying WEP. You are considered to be a legitimate user which, for educational purposes, is left without the secret key. So it all comes down to finding this key. Note that WEP and TKIP were formally deprecated by IEEE with the release of the 802.11-2012 standard, nevertheless we have chosen to retain this exercise because of its learning value.

1.2 WEP Attacks Historical Background

WEP analysis tools implement different theoretical attacks on WEP, the simplest being a brute force attack (trying every possible key until the correct key is found). Originally, the WEP secret-key was only 40 bits long, which is small enough to make this a feasible attack. The brute-force approach was further simplified in some implementations where algorithms for converting human readable pass-phrases into hexadecimal WEP keys were being used, causing the key space entropy to decrease.

To mitigate the brute force attacks, vendors increased the key size. Today the actual key used to initiate the RC4 stream cipher is a concatenation of a 24 bit Initialisation Vector (IV) and a 104 bit secret key. However, later cryptanalysis of WEP showed that the algorithm's security is in fact independent of its key size because of the problem with keystream re-use. The reason for using IVs is to avoid reusing of keystreams, but the number of 2^{24} different IVs is far too small when each packet sent over the 802.11 channel needs a different IV. With a traffic of 11 Mbps a collision is likely to occur in a matter of seconds.

If you know the plaintext corresponding to the ciphertext (known plaintext attack), you are able to compute the key stream (how?). Also, if the sender reuses a keystream of bits to encrypt a new message, you are able to compute $m_1 \oplus m_2$ (how?). In WEP deploying *Shared Key Authentication* clients are authenticated by the AP with a handshake where the AP sends a plaintext challenge and the client responds with an encrypted version of this challenge. An attacker can simply XOR the challenge with the response to recover the keystream used. Since WEP has no mechanisms for preventing old IV values to be used, this opens up for active attacks where messages can be injected by an attacker without access to the secret key.

In 2001, Fluhrer, Mantin, and Shamir published a ciphertext-only attack against RC4 [10]. The attack, known as the FMS attack, recovers the secret key of RC4 with a high probability if around 4 million encrypted packets are available, and is based on three main principles:

- Some “weak” IV values set up the RC4 cipher in a way such that it can leak key information in its output bytes
- Invariance weakness allows use of the output bytes to determine the most probable key bytes

- The first output bytes are always predictable as they contain the SNAP header (we can partially deploy a known plaintext attack)

This enables the determination of key bytes from observations of the keystream. A single weak IV can determine the correct key byte with a probability of 5%, by collecting a large number of weak IVs the most probable key can be found and tested.

Vendors responded to this attack by filtering out the weak IVs, but they could not catch up as the original attack was refined and further classes of weak IVs were discovered. In 2004 a hacker using the nickname KoreK further improved the practical attacks against WEP with the release of a set of statistical attacks that reduced the amount of packets needed for key-recovery to around 500,000. The history of WEP attacks up to this point is very well described in [4]. The paper also demonstrates the fragmentation attack (related to the chopchop attack described by KoreK), where a single packet can be decrypted without the knowledge of the secret key in order to allow packet injection.

In 2007 Tews, Weinmann and Pyshkin published an attack (the PTW attack) that reduced the number of packets needed to as little as 40-85,000 without the weak IV requirement [34]. The PTW attack is an improvement of an attack proposed by Andreas Klein in [16]. The Klein's and PTW attacks have been thoroughly treated on our facultative lecture.

The FMS, KoreK and PTW attacks are implemented in the **aircrack-ng** tool suite for cracking WEP that will be used in this laboratory project.

1.3 Objectives and Work Flow

This is a brief outline of the minimum requirements for your laboratory activities in part one of this assignment. Details of the tools used will be described in more detail next. Be careful to keep a log of your activities as you proceed in order to prepare material for the laboratory report, remember to take some screenshots of relevant steps that you might want to include in the report. Have a look through the questions for your investigation listed in the end of this section before you start working, to see what theoretical questions you are expected to be able to answer after performing the practical parts of the assignment.

Your *target of analysis* is a Cisco AP that is set up in the lab with 104/128-bit WEP encryption and hidden Service Set ID (SSID).

- Check the wireless interfaces on your computers.
- Run **Kismet** to gather information on the parameters of the wireless LAN under analysis. This will include the AP's BSSID and on what channel the AP is communicating.
- Switch the wireless NICs of your computers to the monitor mode.
- Use the **airodump-ng** tool to obtain MAC addresses of legitimate network clients.
- Obtain network's SSID (a synonym for ESSID).

- Use the gathered info in an attempt to determine the secret key using the **aircrack-ng** tool suite. You should try to find the key using an active PTW attack. Use one PC to send and another PC to dump the traffic, this should speed up your attack.
- Finally, demonstrate the first milestone to one of the lab assistants (see the check-list in Section 0.4.2).

1.4 Obtaining a Hidden SSID

Note: It is important to close down the network manager service first before carrying out any of these steps.

```
sudo service NetworkManger stop
```

If you need to start it again you can use the command:

```
sudo service NetworkManger start
```

The SSID is normally carried in every beacon frame sent by the APs. A hidden SSID is an option enabled by many network administrators hoping to hide their networks. What it means is that the beacons contain the Broadcast SSID (an SSID of length zero) instead of the configured SSID, and it is presumed that the clients participating in the WLAN know this name already. This would seem to “hide” the wireless network, as in this configuration, the only evidence of a wireless network is the MAC address of the AP.

However not only beacons, but also Probe request and response, Association request and Reassociation request frames contain the SSID [19]. Figure 1 shows the 802.11 association procedure that is carried out every time a station connects to an AP. An observation of this communication exchange allows to see a hidden SSID. In order to force a legitimate client to (re)associate, one can send a deauthentication frame using the **aireplay-ng** tool.

1.5 Tools and Programs

1.5.1 Root Privileges

Several tools require superuser access to operate as expected. It is good practice to employ root privileges only where they are needed. You can use **sudo <command>** to execute one command as root, and **sudo su** to permanently switch from your regular user to root. In this lab description, we will usually omit **sudo** from the presentation of a command, for the sake of brevity. It is, however, almost always needed.

1.5.2 Kismet

Kismet [15] is a wireless network detector program. It “sniffs up” all nearby wireless networks and presents detailed information about the APs, such as the BSSID (MAC address) and the SSID (name). **Kismet** can record and dump traffic data to a cap file, which can then be read by the program **Wireshark**. **Kismet** determines a hidden (cloaked) SSID when

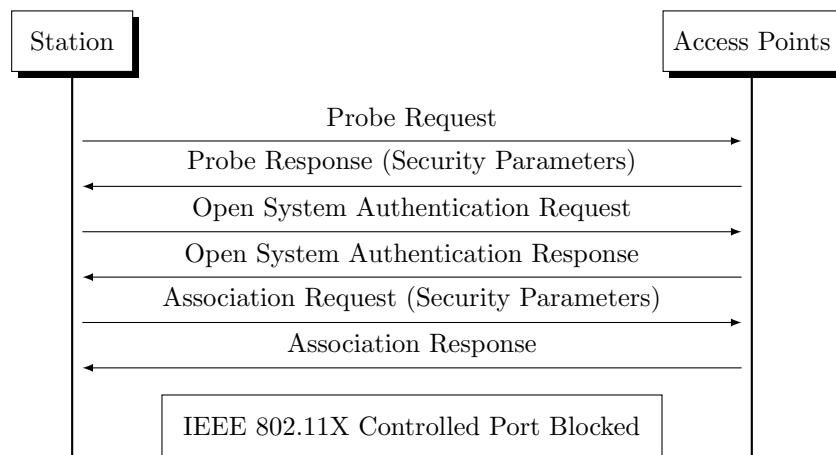


Figure 1: IEEE 802.11 Open System Authentication and Association

frames containing the SSID have been observed. Additionally, Kismet issues alerts (WLAN-specific) to suspicious activity. These alerts can be input to an intrusion detection system, or input to network statistics collection, and much more.

Start the program by typing

```
kismet_server
```

in one terminal window and

```
kismet_client
```

in another window. When the client screen pops up, hit space to close the help screen. Then press 's' followed by 'B' to sort the networks by BSSID. After this you can start using the program. Choose networks using the arrow keys and press enter to display info on them. Hit the 'q' key to go back. If you are interested in a specific network, put the cursor over it and press 'L' in order to lock the channel hopping to this network's channel.

Some of the most used commands are:

- c** Clients. Displays the clients associated with the selected network.
- w** Alerts. Brings up a window to monitor what alerts have been issued.
- q** Cancel. To get back to the previous meny.
- x** Close. Closes a pop-up window (more or less equivalent to q)
- Q** Exits the program.
- h** Brings up the help pop-up screen giving you an overview of the different options

You should stop `kismet_server` by pressing `Ctrl+c` after the necessary information is obtained (see Section 1.3). Further Kismet documentation can be found at [15]. See also a quick tutorial at [36].

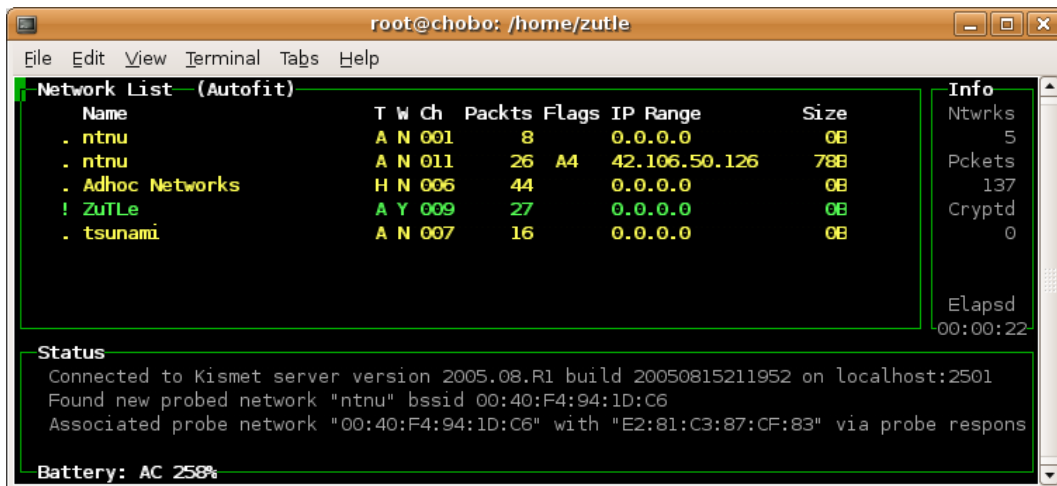


Figure 2: Kismet client interface.

1.5.3 Configuring The Wireless Network Interface Cards

Before we can start launching our attack, let's make sure our hardware and software are set up correctly. We are using wireless NIC's based on the Atheros chipset, with the open-source drivers `ath5k`. When you have attached the external wireless NIC, there will be two wireless cards attached to the NUC, along with an internal card.

These can be seen by running

```
iwconfig
```

You will be using the external card, which will have a long name that looks something like `wlxc4e984ddc451`. Find the unique name of your card and note it down. From now on we will refer to it as `wlan0`.

NOTE: from now on, throughout the entire lab, we refer to the interface as `wlan0`. For example, if the interface is recognised as `wlxc4e984ddc451`. In that case, replace every instance of `wlan0` with the interface `wlxc4e984ddc451`.

The card should be added as source in kismet. To see more information about the network interfaces on your PC, type:

```
ifconfig
iwconfig
```

The second command outputs more details about the wireless interfaces. In this assignment we will be using three different modes for our wireless NIC:

Managed Normal mode.

Master Virtual Access Point.

Monitor Raw monitoring mode. Used to passively sniff traffic (without responding to RTS/CTS and the like)

Your wireless NIC needs to be in monitor mode before you can start listening to traffic from surrounding networks using the **aircrack-ng** tool suite. A convenient way of putting your NIC into monitor mode is by using the script **airmon-ng** which is included in the **aircrack-ng** tool suite. By default, it creates an (virtual) interface named **mon0** in managed mode at the startup. It is not recommended to have more than one **mon** interface working at the same time. So first you have to destroy all the existing interfaces:

```
airmon-ng
airmon-ng stop mon0
airmon-ng stop mon1
...
```

Then ensure that the wireless interface is not already in use by stopping it¹

```
ifconfig wlan0 down
```

finally, create a new interface in monitor mode listing on a specific channel:

```
airmon-ng start wlan0
iw dev wlan0 set channel 6
```

Verify that the interface is listening on the right channel by running:

```
iwlist <interface> channel
```

1.5.4 airodump-ng, aircrack-ng and aireplay-ng

Aircrack-ng [1] is a suite of programs (**airodump-ng**, **airmon-ng**, **aireplay-ng**, ... etc.) for analysing WEP and WPA/WPA2 protected networks. We will be using (and describing) some of these programs in this lab.

Airodump-ng is a packet capture program that sniffs traffic. You can run **airodump-ng** like this:

```
airodump-ng <options> <interface>
```

The following example runs **airodump-ng** on the interface **mon0**, dumping the traffic of the WLAN with BSSID 00:11:22:33:44:55, to the file **filename-01.cap**:

```
airodump-ng --write filename --channel 6 --bssid 00:11:22:33:44:55 wlan0
```

Note that you have to complete the instructions of Section 1.5.3 and set the correct channel, before running **airodump-ng**. It is also convenient to see the list of network clients on the **airodump-ng** screen (Figure 3). A traffic dump file, output by **airodump-ng**, can be fed to **aircrack-ng** or to **Wireshark**.

¹If you need to bring it up again, run: `ifconfig wlan0 up`.

BSSID	PWR	Beacons	# Data	CH	MB	ENC	ESSID
E2:81:C3:87:CF:83	9	4	0	6	11	OPN	hpsetup
00:15:C7:FF:1E:80	60	341	21595	4	54	WPA	Zutle

BSSID	STATION	PWR	Packets	ESSID
00:15:C7:FF:1E:80	00:09:2D:52:B6:05	27	26	Zutle
00:15:C7:FF:1E:80	00:0F:CB:B3:13:9A	49	44739	Zutle

Figure 3: Airodump.

When `airodump-ng` has gathered enough packets, you can start the PTW attack by running `aircrack-ng`. Note that the PTW attack requires a cap file as input. Keep the `airodump-ng` program running as well to get even more material for the cracking process. The more IVs, the less time it takes to recover the key (in general). `Aircrack-ng` has the following synopsis:

```
aircrack-ng <options> <capture file(s)>
```

The following example runs `aircrack-ng` PTW against an AP with a given BSSID using our cap dumpfile:

```
aircrack-ng --bssid 00:11:22:33:44:55 filename-01.cap,
```

or if you have more than one capture file:

```
aircrack-ng --bssid 00:11:22:33:44:55 filename*.cap.
```

Figure 4 demonstrates the insecurity of WEP².

The time it takes to find the key and break WEP is proportional to the amount of traffic you have captured. However, you can speed up the process by using traffic injection tools. There are several available attacks in `aireplay-ng`, and you will have to specify one when you run the program (this is the first parameter passed from the command line). A brief description of the attacks supported by `aireplay-ng` is provided below. Note that in this lab, you do not necessarily have to use every tool. Visit [1] for a more detailed information and command syntax.

²This key is found in ten seconds, after injecting arprequests for about three minutes.

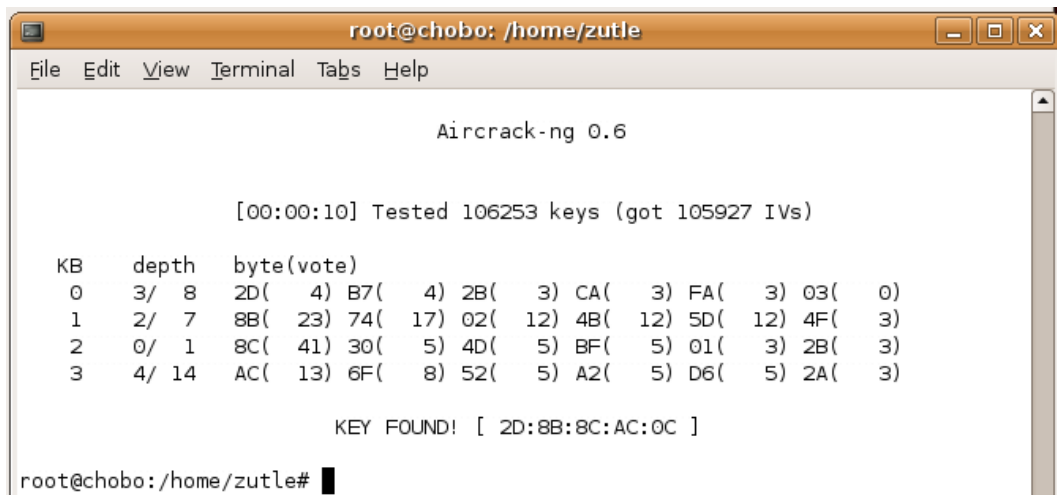


Figure 4: Aircrack-ng.

Fake authentication Use this to associate with the AP. This is required for launching the attacks below³.

Deauthentication Use this when there is a lot of clients associated with the target AP. It will deauthenticate the users, forcing them to reauthenticate and hence generate traffic.

ARP-request reinjection Use this attack to automatically sniff up ARP-requests and inject them into the network. You will need the MAC address of an associated client (either use fake authentication (above), or set your MAC address to resemble that of an already associated client).

Interactive packet replay Use this to inject a packet of your own choice. You can use **wireshark** to have a look at some of the captured traffic, and then try to find a request and a response.

KoreK chopchop This attack decrypts a single packet. If you can manage to decrypt a packet, you can actually forge your own arprequests using **packetforge-ng**, this can be useful if there are no clients associated with the AP.

Fragmentation attack Decrypts longer packets than chopchop and is much faster.

In order to perform packet injection, you will have to be associated with the AP first. To use the fake authentication attack, use the following command:

```

aireplay-ng --fakeauth <delay> -a <bssid> -e <essid> <interface>
aireplay-ng --fakeauth 1000 -a 00:11:22:33:44 -e networkname mon0

```

³Alternatively you can set the source MAC address in the transmitted packets to be the same as for an already authenticated client, i.e. to masquerade. Use the **aireplay-ng -h** option for this.

To inject traffic, use the ARP-replay attack⁴

```
aireplay-ng --arpreplay -b <bssid> <interface>  
aireplay-ng --arpreplay -b 00:11:22:33:44 mon0
```

To deauthenticate a user, issue the following command:

```
aireplay-ng --deauth <#deauths> -a <bssid> -c <client> <interface>  
aireplay-ng --deauth 10 -a 00:11:22:33:44 -c 99:88:77:66:55 mon0
```

where 10 is the number of deauthentication packets to send, and -c the MAC of the client you want to deauthenticate. Do not let this attack run indefinitely, since it might completely disrupt the legitimate client from accessing the network. Hence you will not be able to capture any more IV's.

1.5.5 Wireshark

Wireshark is a network protocol analyser with a convenient graphical interface. When you have recovered an encryption key, you can decrypt the IEEE 802.11 packets and read the plaintext data. Open a .cap file obtained by `airodump-ng`, go to Edit → Preferences → Protocols → IEEE 802.11 and enter the decryption key. Remember to enable the check-box labelled “Enable decryption”. Among the decrypted packets, try to find some web traffic from the legitimate client. You can get more info about Wireshark at [11].

1.6 Questions

After performing this part of the assignment you should be able to answer these questions, please include the answers in your laboratory report.

Q1. Describe the parameters of the AP under analysis, such as the SSID, BSSID, channel number (and optionally the frequency), encryption mechanism, associated clients, the WEP key. How many packets did the PTW attack require? Which web page was the legitimate client browsing?

Q2. Is it possible to run a completely passive PTW attack on 104-bit WEP? Why and under which circumstances?

Q3. Why is it possible to send an arbitrary amount of ARP-requests to the AP without knowing the WEP key?

Q4. What can you do to strengthen your attack when there are some clients, but hardly any traffic at all?

Q5. How can you obtain packets for injection if no clients are associated with the AP?

Q6. Which weaknesses of WEP are we taking advantage of in our attack, and why?

⁴Note that there is some inconsistency in how the different attacks handles the BSSID flag, i.e. for all attacks except for `arpreplay`, the BSSID is denoted by `-a`, but for `arpreplay` you need to use `-b`.

Q7. Under what circumstances would you consider WEP to be sufficiently secure?

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?

2 Password Dictionary Attack

In this part you will set up a WPA-PSK or WPA2-PSK protected WLAN and mount a password dictionary attack against it.

2.1 Background on a Pre-Shared Key Vulnerability

Wi-Fi Protected Access (WPA) is a certification program created by the Wi-Fi Alliance while waiting for the IEEE 802.11i standard to be finished. WPA devices implement a subset of the IEEE 802.11i standard, including the Temporal Key Integrity Protocol (TKIP).

Since the ratification of the 802.11i standard in 2004, the Wi-Fi Alliance refers to their approved implementation of the full 802.11i as WPA2. It differs from its ancestors, among the other things, in the encryption algorithm, which is now AES (Advanced Encryption Standard) instead of RC4. The new AES-based authenticated encryption algorithm is called CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol).

Both WPA and WPA2 use a pairwise key hierarchy defined in the IEEE 802.11i standard shown in Figure 5. Here, $f(\cdot, \cdot, \cdot, \cdot, \cdot)$ is a pseudo-random function producing 512 bits of output (see also Equation 1 below).

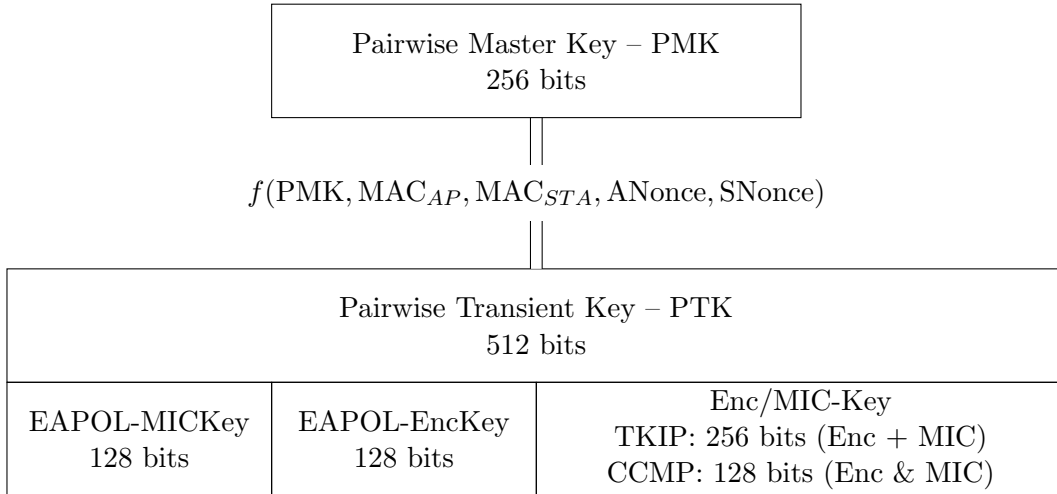


Figure 5: RSN Pairwise Kay Hierarchy

WPA and WPA2 security mechanism comes in two flavours:

PSK (Pre-Shared Key) used in Small office/Home office (SoHo) networks. A PSK is shared among all of the users.

Enterprise doesn't use PSK but one of several types of EAP (Extensible Authentication Protocol) for authentication. This mode of operation is the medium-sized and enterprise's choice, where the use of a single PSK is discouraged (see Section 3).

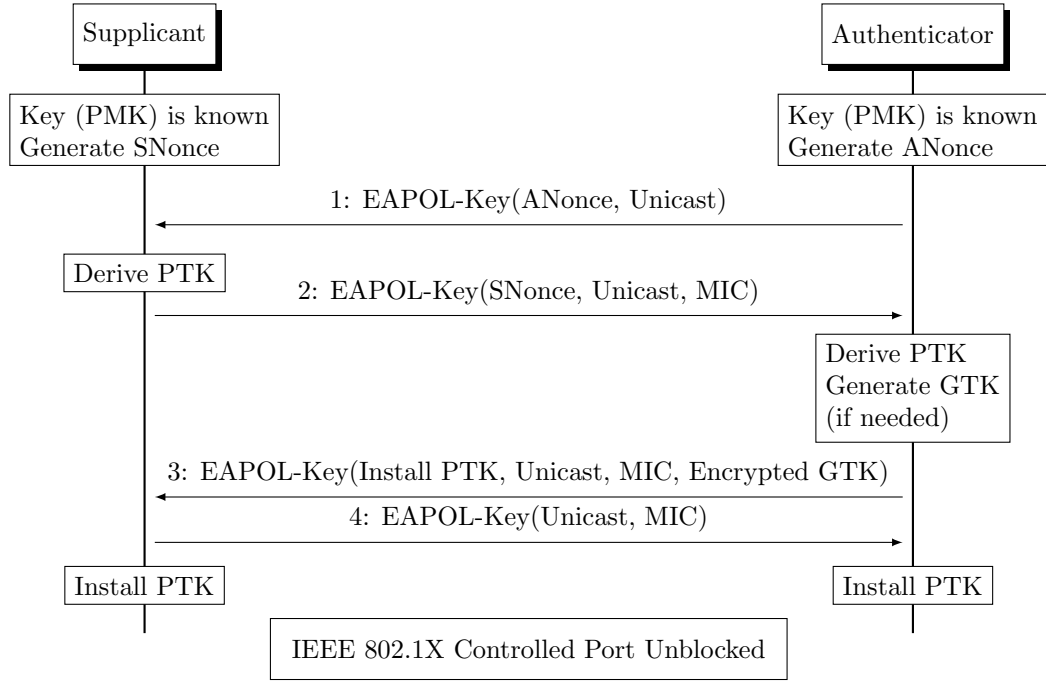


Figure 6: The 4-Way Handshake [14].

We refer to [7, §8 and §9], [18] and [2] for more details.

The station authentication and association procedure in the PSK mode is depicted on Figure 8, except for the Stage 3 (802.1X authentication) which is not being performed. A string of 256 bits is used as a common secret PSK shared between the stations and the AP. It can be input either directly as 64 hexadecimal digits, or using a more user-friendly password method. We will be interested in the latter approach as it is less secure. A password may be from 8 to 63 printable ASCII characters. It is then used to generate PSK using a known algorithm:

$$\text{PSK} = \text{PBKDF2}(\text{password}, \text{SSID}, \text{SSIDlength}, 4096, 256),$$

where PBKDF2 (Password-Based Key Derivation Function) is a key derivation function defined in PKCS#5 v2.0 standard. 4096 is the number of hashes and 256 is the length of the output.

The Pairwise Master Key (PMK) is then computed as

$$\text{PMK} = \text{PSK}.$$

The Pairwise Transient Key (PTK) is derived from the PMK using the 4-Way Handshake, and all information used to calculate its value is transmitted in plain text (see Figure 6).

Here ANonce is an authenticator's nonce and SNonce is a supplicant's nonce.

The unpredictability of the PSK and the PMK is determined by the quality of the password. The password could be subjected to both dictionary and brute force offline attacks. An attack against PSK password was originally proposed by R. Moskowitz in [20]. An attacker captures the 4-Way Handshake messages, either by passively monitoring the wireless network traffic, or actively generating deauthentication frames to speed up the process. In fact, the first two messages are required to start guessing at PSK values. Note from Figure 5 that

$$\text{PTK} = f(\text{PMK}, \text{MAC}_{AP}, \text{MAC}_{STA}, \text{ANonce}, \text{SNonce}), \quad (1)$$

where PMK equals PSK in our case. The attacker reads the MAC addresses and ANonce from the first message and SNonce from the second message. Now the attacker can start *guessing* the PSK value to calculate the PTK and the derived KCK (Key Confirmation Key, an integrity key protecting handshake messages). If the PSK is guessed correctly, the Message Integrity Code (MIC) of the second message could be obtained with the corresponding KCK, otherwise a new guess has to be made.

The program `cowpatty` was created to exploit this flaw, and its source code is included in the `aircrack-ng` to allow PSK dictionary and brute force attacks on WPA and WPA2. The protocol design (4096 hashes for each password attempt) means that a brute force attack is very slow (just a few hundred passwords per second with the latest single processor). Note that it is not possible to pre-compute a table of PSK from a dictionary of potential passwords because the SSID is also an argument to the PBKDF2().

2.2 Objectives and Work Flow

A sketch of your work plan is the following.

- Set up a WPA-PSK or WPA2-PSK wireless AP on one of your PCs.
- Use your personal laptop or other WLAN-enabled device to connect to your AP.
- Use `aircrack-ng` and `john` on your second PC to perform a password dictionary attack against your WLAN.
- Demonstrate your results in Milestone 2 (see the check-list in Section 0.4.2) to one of the teaching assistants.

2.3 Access Point Set Up with Pre-Shared Key

Any wireless NIC that can be put into `master mode` can function as an AP. We will use the server program ("daemon") `hostapd` in conjunction with a wireless NIC in master mode to implement an 802.11i standard AP.

2.3.1 brctl

To provide Internet access for your clients, you will first have to set up a bridge between the Ethernet and wireless interfaces on the AP machine. To do so, we will be using a tool called `bridge-utils`. Create a script file `brup.sh` with the following contents:

```
#!/bin/bash

brctl addbr br0
brctl addif br0 eno1
ifconfig br0 up
dhclient br0
```

Make the script executable and run it:

```
chmod +x brup.sh
sh brup.sh
```

To check the status of the bridge, type:

```
brctl show
```

At some point you might need to delete the bridge, so also create the script `brdown.sh`:

```
#!/bin/bash

ifconfig br0 down
ifconfig eno1 0.0.0.0 down

brctl delif br0 eno1
brctl delbr br0

ifconfig eno1 up
dhclient eno1
```

Again, remember to make this script executable with `chmod`.

2.3.2 Hostapd

`hostapd` is a user space daemon for access point and authentication servers. It implements IEEE 802.11 access point management, IEEE 802.1X/WPA/WPA2/EAP Authenticators, RADIUS client, EAP server, and RADIUS authentication server. In this part, we will use it to set up a WPA access point that uses a password-based PSK. Additionally we will

use it to set our wireless NIC into master mode⁵. In Part 3 we will use it to implement a WPA2/EAP Authenticator, and use **FreeRadius** to implement an external RADIUS authentication server.

All configuration of **hostapd** is based on the file `/etc/hostapd/hostapd.conf`, so start by making a backup copy of it. Then begin editing the copied file. Here you will have to specify different security settings for your virtual AP. Start by setting up **hostapd** as simple as possible, minimizing the editing of the conf file. The lines:

```
interface=wlan0
bridge=br0
driver=nl80211
```

in `hostapd.conf` specify the name of your wireless interface, the bridge and the name of your wireless NIC driver. The bridge connects the wireless interface of the AP with the Ethernet network card, and is necessary in order to provide Internet access for your clients.

Make other necessary configurations as well. Remember, your aim is to set up a WPA access point that uses a password-based PSK.

Give some thoughts to the selection of the password that will establish the PSK. You want to adhere to the following, or similar, restrictions in order to make this experiment both feasible and challenging. The rules of the password selection “game” should satisfy the following conditions:

- the password stems from one and only one properly spelled English word;
- the password includes one or two digits/punctuation marks;
- not more than one capital letter.

Let one member of your group secretly construct the password and carry out the configuration, then the rest of you will try to find it and attack your network.

Start the daemon by executing

```
hostapd -d <pathToConfigFile>
```

The parameter `-d` sets the program in debug output mode. For more detailed information on configuring **hostapd**, see the comments in `hostapd.conf` and [3].

Use your personal laptop, mobile phone or other WLAN-enabled device to connect to your AP (if you don’t have a single WLAN device, ask a student assistant).

2.4 The Password Dictionary Attack

2.4.1 airodump-ng, aireplay-ng and aircrack-ng

A tutorial [5] will be your main guide.

⁵There are various ways of setting your wireless NIC card into master mode, depending on your drivers. For our drivers (nl80211), **hostapd** is necessary to achieve this.

Use `airodump-ng` to record the traffic on your WLAN in order to capture the 4-way authentication handshake. Don't be tempted to manipulate your STA or AP. To speed up the process of catching a 4-way handshake you may try the active deauthentication attack of `aireplay-ng` as explained in Section 1.5.4. Note that `airodump-ng` may in some cases not tell you when you get the handshake. You can try to find the 4-way handshake in your dump using `Wireshark`, or just proceed to the next step if you believe that you've already got a handshake.

Locate and download a good dictionary from the Internet. For the sake of feasibility don't bet for more than 30 000 words. We suggest the tiny wordlist from [27]. Finally, use `aircrack-ng` to run the attack.

2.4.2 john

John the Ripper (`john`) is one of the most popular password testing programs. Since your password isn't just a plain word, we'll use `john` to apply different variations to our dictionary words. The syntax for applying different rules to the wordlist is given in [28].

Start with the default set of rules to produce your extended dictionary:

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

The argument `-stdout` tells `john` to output words to the standard output.

Try to run `aircrack-ng` on your extended wordlist as so:

```
aircrack-ng -a wpa -b <your_bssid> -w extendedwords.lst  
          <your_capture_files(s)>.cap
```

where `-a wpa` means the WPA/WPA2-PSK attack mode. While it runs, look at the rules you're using in the section [List.Rules:Wordlist] of `/etc/john/john.conf`. If it takes really long to find your password with the default rules, consider cooperation with your fellow that knows the password in order to write your own rules to speed up the process.

There's a way to go without a temporary file by redirecting the output from `john` to `aircrack-ng`:

```
john --wordlist=words.lst --stdout --rules | aircrack-ng -a wpa  
          -b <my_ssid> -w - <your_capture_file(s)>.cap
```

2.5 Questions

Please include the answers to the following questions in your laboratory report.

Q9. Why is TKIP a more secure encryption mechanism than the one-key WEP alternative?

Q10. Which information is used to compute the PMK and the PTK in the password-based PSK scenario? Why is the offline password brute-force attack possible?

Q11. Does the compromise of password imply the disclosure of traffic from previous sessions? Justify your answer.

Q12. Does the use of AES in WPA2-PSK give an advantage against a password dictionary attack, as compared to RC4 in WPA-PSK?

Q13. What would be advantages and disadvantages of using a precomputed database of PMKs in a PSK password dictionary attack?

Q14. Suppose an attacker employs 30 days of processing time on the new supercomputer at NTNU (find out its processing power at [24] and recall the password enumeration speed in the lab). Which minimum requirements would you put on a WPA password to ensure that the probability of successful attack does not exceed 2^{-20} ?

3 Setting up RSN-EAP Wireless Access Point

3.1 Objectives

In this part you will set up a wireless AP and configure its security features. You will be implementing the RSN-EAP configuration of the IEEE 802.11-2007 based on two computers equipped with wireless NICs. One will run the authenticator, and also the Radius authentication server (AS), the other will be the supplicant. You should employ:

1. CCMP for enforcing link confidentiality.
2. Mutual authentication of supplicant and AS employing digital certificates and the EAP-TLS authentication protocol.

3.2 Elements of Construction

The system and its components are shown in Figure 7.

Supplicant. A WLAN station requesting wireless Internet access will act as a supplicant in the security protocols.

Authenticator. The role of the AP. Use the `hostapd` utility to set up the interface.

Authentication Server. Normally, this role is run by one centralized machine, but in this lab project you will use the same machine to run both an authenticator and an AS. The role of AS is implemented by the software package `FreeRadius`.

When a supplicant attempts to associate with the authenticator, it will rely on the client software `wpa_supplicant` to initiate the authentication request. The authenticator will then forward the supplicant's request and subsequent EAP messages to the AS, and

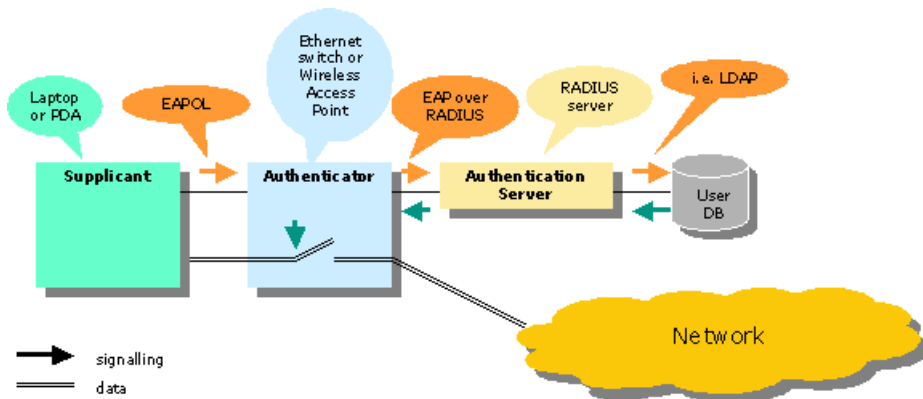


Figure 7: The controlled port concept of 802.1x

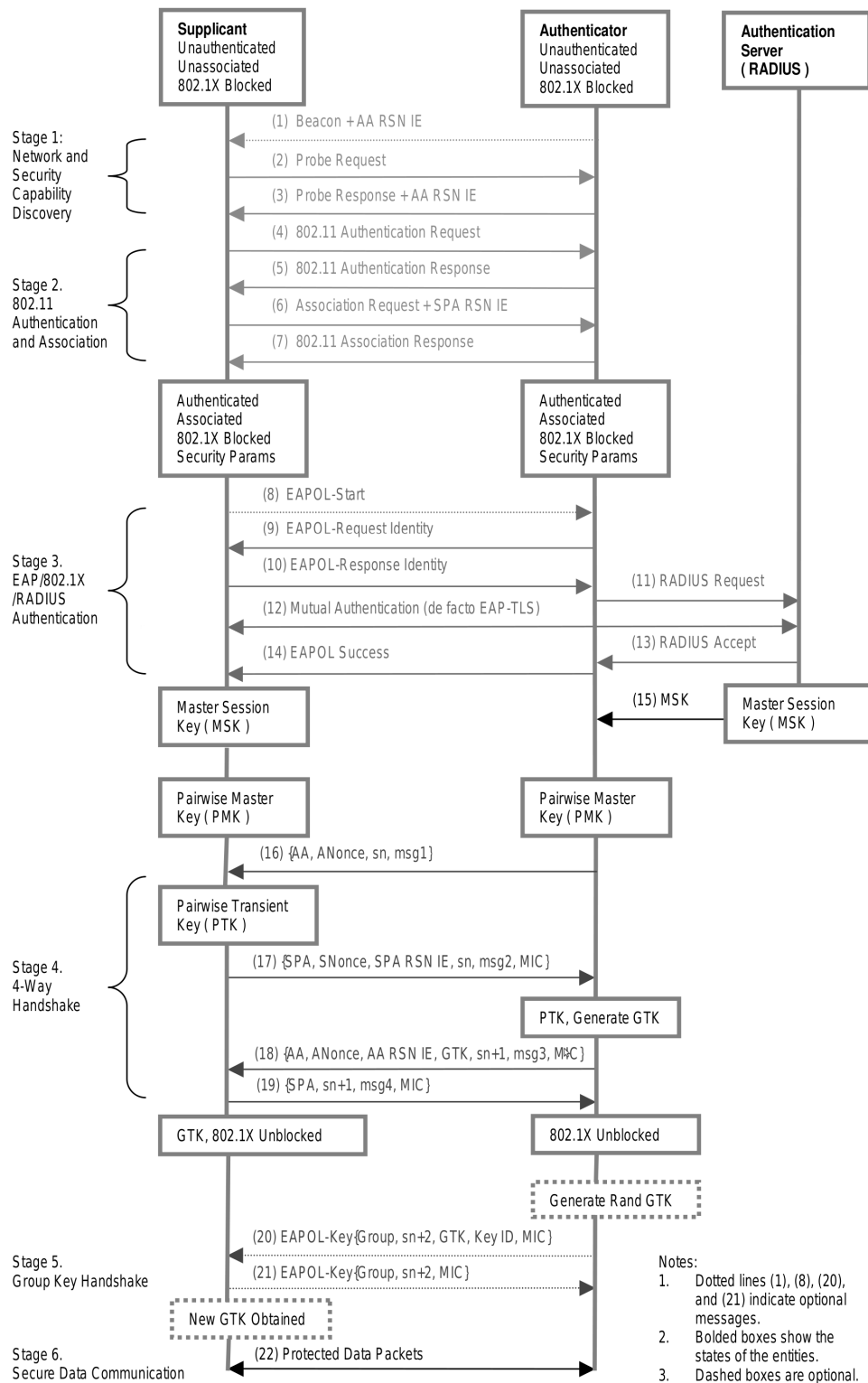


Figure 8: Robust Security Network (RSN) Association Establishment Procedures [12].

send AS's EAP messages back to the supplicant. Only after the supplicant is authenticated will it be able to associate with the authenticator (see Figure 8).

There are many EAP protocols that include the use of public-key certificates. Some employ certificates only on the server side, like EAP-PEAP, while others, like EAP-TLS, can use client certificates as well. We will use public-key certificates on both client and server side to ensure mutual authentication in the lab set up.

You will set up an authentication mechanism based on EAP-TLS (considered to be the most robust protocol). You should modify the configuration files (they are split up across several text files according to their functionality), establish your own certificate authority (CA) and create certificates for the server and at least one user. Use `OpenSSL` tool for this. After having created the private keys and certified the public keys indicate the correct paths for the certificate files in the configuration files of both the supplicant and the Radius server.

We recommend reading [7, §8 and §9] or [14, §5.4.3 and §8] to get the background and understanding of the framework you are going to set up in this part.

3.3 Tools and Programs

3.3.1 OpenSSL

Now you separate the responsibilities for each group member, so that one of you is the CA, another one the server owner, and the third one is representing the client. Do the following:

- Create a private, a public key and a self-signed certificate to be used by your CA.
- Create a private and a public key for the server.
- Create a private and a public key for the client.
- Sign the server's and the client's certificates with the CA's private key.

Although you might be familiar with the `OpenSSL` [9] tool already (recall the TTM4135 course), we give a quick summary of the required commands here. We use a neat tutorial [29] as a basis.

```
mkdir CA
cd CA
mkdir newcerts private
echo '01' > serial
touch index.txt
```

Note that when copy-pasting from pdf, some symbols, as, for example, quotation marks, may be treated wrongly (make sure that the file `serial` contains 01).

Now, download the configuration file `openssl.cnf`, provided at the bottom of the page of [29], into the `CA` folder and inspect the file. Update security-related parameters with respect to the strength of the cryptographic primitives. Also set the following values in order to avoid entering them several times:

```
# ./openssl.conf

# Default values for the above, for consistency and less typing.
# Variable name          Value
#-----
0.organizationName_default = NTNU
organizationalUnitName_default = ITEM
localityName_default = Trondheim
stateOrProvinceName_default = Sor-Trondelag
countryName_default = NO
```

Let's create the CA's self signed certificate first:

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

During this command execution you will be prompted to create a password for storing the CA's private key.

Then we create certificate requests for the supplicant and the AS. Each new private key will be stored with a new password which you will create:

```
openssl req -new -out supplicantreq.pem
  -keyout private/supplicantkey.pem -config ./openssl.cnf
openssl req -new -out asreq.pem -keyout private/askey.pem
  -config ./openssl.cnf
```

To sign the supplicant's and the AS's public keys, type:

```
openssl ca -out newcerts/supplicantcert.pem -config ./openssl.cnf
  -infiles supplicantreq.pem
openssl ca -out newcerts/ascert.pem -config ./openssl.cnf
  -infiles asreq.pem
```

Use the following command to display the information about a certificate:

```
openssl x509 -in <CertificateFileName> -noout -text
```

You can transfer certificates and keys to clients by using `scp` (secure copy) in one of the following ways:

```
scp <username>@<ipaddress>:<RemoteSourceFile> <LocalDestinationFolder>
scp <LocalSourceFile> <username>@<ipaddress>:<RemoteDestinationFolder>
```

3.3.2 Authentication Server

Use your AP machine from part two. The back-end AS will be implemented on the same machine as the one running `hostapd`.

FreeRadius

FreeRadius has several configuration files, but the only ones you will have to modify are `eap.conf` and `clients.conf` in the folder `/etc/freeradius/`. In the former file, set the default EAP method to TLS, and locate the part of the config file specifying the TLS module. After you have created the keys and certificates with **OpenSSL**, you will have to give the correct paths for the key and certificate files. You will also need a file with Diffie-Hellman parameters and another one with random data⁶. Create them by typing

```
openssl dhparam -out dh 1024
openssl dhparam -out random 256
```

Before you start **freeradius**, you may have to stop an already-running instance by typing

```
/etc/init.d/freeradius stop
```

Run the server by typing

```
freeradius -X
```

This will start the daemon in the foreground, letting you see what is going on.

Hostapd

Configure **hostapd** as an AP deploying EAP-TLS, CCMP and an external Radius authentication server (that will run on the same PC). Note that you will have to make more edits now than in Part 2, where you only used pre-shared keys. When you are specifying the IP address of the Radius server, simply use `127.0.0.1` (loopback), or `localhost`. Make sure you comment out the PSK specific configurations used previously in the lab. Also ensure that your bridge is set up properly (if necessary, delete it first, then recreate it, as described in Section 2.3.1). Start the **hostapd** process.

3.3.3 Supplicant Side

The computer not used for the authenticator and authentication server, will fill the role of the supplicant. First make sure that the NIC is set into managed mode. To do this, first stop all interfaces:

```
airmon-ng stop mon0
airmon-ng stop mon1
...
```

Now, the connection can be established manually simply by accessing the network icon in the top right corner and selecting the right network. You will then be prompted to fill out all necessary settings and credentials. The supplicant will sometimes not obtain an IP-address automatically after having associated (and authenticated) with the access point. Force a DHCP-request, with the following command:

⁶You can also use `/dev/urandom` as your random file.

```
dhclient wlan0
```

The process above can be automated with the tool `wpa_supplicant`. However, this tool may conflict with the NetworkManager service, so first we will stop it⁷:

```
service network-manager stop
```

Then, we have to create a configuration file for `wpa_supplicant`, holding the parameters needed for EAP-TLS. You can use the following template:

```
#####
# WPA Supplicant config file suggestion
# WPA2-EAP/CCMP using EAP-TLS
#####

ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="your net's ssid"
    key_mgmt=WPA-EAP
    proto=WPA2
    pairwise=CCMP
    group=CCMP
    eap=TLS
    ca_cert="/mylocation/cacert.pem"
    client_cert="/mylocation/supplicantcert.pem"
    private_key="/mylocation/supplicantkey.pem"
    private_key_passwd="passhrase"
    identity="any name goes"
}
```

The configuration parser is very picky, for instance, don't include additional spaces in the configuration file. This will give an error saying that the file can't be parsed.

Before we start `wpa_supplicant`, make sure that any old processes are removed:

```
rm /var/run/wpa_supplicant/wlan0
```

Now run `wpa_supplicant` by the following command:

```
wpa_supplicant -D nl80211 -i wlan0 -c <pathToConfigFile> -d -K
dhclient wlan0
```

where the `-d` flag means debug mode (which will provide some more output) and the `-K` flag will include keys and passwords in the output as well.

There is also a graphical user interface associated with `wpa_supplicant`, called `wpa_gui` (Figure 9). To run it, simply type `wpa_gui` (as root) in another terminal window, after you have started the `wpa_supplicant` program.

⁷To start NetworkManager again later, run: `service network-manager start`.

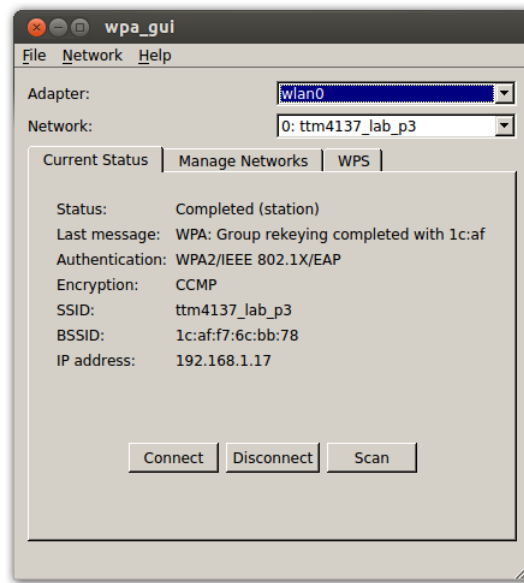


Figure 9: The `wpa_supplicant` with its graphical frontend `wpa_gui`.

3.4 Questions

Please include the answers in your laboratory report.

Q15. When would it be appropriate to use WPA2-PSK instead of WPA2-EAP (as a key management scheme)?

Q16. EAP-TLS deploys mutual authentication of two communicating parties. What kind of attack is possible against authentication protocols lacking such authentication?

Q17. Give an overview of the EAP authentication protocols which can be used in WPA2-Enterprise WLANs.

Q18. List the security-related protocols used in your RSN-EAP-TLS setup and explain their purpose.

4 Mobile Network Security

4.1 Objective and Work Plan

The objective of this part is to explore some security aspects of mobile networks by setting up and configure a GSM one-cell test network. You will use a USRP (Universal Software Radio Peripheral) device controlled by a computer that runs an open-source GSM software. You will configure several access parameters, different types of network access control, and enable radio-link encryption. Finally, you will compare the security mechanisms enabled during this lab exercise with the security mechanisms in commercial mobile access networks.

The work plan contains the following tasks:

1. Set up a one-cell GSM network.
2. Enable ‘Open Registration’.
3. Enable ‘Cached Authentication’.
4. Enable ‘Full Authentication’.
5. Enable link encryption.

You must demonstrate your results of each task to the teaching assistant.

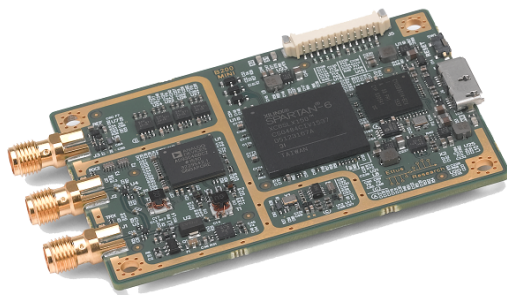
4.2 Tools and Programs

4.2.1 USRP B200mini

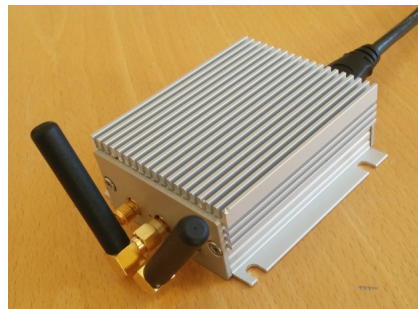
A Universal Software Radio Peripheral (USRP) is a device used by universities and research labs to prototype wireless communication systems. The B200mini is a USRP that can be programmed to operate over a wide radio-frequency range (70MHz - 6GHz) communicating in full duplex. For instance, it can be used in all of the GSM, UMTS, and LTE frequency bands. You can find more detailed technical specifications on the B200mini in Ref [30].

The B200mini board presents several inputs and outputs, some of them which you might find useful for setup or connectivity check:

- SW1 (Hard Reset Switch): Resets the USB controller or performs system reset. You might need to press it before performing connectivity tests or starting OpenBTS.
- PWR LED (Power Indicator): The LED of PWR shows orange when power is on.
- TRX LED (TX/RX Activity): You should use TRX for transmitting. The LED of TRX shows red when transmission is on. (Green for receiving, red for transmitting, orange when switching between transmitting and receiving.)
- RX2 LED (RX2 Activity): The LED of RX2 LED shows green when the receiver is on.



(a) Ettus Research B200mini [30].



(b) Enclosed USRP and antennas

Locate the connectors and LEDs on the device. Figure 10a shows the USRP B200mini board, while Figure 10b shows the hardware enclosed, as available to you for the lab, with antennas mounted. You should mount antennas at both TRX and RX to gain proper sensitivity at transmit and receive [8]. For more information about connectors and switches, see [32].

4.2.2 SIM Cards

Programmable SIM cards from Sysmocom will be used for testing purposes in some scenarios of the assignment [33]. Figure 11 shows the SIM cards available to you for the lab. The cards were programmed in advance with the necessary identifiers and keys. Refer to Appendix B for the values that are stored on the smart cards that have been assigned to your group.

4.2.3 OpenBTS.

OpenBTS is a Linux-based open source software developed by Range Networks that uses a software-defined radio [21]. The OpenBTS implements most of the GSM stack above the radio modem; however, it requires some other distinct applications as prerequisites:



Figure 11: Sysmocom programmable SIM cards.

```
wireless@wirelesslabtop: ~  
wireless@wirelesslabtop:~$ uhd_find_devices  
Linux; GNU C++ version 4.8.2; Boost_105400; UHD_003.009.001-release  
-- Loading firmware image: /usr/share/uhd/images/usrp_b200_fw.hex... done  
-- UHD Device 0  
-----  
Device Address:  
  type: b200  
  name: B200mini  
  serial: 30B0060  
  product: B200mini
```

Figure 12: UHD version and USRP B200mini correctly detected.

- SMQueue (SIP Message Queue) stores and forwards text messages, being a prerequisite for SMS delivery between mobile stations using OpenBTS.
- Asterisk is a Voice over IP (VoIP) switch that is responsible for call establishment between mobile stations using OpenBTS. Asterisk is an open source framework sponsored by Digium [6].
- SIPAuthServe (SIP Authorisation Server) is an application that manages the subscribers database, thus replacing the HLR (Home Location Register) found in a conventional GSM network.

For more information on OpenBTS and the applications, see [13, 22].

4.2.4 Other Tools

The Wireshark program can be used to inspect the GSM communication, similar to what you did in earlier stages of this lab assignment (see for example Subsection 1.5.5).

4.3 The GSM Network Setup

All the necessary software has been downloaded and built under `dev/BUILDS/<timestamp>`. A detailed step by step installation is given in [13]. Note that the B200mini requires the UHD (USRP Hardware Driver) version 3.9.0 or later [30] and special rules under Linux, being a USB connected device [31].

4.3.1 Connectivity Testing

Connect the USRP device to your computer. First, be sure that none of the processes are running, so run a script located in `dev` folder:

```
$ cd dev/  
$ sudo ./openbts-stop.sh
```

```
wireless@wirelesslabtop: /OpenBTS
wireless@wirelesslabtop:/OpenBTS$ sudo ./transceiver
linux; GNU C++ version 4.8.2; Boost_105400; UHD_003.009.001-release

Using internal clock reference
-- Detected Device: B200mini
-- Operating over USB 2.
-- Initialize CODEC control...
-- Initialize Radio control...
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Asking for clock rate 16.000000 MHz...
-- Actually got clock rate 16.000000 MHz.
-- Performing timer loopback test... pass
-- Setting master clock rate selection to 'automatic'.
-- Asking for clock rate 32.000000 MHz...
-- Actually got clock rate 32.000000 MHz.
-- Performing timer loopback test... pass
-- Asking for clock rate 26.000000 MHz...
-- Actually got clock rate 26.000000 MHz.
-- Performing timer loopback test... pass
```

Figure 13: Transceiver application communicates correctly with the B200mini.

```
wireless@wirelesslabtop: /OpenBTS
wireless@wirelesslabtop:/OpenBTS$ uhd_usrp_probe
linux; GNU C++ version 4.8.2; Boost_105400; UHD_003.009.001-release

-- Detected Device: B200mini
-- Operating over USB 2.
-- Initialize CODEC control...
-- Initialize Radio control...
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Asking for clock rate 16.000000 MHz...
-- Actually got clock rate 16.000000 MHz.
-- Performing timer loopback test... pass
-- Setting master clock rate selection to 'automatic'.

-----
Device: B-Series Device
-----
Mboard: B200mini
revision: 1
product: 30521
serial: 30B0060
name: B200mini
```

Figure 14: USRP B200mini technical configuration.

Run `$ uhd_find_devices` the connection to the USRP, as illustrated in Figure 12. If the device is not found then a USRP hardware reset might be required. Press the reset button on the B200mini board and try again.

Go to the `/OpenBTS` folder and run `$ sudo ./transceiver`. The hardware is correctly detected if it continues to run for some seconds while providing information about the device, as illustrated in Figure 13.

You can run `$ uhd_usrp_probe` to see the technical configuration. Figure 14 shows a successful attempt of this command.

Note that the OpenBTS process must *not* be running while executing the previous commands. However, when ready to proceed, you can start all the processes by:

```
$ cd /home/ttm4137/dev/
$ sudo ./openbts-start.sh
```

For more information about the start and stop scripts, see [17].

It might take some time for it to start, so wait until the LEDs turn on. Then, type `/OpenBTS/OpenBTSCLI` to start the OpenBTS Command Line program. Make use of the `help` command for an overview of the available commands (Figure 15). You need to use some of these commands later in the lab; for detailed information on OpenBTS commands, see the manual [22].

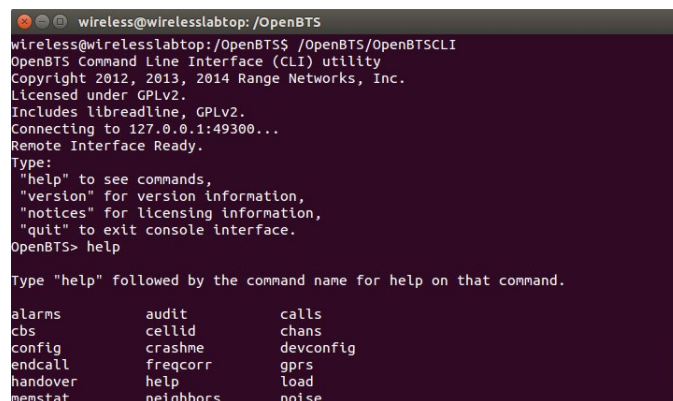
4.3.2 Network Setup

You should now be ready to set up your mobile network. Run the OpenBTS console command `config GSM.Radio` to see the default configuration. Note your group number, and set the frequency band operation of your USRP according to our allocation table in Appendix A. You can find more information about the physical uplink and downlink frequencies for each ARFCN in Ref. [37]. You can also set the MNC (Mobile Network Code) to your group number. This will avoid other students to connect to your network.

Configure your USRP for lab desk top minimum power radio operation. Reduce your radio interference with your neighbouring cells by using low radio power levels. Set the parameter `GSM.Radio.RxGain` to 0. Note that for this you must use `devconfig` instead of `config` (see Table 1). Make sure that the downlink transmission power is set to 0 (`power 0`), and check that the current noise level for your base station is acceptable (`noise`).

Run `config GSM.Identity` to check the default configuration. Name your network by setting one of these parameters. Most handsets will show the name of your network when registered to your network. You might even consider sending an SMS welcome message to your new mobile users.

Some of the configuration changes that you may want to do will require an OpenBTS process suite restart. This restart procedure can turn out to be necessary later in the lab assignment too. The command `audit` lists the changes made to the default configuration.



```
wireless@wirelesslabtop: /OpenBTS
wireless@wirelesslabtop:/OpenBTS$ /OpenBTS/OpenBTSCLI
OpenBTS Command Line Interface (CLI) utility
Copyright 2012, 2013, 2014 Range Networks, Inc.
Licensed under GPLv2.
Includes libreadline, GPLv2.
Connecting to 127.0.0.1:49300...
Remote Interface Ready.
Type:
"help" to see commands,
"version" for version information,
"notices" for licensing information,
"quit" to exit console interface.
OpenBTS> help

Type "help" followed by the command name for help on that command.

alarms      audit       calls
cbs         cellid     chans
config      crashme    devconfig
endcall     freqcorr   gprs
handover    help       load
memstat     neighbors  noise
```

Figure 15: OpenBTS CLI and available commands.

Command	Details
<code>config</code>	Searches for OpenBTS parameters and changes their value e.g.: <code>config GSM.Identity.ShortName myNetwork</code>
<code>devconfig</code>	Acts similar to <code>config</code> , but accepts more parameters (it runs at the developer level) e.g.: <code>devconfig GSM.Radio.RxGain 0</code>
<code>rawconfig</code>	Removes input validation (it allows to change a parameter to a value outside of the allowed range) and defines new parameter - value pairs e.g.: <code>rawconfig NewTestParam 1</code>
<code>unconfig</code>	Disables optional features e.g.: <code>unconfig Control.LUR.OpenRegistration</code>
<code>rmconfig</code>	Restores a parameter to its default value or removes a custom defined parameter e.g.: <code>rmconfig GSM.Identity.ShortName</code>

Table 1: Configuration commands in OpenBTS

Testing. Now, turn on your phones and try to select your base station by performing a manual search for mobile network operators. Identify your test network by the name you assigned to it . Some mobile phone models might encounter difficulties at this point. If this is the case, then you can try the airplane mode on and off, or even restart the phone. You will receive an error message because the authorisation fails.

Run the `tmsis` command to see the registration attempts, showing the IMSI, IMEI and other information. Use `tmsis -l` to display the detailed information stored in this registration database. Take a close look at the columns TMSI and AUTH, as you will need to refer to those later in the lab.

Ask one of the teaching assistants to check your configuration before moving on.

Take care! Some handsets will show the network as Test PLMN 1-1, even if you had previously set up a network name. If so, call `*#06#` to find the IMEI of your phone, then look for it in the table listed by the `tmsis` command (ignore the last digit, as in OpenBTS this is always 0) . You will so identify the corresponding IMSI, which you will later use for configurations. Later on, when you connect to a network, check the name of the operator displayed on your phone, and be sure to match the name you set up.

4.4 GSM Security Analysis

Now you will take a closer look at GSM authentication and access control, (un)linkability, and confidentiality mechanisms.

4.4.1 Capturing Traffic

First, you need to find out how to capture and store the network traffic. Verify that `Control.GSMTAP.TargetIP` targets to your machine (localhost default value 127.0.0.1 is fine), and enable `Control.GSMTAP.GSM`. Then, in a new terminal window, run the following command to store the traffic to a pcap file:

```
$ sudo tcpdump -i lo -n -s0 -w foreexample.pcap
```

You will have to analyse your captured communication. Use the tool Wireshark and identify messages belonging to distinct configurations, then explain your findings in the lab report. One approach can be to use different files for each of the main steps below. Most of the information you need is contained within the SIP (Session Initiation Protocol) messages. Hint: Wireshark filter.

4.4.2 Unlinkability

GSM, UMTS, and LTE mobile access networks issue random strings for temporary subscriber identification replacing the permanent identity string of IMSI. One purpose for this is to secure against a passive attacker (eavesdropper) that wants to link calls made or received by the same subscriber.

Enable TMSI allocation for your network. The command `config Control.LUR` will help you find the parameter you need to change. Refer to the OpenBTS manual if you need more help [22]. Think about what TMSI allocation is supposed to change in the message flow and remember to check this aspect later on, as you will refer to this in your lab report.

4.4.3 Access Control

The access control and authentication protocols of mobile networks are central topics in this wireless security course. Here we ask you to do some experiments with two access control mechanisms implemented in OpenBTS: Open Registration and Cached Authentication. Then, you will use the programmable SIM cards and experience with the full authentication in GSM.

Open Registration. Proceed now to enable Open Registration mode. This type of registration allow an IMSI that match a given regular expression to access the network. Take care, if there is no regular expression defined, then any handset will be allowed to register to the network! You can find more information about Open Registration in [13], Chapter 6.

Type `config OpenRegistration` to the OpenBTS console to list the parameters related to Open Registration. Notice that `Control.LUR.OpenRegistration` is disabled. Enable it by setting a regular expression. Simply set the exact IMSI to allow just one of your handsets to register.

```
config Control.LUR.OpenRegistration <IMSI>
```

Retry to connect your phone to your network. This time registration should work and, if the SMS service is enabled, you receive a welcome SMS message. Run the `tmsis` command and note the AUTH column. Inspect the traffic file and find the registration messages. Note the findings for your lab report.

Demonstrate your results to one of the teaching assistants.

Cached Authentication. In Subsection 4.3.2, handset registration to the network failed because the authentication was unsuccessful. In the previous paragraph, registration succeeded because Open Registration was enabled. However, Open Registration allows connectivity to the GSM network at a basic level, disregarding important functionalities (e.g.: calls between your mobile stations).

Disable Open Registration and proceed with 'Cached Authentication' mode. You will now manually authorise your phones to have access to your network and assign them MSISDNs (do not assign MSISDNs starting with 0, this might cause problems). For this, use `./nmcli.py` under the NodeManager folder (refer to [13] if you need more information):

```
$ cd dev/NodeManager
$ ./nmcli.py sipauthserve subscribers create <name> IMSI<IMSI> <MSISDN>
```

Other methods of phone configuration (provisioning) are possible. If you are able to use other methods please describe it in the lab report. Check your provisioning using the next command:

```
$ ./nmcli.py sipauthserve subscribers read
```

While you are experimenting with handsets provisioning, you might find it useful to truncate the TMSI table by the OpenBTS command `tmsis clear` and delete subscribers by running:

```
$ ./nmcli.py sipauthserve subscribers delete imsi IMSI<IMSI>
```

Register two mobile phones to your network. You should now be able to send SMS messages from one phone to the other and make calls (use the phone numbers you assigned for this). An interesting number you could call is 2600 (Echo call).

Run again the `tmsis -l` command and notice any differences in the TMSI table. Also, you might want to take a look at the K_i column, which is stored in the `sip_buddies` database table (see [22] for more details). Read about how Cached Authentication works [22], notice any differences from the Open Registration (both in the TMSI table and the authentication messages you have captured), and include your results in the final report.

Demonstrate calls and SMS functionality to one of the teaching assistants.

Full Authentication. You will further proceed to register the programmable SIM cards to the network. As a difference from the previous task, you now need to provision also the cryptographic key K_i when you manually authorise your phones to access your network:

```
$ cd dev/NodeManager
$ ./nmcli.py sipauthserve subscribers create <name> IMSI<IMSI> <MSISDN> <Ki>
```

Refer to Appendix B for the values that are stored on the smart cards that have been assigned to your group. Register your phones to the network. Notice any difference from 'Cached Authentication', when you have used commercial SIM cards and you did not provision the cryptographic key K_i . As before, use Wireshark to analyse the authentication handshake.

Confidentiality. Maintain the same settings and provisioning of the handsets from the 'Cached Authentication' and Full Authentication. Read more about encryption in the OpenBTS User Manual [22].

From the OpenBTS console, run `config GSM.Cipher` to inspect encryption related parameters in the OpenBTS implementation (Figure 16):

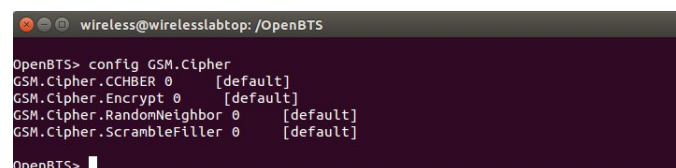
A screenshot of a terminal window titled 'wireless@wirelesslaptop: /OpenBTS'. The terminal shows the OpenBTS command-line interface. The user has entered the command 'config GSM.Cipher', which has resulted in a list of parameters and their current values: 'GSM.Cipher.CCHBER 0 [default]', 'GSM.Cipher.Encrypt 0 [default]', 'GSM.Cipher.RandomNeighbor 0 [default]', and 'GSM.Cipher.ScrambleFiller 0 [default]'. The prompt 'OpenBTS>' is visible at the bottom of the terminal.

Figure 16: OpenBTS encryption parameters.

Enable encryption by changing the value of `GSM.Cipher.Encrypt`, and then make calls between the handsets. Capture the traffic, and pay attention to differences between encryption disabled and enabled. Inspect the OpenBTS TMSI table and see if any modification took place. In particular, check any difference between when you are using your own SIM cards and when you are using the programmable SIM cards.

Demonstrate encrypted calls through your network to one of the teaching assistants.

4.5 Questions

You are expected to include answers to the following questions in your laboratory report.

Q19. Make a technical specification of the equipment you used and the mobile network you set up.

Q20. Note similarities and differences between the OpenBTS implementation you experimented with and the ideal textbook/lecture description with respect to the usage of the subscriber's temporary identity (TMSI).

Q21. Explain the technical differences between 'Open Registration' and 'Cached Authentication' based on the data in your access database ('TMSI-table') and the authentication messages that you captured. Exemplify.

Q22. Compare OpenBTS 'Cached Authentication' to the 'Full Authentication' and the textbook/lecture/standard description of the GSM authentication. What is required to perform the complete GSM authentication protocol and how is this achieved in the lab?

Q23. Compare the OpenBTS encryption behaviour to the GSM encryption standard. Describe the technical differences that you can find when OpenBTS encryption is disabled and when it is enabled, based on data in your 'TMSI-table', the key K_i , and the protocol messages that you captured. Refer to both scenarios: using your own SIM card, respectively using the programmable SIM cards. Exemplify.

Q24. Which security modes of operation are signalled to the user by the phone's display? In particular, consider various types of registration, authentication, and encryption modes.

Q25. How can this lab project be improved? (Answering is optional and does not influence your grade.)

Appendices

A ARFCN allocation per group

All groups use GSM 1900 band, which is commonly unused by telecom operators in Norway [39], but different ARFCN to avoid interferences [37]. Set `GSM.Radio.Band` and `GSM.Radio.CO` parameters to the values assigned to your group in Table 2, then ask a teaching assistant to check the configuration before moving on. **Take care!** Do not confuse with `GSM.Radio.ARFCNs`, which is the number of ARFCNs to use and should not be changed.

Group No.	Band (MHz)	ARFCN	Group No.	Band (MHz)	ARFCN
Group 1	1900	514	Group 16	1900	664
Group 2	1900	524	Group 17	1900	674
Group 3	1900	534	Group 18	1900	684
Group 4	1900	544	Group 19	1900	694
Group 5	1900	554	Group 20	1900	704
Group 6	1900	564	Group 21	1900	714
Group 7	1900	574	Group 22	1900	724
Group 8	1900	584	Group 23	1900	734
Group 9	1900	594	Group 24	1900	744
Group 10	1900	604	Group 25	1900	754
Group 11	1900	614	Group 26	1900	764
Group 12	1900	624	Group 27	1900	774
Group 13	1900	634	Group 28	1900	784
Group 14	1900	644	Group 29	1900	794
Group 15	1900	654	Group 30	1900	804

Table 2: Band and frequency allocation per group

B Programmable SIM Cards

The following table contains the IMSI and the cryptographic key K_i stored on the SIM cards. Identify the cards you are using (by the serial no. on the SIM card or by the IMSI) and use the corresponding key K_i when you create the subscribers.

IMSI	Serial No.	Ki
901700000011201	8988211000000112014	6FD60656E2B7F86FBEE9A31518AC18BE
901700000011202	8988211000000112022	1A45A8592402C304D902AC0FCBAB3764
901700000011203	8988211000000112030	582E25DAD1417308486D63C117582D09
901700000011204	8988211000000112048	E32FE9D554D1ECA97467EDBC86BEC1CE
901700000011205	8988211000000112055	8BE76D863C1B9D834C46FD37765E8D35
901700000011206	8988211000000112063	A5D710C0E40C5C5106AE534B2E475BA9
901700000011207	8988211000000112071	A383038397387EA1C025C83EE0D091D5
901700000011208	8988211000000112089	8F98DE4ADDCFA59F8656FEF0BA945217
901700000011209	8988211000000112097	CAC875F6C98518D0B5CFC62CDE0DF785
901700000011210	8988211000000112105	9F34841605A07C50B3CB696D7F9AA263
901700000012910	8988211000000120000	2B45C7C8FFE65DA8299C37A544017D69
901700000012911	8988211000000120000	1C5B79CBA1FF2CCAFDF9E6E4597B6FAC
901700000012912	8988211000000120000	AEC0233BAE21CFA9E26DC59AE67BAB45
901700000012913	8988211000000120000	EF226DA2F29085F8B432B0988F5E0658
901700000012914	8988211000000120000	D439269A3D1E97C590AD0F03FD7872EB
901700000012915	8988211000000120000	9AF25AE1ABCCF1DD73EEF382A3AD4EBA
901700000012916	8988211000000120000	D03538175F73F7DB27129A0CA736A73B
901700000012917	8988211000000120000	D6B9CCAF59A0B8244B7CE71D2D795007
901700000012918	8988211000000120000	41AD06A8C82817196A73DC2B68A13672
901700000012919	8988211000000120000	52B3A534A17E9FB3DE553273FA44A4D6
901700000012920	8988211000000120000	459CC90863BF7B9DB101930FDC519FEF
901700000012921	8988211000000120000	8A529AF0D63BD11B5899A92F6D50CFAB
901700000012922	8988211000000120000	BF0C3D9B66D1A5F52E683380036E7595
901700000012923	8988211000000120000	8602E2B90724E5E64F722F8B51B12313
901700000012924	8988211000000120000	AE6200293DA9820ECF0D7915F5570E5A
901700000012925	8988211000000120000	64A5D236684F3C3510E908FF529D1E19
901700000012926	8988211000000120000	39BB98596245D3F9B7BAD88FFB7FBDD9
901700000012927	8988211000000120000	9135D2F3700D24836249E76F71EAD93D
901700000012928	8988211000000120000	08E53B9EC852A9B132209C8AD645E75D
901700000012929	8988211000000120000	D1E073C6CB4C75A97D3E8D4A781559FD

Table 3: Values stored on the SIM cards

References

- [1] Aircrack-ng. Aircrack-ng suite. <http://www.aircrack-ng.org/doku.php#documentation>. (Cited in sections 1.5.4 and 1.5.4.)
- [2] Wi-Fi Alliance. The State of Wi-Fi Security. http://www.wi-fi.org/sites/default/files/uploads/files/wp_State_of_Wi-Fi_Security_20120125.pdf. (Cited in section 2.1.)
- [3] Hostapd IEEE 802.11 AP. IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. <http://hostap.epitest.fi/hostapd/>. (Cited in section 2.3.2.)
- [4] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in WEP's coffin. In *IEEE Symposium on Security and Privacy*, pages 386–400. IEEE Computer Society/IEEE Computer Society, 2006. (Cited in section 1.2.)
- [5] darkAudax. Tutorial: How to crack WPA/WPA2. http://www.aircrack-ng.org/doku.php?id=cracking_wpa, 2010. (Cited in section 2.4.1.)
- [6] Digium. Asterisk. <http://www.asterisk.org>. (Cited in section 4.2.3.)
- [7] John Edney and William A. Arbaugh. *Real 802.11 Security*. Addison Wesley, Reading, Massachusetts, 2003. (Cited in sections 2.1 and 3.2.)
- [8] Pulse Electronics. W1900 antenna. http://www.pulseelectronics.com/products/old_antennas/products__solutions/antennas_for_wireless_devices/wd_antennas/w1900/_w1902_penta_band_right_angle_stubby_antenna. (Cited in section 4.2.1.)
- [9] Ralf S. Engelschall. OpenSSL project. <http://www.openssl.org>. (Cited in section 3.3.1.)
- [10] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography, SAC '01*, pages 1–24, London, UK, 2001. Springer-Verlag. (Cited in section 1.2.)
- [11] Wireshark Foundation. Wireshark. <http://www.wireshark.org/>. (Cited in section 1.5.5.)
- [12] Changhua He and John C. Mitchell. *Security Analysis and Improvements for IEEE 802.11i*. In NDSS. The Internet Society, 2005. (Cited in section 8.)
- [13] Michael Iedema. *Getting started with OpenBTS*. O'Reilly Media, Inc., 2014. (Cited in sections 4.2.3, 4.3, 4.4.3, and 4.4.3.)

- [14] IEEE. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11-2012, 29 March 2012. (Cited in sections 0.2, 6, and 3.2.)
- [15] Kismet. <http://www.kismetwireless.net/>. (Cited in sections 1.5.2 and 1.5.2.)
- [16] Andreas Klein. Attacks on the RC4 stream cipher. *Des. Codes Cryptography*, 48(3):269–286, September 2008. (Cited in section 1.2.)
- [17] Nadiia Kotelnikova. OpenBTS systemd scripts. https://github.com/nadiia-kotelnikova/openbts_systemd_scripts. (Cited in section 4.3.1.)
- [18] Guillaume Lehenbre. Wi-Fi security — WEP, WPA and WPA2. http://www.hsc.fr/ressources/articles/hakin9_wifi/hakin9_wifi_EN.pdf. (Cited in section 2.1.)
- [19] Robert Moskowitz. Debunking the myth of SSID hiding. Technical report, ICSA Labs, 2003. (Cited in section 1.4.)
- [20] Robert Moskowitz. Weakness in passphrase choice in WPA interface. http://wifinetnews.com/archives/2003/11/weakness_in_passphrase_choice_in_wpa_interface.html, 2003. (Cited in section 2.1.)
- [21] Range Networks. OpenBTS. <http://openbts.org>. (Cited in section 4.2.3.)
- [22] Range Networks. OpenBTS Application Suite - User Manual. <http://openbts.org/site/wp-content/uploads/2014/07/OpenBTS-4.0-Manual.pdf>, 2014. (Cited in sections 4.2.3, 4.3.1, 4.4.2, 4.4.3, and 4.4.3.)
- [23] NTNU. Bibsys ask. <http://ask.bibsys.no>. (Cited in section 0.4.3.)
- [24] NTNU. The Norwegian University of Science and Technology, met.no buy new supercomputer. <http://www.ntnu.edu/news/new-supercomputer-to-be-installed>. (Cited in section 2.5.)
- [25] NTNU. Universitetsbiblioteket i Trondheim (UBiT) elektronisk søkeportal. <http://www.ntnu.no/ub/eubit/portal.php>. (Cited in section 0.4.3.)
- [26] Toshihiro Ohigashi and Masakatu Morii. A practical message falsification attack on WPA. http://n-pn.info/repo/Zenk-Security/Techniques_d.attaques___Failles/A%20Practical%20Message%20Falsification%20Attack%20on%20WPA.pdf. (Cited in section 0.3.4.)
- [27] OpenWall. English wordlists. <ftp://ftp.openwall.com/pub/wordlists/languages/English/>. (Cited in section 2.4.1.)
- [28] OpenWall. John the Ripper syntax and rules. <http://www.openwall.com/john/doc/RULES.shtml>. (Cited in section 2.4.2.)

- [29] Marcus Redivo. Creating and using SSL certificates. <http://www.eclectica.ca/howto/ssl-cert-howto.php>. (Cited in section 3.3.1.)
- [30] Ettus Research. USRP B200mini. <https://www.ettus.com/product/details/USRP-B200mini>. (Cited in sections 4.2.1, 10a, and 4.3.)
- [31] Ettus Research. USRP hardware driver and USRP manual - LibUSB. http://files.ettus.com/manual/page_transport.html#transport_usb. (Cited in section 4.3.)
- [32] Ettus Research. USRP hardware driver and USRP manual - USRP B2x0 Series. http://files.ettus.com/manual/page_usrp_b200.html. (Cited in section 4.2.1.)
- [33] Sysmocom. Programmable SIM Cards. <http://shop.sysmocom.de/products/symousim-sjs1-4ff>. (Cited in section 4.2.2.)
- [34] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In *Proceedings of the 8th international conference on Information security applications*, WISA'07, pages 188–202, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited in section 1.2.)
- [35] Yosuke Todo, Yuki Ozawa, Toshihiro Ohigashi, and Masakatu Morii. Falsification attacks against WPA-TKIP in a realistic environment. *IEICE Transactions*, 95-D(2):588–595, 2012. (Cited in section 0.3.4.)
- [36] Aaron Weiss. Introduction to Kismet. <http://www.wi-fiplanet.com/tutorials/article.php/3595531>, 2006. (Cited in section 1.5.2.)
- [37] Wikipedia. Absolute radio-frequency channel number. https://en.wikipedia.org/wiki/Absolute_radio-frequency_channel_number. (Cited in sections 4.3.2 and A.)
- [38] Wikipedia. Citation. <http://en.wikipedia.org/wiki/Citation>. (Cited in section 0.4.3.)
- [39] World Time Zone. GSM world coverage map and GSM country list. <http://www.worldtimezone.com/gsm.html>. (Cited in section A.)