

BACHELOR THESIS
COMPUTING SCIENCE



RADBOUD UNIVERSITY

Developing ELVis:
A 4G network analysis and visualisation tool

Author:
Thomas Luijkman
s1039468

First supervisor/assessor:
PhD, Katharina Kohls
kkohls@cs.ru.nl

Second assessor:
Associate professor, Erik Poll
erikpoll@cs.ru.nl

January 17, 2022

Abstract

Mobile networks are a fairly specialised field of study in computing science. However, with billions of people using mobile networks every single day, it is important for the network to behave securely and correct. To analyse traffic on a mobile network is no simple task.

This report documents the building of an analysis and visualiser of 4G/LTE network traffic. The purpose of such a tool is to give individuals more insight into the traffic flowing in and out of their mobile phone. The resulting product might be useful for developers interested in building on a mobile network. However, for simply interested individuals, the product still requires too much knowledge of mobile networks and general network security.

Contents

1	Introduction	4
2	Technical background	7
2.1	Components of a 4G network	7
2.1.1	User Equipment	8
2.1.2	Evolved Node B	9
2.1.3	Evolved Packet Core	9
2.2	The 4G protocol stack	10
2.3	Connection establishment	12
2.3.1	MIB and SIB messages	13
2.3.2	RRC connection establishment	13
2.3.3	Initial attach	14
3	ELVis concept and framework	17
3.1	The <code>Packet</code> class	19
3.2	Parsing the data	19
3.3	Control flow	20
3.4	Visualising packet flow	20
4	Capture analysis	23
4.1	Experimental setup	24
4.2	Analysing the <code>SecurityModeCommand</code>	25
4.3	Analysing the identity request	26
4.4	Analysing the authentication process	26
4.5	PDCP MAC invalidation	27
5	Discussion	29
5.1	Limitations	29
5.1.1	Limitations on parsing	29
5.1.2	Limitations on visualising	30
5.1.3	Analysis limitations	30
5.1.4	Future analysis opportunities	31
5.2	Use cases	32
5.2.1	Developers and Mobile Network Operators	32

5.2.2	Interested individuals	33
6	Conclusions	35
A	LTE authentication	39
A.1	The MILENAGE protocol	39
A.2	The XOR protocol	41
B	Code snippets	42
C	Patching srsRAN	45
C.1	Invalidating the PDCP MAC	45
C.2	Patching authentication	46

Author's note

This thesis report greatly concerns the development of a piece of software. While the source code for this piece of software should be contained in the same folder as this report, or packaged in the same `.zip` file. If this is not the case, the source code will always be publicly available at the following GitHub page: <https://www.github.com/thomasluijkman/4Gvisualiser>¹

¹Note that this URL links to the latest release of the software. In case the software was changed over time, and you want to see the state of the software as it was when this report was completed, the following link can be used: <https://github.com/thomasluijkman/4Gvisualiser/tree/835fca65b34288b562546ffa7ae731c166eb602>

Chapter 1

Introduction

Although the transition to the fifth generation of mobile networks (also known as 5G) is well underway, the usage of the 4G/LTE mobile network is far from over. Projections show that in 2026, nearly a decade after the inception of 5G, there will still be over four billion devices using the previous generation.[26]

However, tools that perform security analysis on this specific type of communication are difficult to come by. There are applications like Wireshark, which accept PCAP files and show general data based on all of the packets present in the file. This data includes information about which protocols are being used, what state these protocols are in and between which devices the packet is being sent, to name a few. Research has also been done to use machine learning to find potential security flaws in official specifications.[7] This thesis documents the development of a tool, which parses, visualises and analyses the correctness and security of communication on 4G networks.² The aim of the development of such a tool is to advance the accessibility of 4G communication

This makes it difficult to perform adequate security analysis on 4G networks, as it would entail going over each individual packet in the file and checking if the data complies with the exact way it was designed. The existing specifications are also not nearly accessible enough to be read by your average developer; every protocol has multiple documents consisting of hundreds of pages, all of them needing deep background knowledge of mobile networks.

The problem with Wireshark and other tools like it, is that they do not perform any kind of in-depth analysis concerning the security and correctness of the files that they are being fed. For example, a packet with an incorrect sequence number will still appear as a regular packet in Wireshark's interface, instead of showing that there is something wrong with this

²For convenience's sake, this tool will be referred to as a "(4G network) visualiser" or its name, "ELVis", for the remainder of this report.

packet.³

This is not to say that security analysis has not been performed on 4G/LTE networks. Rupprecht *et al.* used their own testing framework, mimicking a real LTE setup to try and find flaws in the authentication and encryption of traffic on the mobile network.[23] They found that it is still possible to send unencrypted traffic over 4G/LTE networks, thus allowing for attacks on the confidentiality of the network. This is not allowed according to the official 3GPP specifications.[2] The user is also never informed whenever traffic is sent and received unencrypted. Chlosta *et al.* found that there are still commercial providers in several countries allowing communication without security functions enabled – again going against official specifications.[8]

This tells us a few things about the state of 4G/LTE security. First, it shows that there are still flaws in the security of this mobile network. Security analysis is therefore still very much necessary. It also shows that the average user does not have a lot of insight into the way their mobile device connects to a 4G network. Nowadays, most web traffic is encrypted by HTTPS standards, but this is of course far from the only network traffic. Since encryption settings for mobile networks happen “under the hood”, users will not know that their traffic could be read by a smart attacker. And if the wrong data is read, privacy leakage can occur.[14]

This thesis documents the development of a tool, which parses, visualises and analyses the correctness and security of communication on 4G networks.⁴ The aim of the development of such a tool is to advance the accessibility of 4G communication analysis for developers and researchers alike. It also provides a reflection on the finished product and possible further amendments that can be made.

Note that analysing all aspects of 4G traffic is out of scope for a thesis project. That is why the finished product will not be analysing all aspects of 4G/LTE networks. The visualisation and analysis will be limited to connection establishment, from the moment the mobile device starts its connection with a base station until the connection establishment is complete, and the device is ready to perform actual communication over the mobile network. It is also a practical analysis, taking a packet capture from 4G/LTE communication and checking if everything is going according to specification, instead of theoretical analysis of the security of protocols done by others.[12, 14, 25, 29]

Chapter 2 will go over the protocols necessary to understand what happens when a mobile device tries to connect with a 4G/LTE network. Chapter

³Note that this only holds for the 4G/LTE configuration. When viewing more well-known attacks on regular network security, such as ARP-spoofing, does get picked up by Wireshark and displayed differently than a normal ARP packet.

⁴For convenience’s sake, this tool will be referred to as a “(4G network) visualiser” or its name, “ELVis”, for the remainder of this report.

3 will explain the framework on which the analysis component of our tool is built, before diving deeper into the analysis itself in chapter 4. There will be a discussion on the limitations of the tool and in chapter 5.1 and possible use cases of the tool in chapter 5.2.

Chapter 2

Technical background

The 4G network has a layered network protocol stack, similar to how wired networks operate. To understand the development process of our visualiser, we will first give a quick overview of the different protocols on the mobile network, before moving on to how these protocols interact with each other in the context of connection establishment.⁵

2.1 Components of a 4G network

First it is important to explain some of the terminology used when working with mobile networks. In this section, we will briefly explain all these components and how they interact with each other in a mobile network.[10]

A brief overview of the 4G network architecture can be found in figure 2.1. The grey boxes represent the major components in the 4G network architecture: these are the UE (mobile device), eNodeB (radio tower) and EPC (core network). The mobile device has a direct (physical) connection to the radio tower. This radio tower forwards all communication to the core network, and this core network handles the communication depending on the type of communication.

There are two “planes” in which messages between a mobile device and the core network can fall. There is the user plane, which is taken care of by the blue components of the EPC in figure 2.1. Messages on the user plane (also referred to as the data plane in literature) will be forwarded by the core network to external networks, such as the internet or from the internet to the mobile device. The other plane is the control plane, denoted by the red components of the EPC in figure 2.1. The control plane takes care of the connection between the mobile device and the core network. Connection establishment, authentication and security handling are all part of the control plane’s responsibilities.

⁵Note that concepts like “cells” and “channels”, while mentioned in passing in this report, will not be thoroughly explained here, as they are too low-level to be relevant for this project.

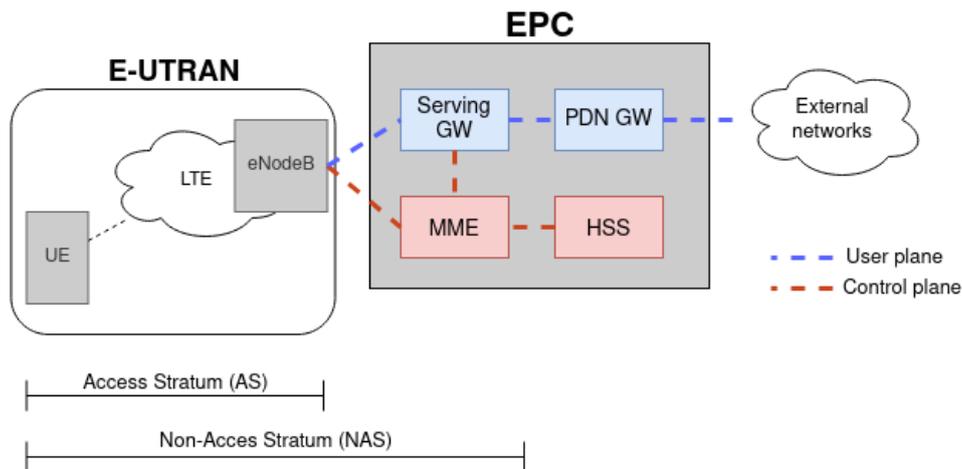


Figure 2.1: The 4G network architecture[13]

This research focuses on connection establishment for a mobile device to a mobile network. This means that, for this research, we will focus on messages sent over the control plane. An overview for components on the user plane will also be provided, however, to give a complete picture of how the 4G components interact with each other.

At the bottom of the image there are two more ways to differentiate between communication: there is the Access Stratum and the Non-Access Stratum. The AS refers to communication between the mobile device and the radio tower. The NAS refers to communication between the mobile device and the MME, a component in the core network on the control plane.⁶ This is an important distinction as AS and NAS can have different configurations with regards to security, for example.

2.1.1 User Equipment

The User Equipment (commonly abbreviated as UE) refers to any mobile device that sends or receives data via a mobile network. The handling of the mobile network connection happens in two separate parts of the UE: there is the application processor, on which the operating system and all applications are run, and there is the baseband processor, which handles the connection to the mobile network. The baseband processor receives data to send from the application processor, brings it through the protocol stack and sends it to the base station via radio waves. It also receives packets via radio waves and transmits these to the application processor.

All UE devices have a SIM-card, which, in LTE standards, is referred to as the Universal Integrated Circuit Card, or UICC. This SIM-card contains

⁶Note that the term “NAS” both refers to the connection between a mobile device and the MME, and the protocol and messages that are sent over this connection.

a International Mobile Subscriber Identity (IMSI) value, which is used by Internet Service Providers to see if you are part of their network. There is also an International Mobile Equipment Identifier (IMEI), which is used to identify the device that is connecting to the mobile network.

These values are both unique to all individual users and devices, and are crucial to keep hidden. If someone were to perform a man-in-the-middle attack, they might be able to get accurate location data of subscribers who use location-based services. If an IMSI were attached to each message that was sent, the man in the middle might be able to know exactly who is exactly where at any given time.

To protect the confidentiality of a mobile network user, each mobile device is assigned a Global Unique Temporary Identifier (GUTI). This GUTI is unique to each mobile device and can be used to identify a user to the mobile network as to keep the IMSI hidden. It consists of the GUMMEI, which identifies the core network that assigned the GUTI, and a random temporary value to identify the device.

2.1.2 Evolved Node B

The Evolved Node B is the base station the UE connects to, and is often abbreviated as eNodeB or eNB. Once connection from a UE to a eNB has been established, the eNB serves as a middle-man sending packets from the UE to the core network and vice versa.

The base stations also communicate with each other, in order to let mobile devices change location without losing the network connection. The network of eNodeB's communicating with each other is referred to as E-UTRAN. This name is also used to refer to the connection between UE and eNB. The research will be focused on traffic on E-UTRAN. Traffic from the mobile device to the base station is often referred to as the uplink, and traffic from the eNodeB to the UE is called the downlink.

2.1.3 Evolved Packet Core

The Evolved Packet Core (commonly abbreviated as EPC) is the core network of 4G/LTE.[9] It consists of several components:

- **Mobility Management Entity (MME):** The MME handles a lot of the control of the connection between the UE and the EPC. This is the part of the EPC the UE talks to during the initial attach procedure⁷. The MME also handles the handover procedure when the UE needs to connect to a different base station.[6]
- **Home Subscriber Server (HSS):** The HSS has stored data of the

⁷This procedure will be explained in section 2.3.3

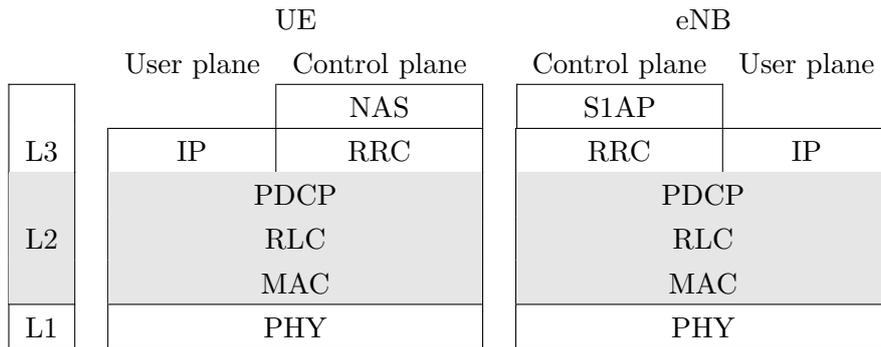


Figure 2.2: Protocol stack of the baseband processor

subscriber, including the IMSI and cryptographic keys when communication is encrypted.

- **Serving Gateway (S-GW):** The Serving Gateway routes the IP packets from the UE to external networks and vice versa.
- **Packet Gateway (P-GW or PDN GW):** The Packet Gateway (also known as the Packet Data Network gateway) provides the connection to external, non-mobile networks. It routes packets received from the S-GW to external networks and vice-versa. During the initial attach, the P-GW allocates IP-addresses. Although in literature, the P-GW and S-GW are often described as two separate entities, they are sometimes combined in practice.[13]

2.2 The 4G protocol stack

4G also has a layered network protocol stack, similar to networks like Ethernet. To illustrate how the network flow differs from one of these networks, we will explain how a packet gets built and transmitted from a UE device and what the eNB does with it.

Similar to Ethernet, the process of a UE and a eNodeB communicating can be seen as a message travelling through five “layers”, where each layer houses different protocols.⁸ The top layer consists of application protocols, like a computer communicating with an e-mail server. The fourth layer (also known as the transport layer) is used to provide reliable data transfer between the endpoints of communication. An example of a fourth layer protocol is TCP.

The third layer is the network layer and is used for the routing and addressing of points on a network. The Internet Protocol (IP) is a well-

⁸Officially, the OSI-model that is most commonly used in network architecture consists of seven layers[21], but to prevent needless complication the top three layers will all fall under the umbrella term: ‘application layer’.

known protocol on the third layer. The second layer provides reliable data transfer between two points in a route. The first layer is the physical layer and are the actual bits moving over a wire or air interface between two points in a network.

The fifth and fourth layer are handled by the application processor and are the exact same for LTE, compared to Ethernet networks. On the third layer, the application processor hands over the data prepared by the upper layers to the baseband processor, which handles the third, second and first layer of 4G communication. This is where the network protocols used differ from the more commonly known Ethernet variants. A visual representation of the protocols implemented on a baseband processor can be seen in figure 2.2. The baseband processor takes the data handed over by the application processor through all of the protocols in the bottom layers, until it arrives at the first layer (the physical layer), where it will be transmitted to the base station via radio waves and travel back up the protocol stack here. At the third layer, the packet is handed over to the EPC, where it goes through the S-GW and PDN-GW to the external networks.

Below is an explanation of the protocols that are used by the baseband processor[19]:

- **Non Access Stratum (NAS):** Provides security, authentication and authorisation for the UE device. This is the main protocol used for communication between the UE and MME.
- **S1 Application Protocol (S1AP):** Protocol on the control plane used for communication between the eNodeB and EPC. The eNodeB talks to the EPC via the MME on the control plane, and S1AP is used for this purpose.⁹
- **Radio Resource Control (RRC):** Manages connection on the control plane between the UE and eNodeB. Also manages ciphering and integrity protection. RRC also encapsulates NAS messages.
- **Packet Data Convergence Protocol (PDCP):** Receives the data ready for transmitting from the IP protocol, and data for ciphering and integrity protection from the RRC protocol. Also performs tasks like duplicate control and in-sequence delivery.
- **Radio Link Control (RLC):** Sends the PDCP data in three different modes:

1. *Transparent Mode (TM):* PDCP data simply passes through. Nothing is done to check correctness of the data.

⁹This protocol is not part of the E-UTRAN protocol stack as it does not concern the UE at all. It is still included in this explanation for completeness' sake, however.

2. *Unacknowledged Mode (UM)*: PDCP data can be altered and re-ordered when necessary. There is also duplicate detection. However, no acknowledgement is sent back.
 3. *Acknowledged Mode (AM)*: The addition of acknowledged data allows for retransmission of data, so that missing segments can get sent again, leading to better integrity of the data.
- **Medium Access Control (MAC)**: Responsible for multiplexing the data to and from the PHY layer. Also responsible for scheduling within UE devices (at the UE's end) or between UE devices (at the eNB's end).
 - **Physical layer (PHY)**: Transmits data received from the MAC protocol over the air through radio waves.

Note that these are very general explanations of the protocols. All of these protocols are part of the connection establishment process which will be analysed, thus it is good to have an overview of the structure of a 4G network and its protocol stack before moving on to the topic of connection establishment of a UE device.

2.3 Connection establishment

This section will dive deeper into how these protocols interact with each other when the UE is connecting to an LTE network how the messages passed between UE and eNB are structured. Since this thesis focuses on an analysis of the correctness and security of the connection establishment on a 4G/LTE network, it will be important to understand the process of establishing a connection. There are three major phases taking place here: eNodeB selection, the connection establishment with RRC and the initial attach.

The eNB selection happens using broadcast messages containing information about the base station. The UE uses these messages to then select a base station to connect to. The RRC protocol is used to establish a connection between the UE and a base station. If and only if the RRC connection establishment procedure has succeeded, the UE will go through the attach procedure, where it authenticates itself to the EPC via the NAS protocol. After the attach has been accepted, the UE is officially connected to the LTE network. In this chapter we will provide a detailed explanation of the different exchanges that happen between all components of a 4G network for connection establishment to happen.

2.3.1 MIB and SIB messages

There is often more than one base station to connect to for a given UE device. To inform the mobile device about a possible connection point, the eNodeB periodically sends out messages informing the UE.[12] There are two types of these messages: the Master Information Block message (MIB) and System Information Block messages (SIB).

The MIB contains important information about the bandwidth possible on that eNodeB, and the System Frame Number.[24] This is a clock value which resets approximately every 10 seconds which the UE needs to synchronise to for communication.¹⁰ This message is generated with up-to-date information every 40 milliseconds, and retransmitted every 10 milliseconds.

There are 11 SIB messages, but for connection establishment only the first two are relevant. SIB1 contains information about the identity of the network, such as the Mobile Network Code and Mobile Country Code and the specific cell a UE can connect to. It also contains scheduling info for other SIB messages the UE should want.

SIB2 contains configuration for RRC and the random access procedure that follows when the UE has selected a base station to connect to. It also contains configuration values for specific channels over which communication can occur. After this message the UE can select a base station to connect to and perform the random access procedure, used to synchronise the UE and eNodeB, after which the rest of connection establishment can take place.

2.3.2 RRC connection establishment

The RRC connection establishment procedure is defined in the official 3GPP¹¹ specifications.[4] It starts with the sending of a **RRC Connection Request** message. This message is sent on the uplink, and contains the reason for establishing a connection and the identity of the UE. The identity value can either be a large random value or a SAE-Temporary Mobile Station Identifier (S-TMSI). This identifier is sometimes used to protect the confidentiality of the IMSI.

The UE then waits for the reception of a **RRC Connection Setup** message. With this message, it receives some configuration variables from the eNB for RRC and other lower-layer protocols. It will perform configuration based on these values, after which the protocol will enter a connected state, which means the connection is complete. The UE will set up one more message called **RRC Connection Setup Complete** with some more values received from upper layers. If there is already a registered MME the application processor wishes to connect to, this value is also included in the

¹⁰LTE has also introduced the HFN or Hyper Frame Number, which increases every time the SFN resets. This is not relevant to connection establishment, however.

¹¹Third Generation Partnership Project, the organisation behind all the 3G, 4G and 5G specifications.

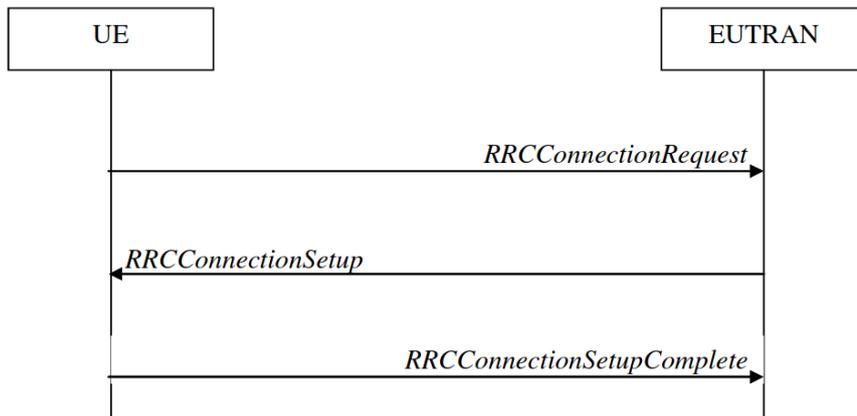


Figure 2.3: Successful RRC connection establishment procedure[4]

message. The protocol now enters the RRC_CONNECTED state, and the initial attach can begin.

If no RRC Connection Setup message has been received after a certain time, protocol configurations will be reset and upper layers will be informed of the fact that connection could not be established.

If the connection is rejected by the base station for whatever reason, the mobile device will receive a RRC Connection Reject message instead of a RRC Connection Setup message. If this occurs, the UE will reset configuration and let upper layers know that connection could not be established. Connection is barred for a while, until a timer has expired, after which upper layers can try to connect again.

2.3.3 Initial attach

Although the RRC connection setup and initial attach procedure are separated in this thesis, they do have some overlap in the RRC Connection Setup Complete message. This message sends the “attach request”, which sets the attach procedure in motion. The attach procedure is handled by the NAS protocol, as described in the 3GPP specifications.[1] With the attach request, the UE sends along information about itself, security functions it can use (cyphering and integrity functions, for example) locational data such as the mobile country code, and other, more specific values that are not important at this point.

There are a few different message exchanges happening during the initial attach, as can be seen in figure 2.4. This figure shows the most important exchanges, but not all of them. Other important exchanges that are not shown here are the identity request and the UE capability information. In the following subsections, each of these exchanges will be explained in more detail.

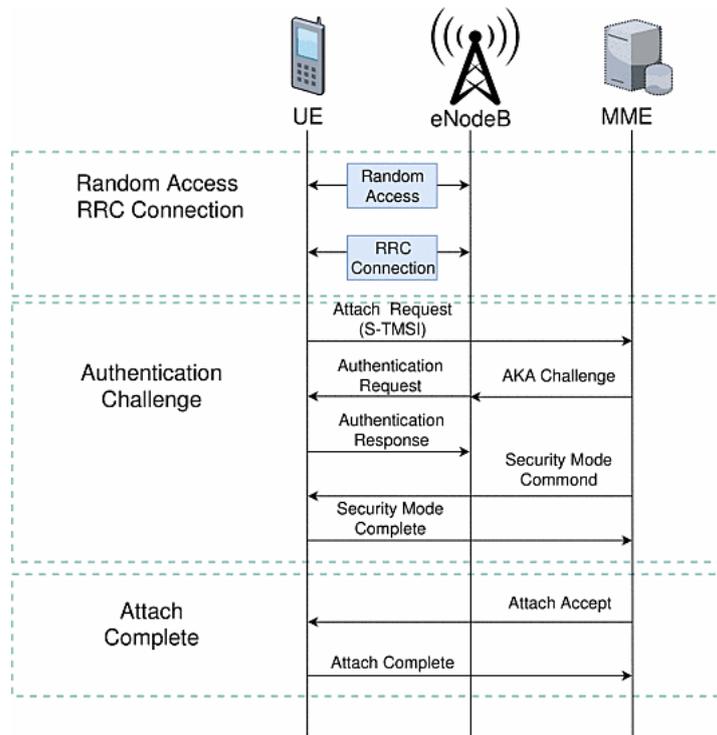


Figure 2.4: Successful connection establishment[12]

Identity request

The identity request is the first exchange to happen after the attach request is sent. This message is sent from the MME to the UE to find if the UE is “subscribed” to this mobile network. It requests the IMSI of the UE, which the UE delivers in the identity response message. This is the only time the IMSI is sent over the mobile network.

It is important to note that the identity request does not happen all the time. The core network will only ever send the identity request if the GUTI or S-TMSI sent to the network are not known to the network.

Authentication

After the identity of the connecting UE has been established, the authentication procedure happens. This happens by an authentication request and response message. The EPC uses the IMSI provided by the UE to generate an authentication value (AUTN) and a random nonce value (RAND) to be used as input for a cryptographic function, together with a secret key K which both the EPC and UE should have knowledge of. With the EPC, the key is stored in the HSS and with the UE, it is stored in the SIM card. Both

parties use the same cryptographic function to calculate a specific value.¹² This cryptographic function also calculates other keys used for ciphering and integrity protection later. The UE sends its results (RES) to the EPC, and if the result matches the expected result (XRES), the authentication is complete.[17]

Authentication does not always succeed, of course. If RES is not equal to XRES, there will not be a successful authentication, and the EPC will send an authentication reject message. Upon the reception of this message, the authentication will fail and the SIM will be considered invalid until the UE is switched off. The next time the UE is switched on, it can attempt another authentication. If the MAC provided in the request is not accepted, the UE will send an authentication failure message.

Security options

After authentication comes the security mode command. This is done by the EPC sending a message to the UE via the NAS protocol, replaying the security options the UE provided in the attach request, and providing the EPC's choice of cyphering and integrity functions. The UE sends a message back confirming the NAS choice of functions. The same is then done with the RRC protocol for the security of the E-UTRAN connection.

Capability information

The final phase of the attach procedure is the UE capability information. The EPC first asks the UE for its capabilities, which the UE provides. This is the largest message sent in the connection establishment procedure, being over 1600 bytes of data.

After this message, there is one final `RRCConnectionReconfiguration` which contains similar data as the `RRCConnectionSetup` message but with some parameters changed to reflect the parameters provided by the UE capability information. If this procedure goes well, the connection reconfiguration also contains the NAS message saying that the attach is accepted. The UE confirms this by sending a message that the attach is complete, and from this point on, data can be transferred over the mobile network.

¹²For those interested, a summary of the LTE Authentication and Key Agreement functions is provided in appendix A.

Chapter 3

ELVis concept and framework

Now that all the background knowledge is provided, it is time to start discussion on the 4G network visualiser and analyser as described in chapter 1. This chapter describes the functionality of the tool and the framework on which the analysis component is built.

The 4G visualiser will be named ELVis, an acronym which stands for E-UTRAN LTE Visualiser. The name was chosen since the security analysis is mainly focused on E-UTRAN. Currently, analysis visualisation is only focused on E-UTRAN communication. Protocols like S1AP are not compatible with the visualiser in its current state, since it relies on the presence of a MAC layer in every packet present in the capture. For reference, ELVis was constantly tested using the `enb.pcap` and `ue_mac.pcap` files generated by srsRAN, and works using these packet captures.

ELVis can be seen as three distinct components working together to make a full application. There is the control segment, which parses the command line arguments before parsing the packet capture by taking data from a packet capture and stores it in a custom-defined `Packet` class. The visualiser takes parsed data and displays it, also taking analysis results into account. The analyser is the most involved component of the three, which is why the discussion on this piece of code has been placed into its own chapter.

Figure 3.1 shows the general program structure of ELVis. Everything in the grey box is included in the source code, everything outside should be installed prior to running the program. As is visible in the diagram, the program starts off by parsing the command line arguments before starting to parse the packet capture file. After this has succeeded, the program will analyse and visualise the data.

By default it does both – first analysing the data and then visualising it, using the analysis results to add more details and colours to the visualisation. However, the user can choose to do only one of the two options via the

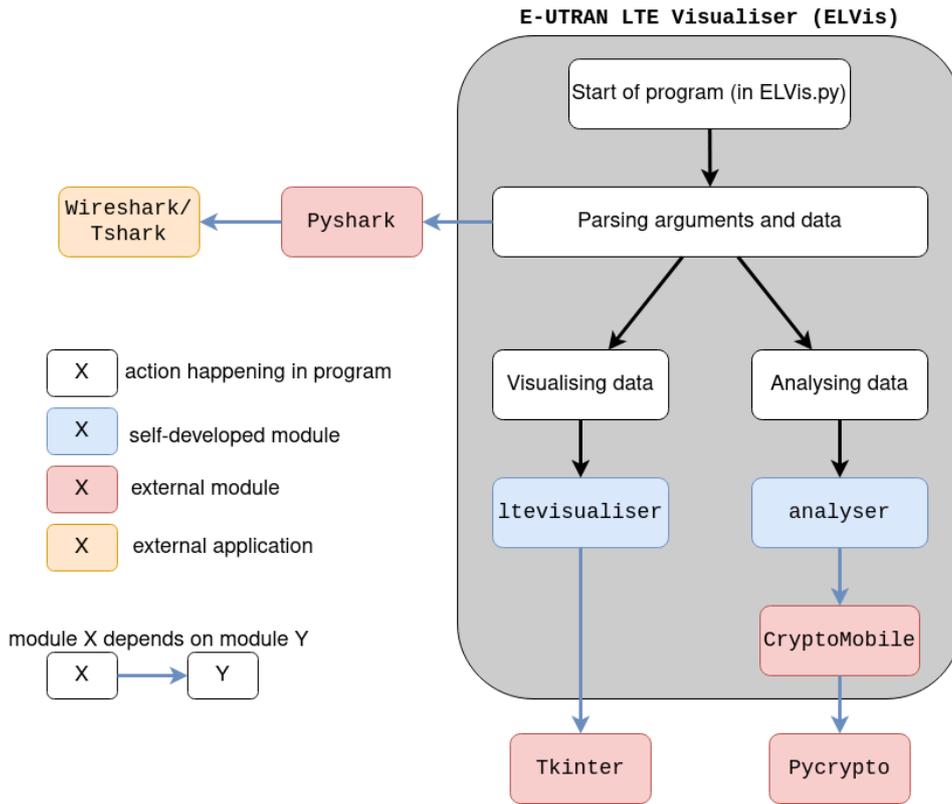


Figure 3.1: Program structure of ELVis

command line. If it only visualises, it will skip all analysis and provide the UI. If it only analyses, the results of analysis will be shown in the terminal from which ELVis is run.

While this report will contain the most important bits of the source code, which can be found in appendix B, it might be nice to take a look at the source code as a whole, to see how different parts of the program might interact. As explained in the author’s note, the source code should be provided in the same `.zip` file as this report. However, if the source code is absent, for whatever reason, it can always be found at the following GitHub page: <https://www.github.com/thomasluijkman/4Gvisualiser>. The `readme.md` file contains instructions on how to correctly install dependencies for the program.

As is visible in both the repository and listings, ELVis was developed entirely in Python on Ubuntu 20.04. The program should also work on older versions of Ubuntu and might also work on Windows, but testing has only been performed on Ubuntu 20.04, so no guarantees can be given on correct workings on other platforms.

3.1 The Packet class

Our `Packet` class mostly consists of a bunch of variables bundled together, to keep all the data stored neatly and make sure that everything that is needed for visualisation and analysis is easily accessible. Important attributes are `Packet.data`, which is the parsed data, `Packet.summary`, which is the one-line summary used in the visualiser, and `Packet.category`, a list of all phases of connection establishment that a packet is involved in. There is also `Packet.analysis`, which is a string showing all errors and warnings gathered by the analysis, and `Packet.eval`, a numerical score of the “wrongness” of a packet that is used to generate the colour coding in the visualiser.

Mostly it is used in the same way you might use a `struct` in C++, however there are a few auxiliary functions present, such as the function `get_colour`, which returns an RGB colour to use for visualisation, where a red packet means there are severe errors, and a yellow packet means there are only small errors or warnings. There is also `add_analysis()` which appends a sentence to `Packet.analysis` and updates the evaluation score. The `process_summary` packet is used to make sure a packet’s `summary` does not exceed a maximal character limit for visualisation.

3.2 Parsing the data

The first piece of code that was written concerned the parsing of a `.pcap` file into Python. The files used for this purpose in the source code are `main.py` and `packet.py`. The library responsible for parsing packet captures into Python is called Pyshark, and describes itself as a “Python wrapper for tshark, allowing python packet parsing using wireshark dissectors.”[15]

So, while this does require the user to have installed Wireshark and tshark, a terminal-based implementation of Wireshark, it is probably the simplest library that does exactly what we need. Alternatives that were considered were `dpkt`¹³ and `Scapy`¹⁴. However, while both of these libraries had a lot of recommendations online concerning packet parsing in Python, they both did not allow the parsing of LTE packets, which made it useless for us.

Wireshark, however, also does not parse LTE packets by default. For that, a specific user profile must be created.¹⁵ After the creation of this configuration profile, the packet capture can be loaded into Python quite easily.

¹³See: <https://github.com/kbandla/dpkt>

¹⁴See: <https://scapy.net>

¹⁵For details on parsing LTE packets in Wireshark, refer to page 16 of the srsRAN documentation.[28]

It is not this easy in our program, however. We also want to store the one-line summary you see in the user interface of Wireshark. There is also a field called `mac-lte`, which we can not access, since Python does not parse expressions like `'packet.mac-lte.direction'` correctly.¹⁶ We also want to store our data in our defined `Packet` class. All these restrictions make the parsing a bit more involved, which can be shown by comparing listing B.1, which is the standard way of reading a file into Python using Pyshark, and listing B.2, which is the current implementation of our parsing function.

3.3 Control flow

The user has a few options to choose from. The specifics of all the options can be found in the `readme.md` file, in the same place as the source code. The command line arguments are all handled by `main.py`. Since ELV is a terminal-based application, the user can provide command line arguments to change the behaviour of the program. Arguments are read into the program via the `sys.argv` list. The function to parse command line arguments is fairly long, but listing B.3 can be looked at for the general idea.

Command line arguments are put into one of two Python dictionaries: `options` and `parse_options`. The first one concerns general options, such as the `analyse-only` and `visualise-only` options. The second one contains parse-specific options. These options are translated to Tshark arguments and are fed directly into the `FileCapture` command in listing B.2.

3.4 Visualising packet flow

The next step in the development process was visualising data. We first looked into using the image processing features of OpenCV,¹⁷ but ultimately we chose for Tkinter¹⁸, a module for creating images and a graphical user interface in the Python standard library. This module is the foundation of the visualisation of the packets parsed using the script described in section 3.2. The visualisation is all handled by the `ltevisualiser` module.

The visualiser starts by creating a window and adding two vertical lines. These lines represent the UE and eNodeB endpoints. After this, it processes the packets to be shown on screen. There can be five packets shown on the screen at any point in time – at this point there is no variable screen size. These five packets have their summary shown above an arrow. The arrow is directed to the destination of the message. The user interface can be seen in figure 3.2.

¹⁶The IDE immediately raises a red flag: “Unresolved reference: `lte`”

¹⁷See: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

¹⁸See: <https://docs.python.org/3/library/tkinter.html>

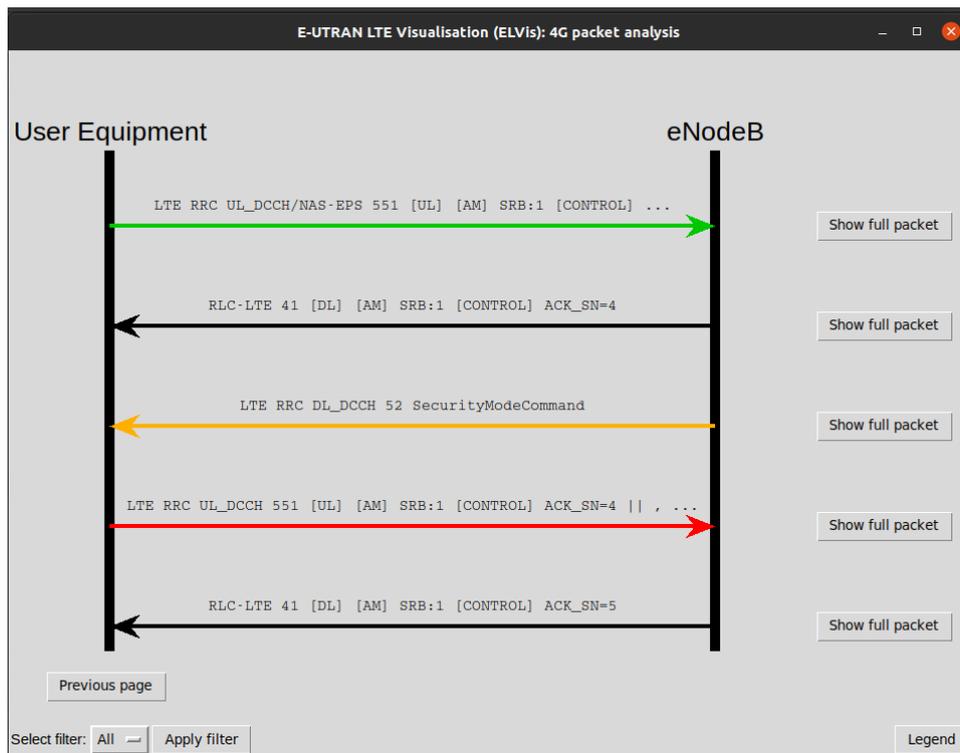


Figure 3.2: The user interface of ELVis

At the bottom of the vertical lines are buttons to traverse the packets loaded into the script. To the right of the graph are buttons to show packets in more detail. If the user clicks this button, a new window pops up, showing a summary of the packet, all the categories it belongs to, and the analysis result of the packet. Below that is the packet in its entirety as parsed by pyshark. An example of ELVis showing the details of a packet can be seen in figure 3.3.

```
Packet: LTE RRC_DL_DCCH/NAS-EPS 66 DLInformationTransfer, Security ...
-----SHORT DESCRIPTION-----
Summary: 13 0.178135 LTE RRC_DL_DCCH/NAS-EPS 66 DLInformationTransfer, Security mode command
Categories: ['Information Transfer', 'Security Mode Command', 'Attach Procedure', 'Analysed']
Error score: 1
-----ANALYSIS-----
WARNING: Null ciphering algorithm in use. Data is not encrypted over air interface.
        Data could be read by third parties.
-----FULL PACKET DATA-----
Packet (Length: 66)
Layer USER_DLT:
Layer UDP:
    Source Port: 48879
    Destination Port: 57005
    Length: 66
    Checksum: 0 (Illegal)
    Expert Info (Error/Checksum): Illegal Checksum value (0)
    Illegal Checksum value (0)
    Severity level: Error
    Group: Checksum
    Checksum Status: Unknown
    Stream index: 0
    Timestamps
    Time since first frame: 0.178135000 seconds
    Time since previous frame: 0.001056000 seconds
```

Figure 3.3: Showing a packet in ELVis

Chapter 4

Capture analysis

Arguably the most important part of ELVis is the packet analysis. The idea is to go over every packet at least once to find errors compared to the correctness of the protocol. The files in the source code performing the analysis are all located in the `analyser` module of the program. To perform analysis, the packets have been split into different categories. The categories relevant for analysis are:

- SecurityModeCommand
- Identity request
- Authentication request
- NAS (those used in the initial attach procedure, see section 2.3.3)

Note that not all RRC and NAS messages are being analysed in the thesis report. Only those relevant to the connection establishment will be taken into account here.

In the following subsections, we will discuss how the analysis is performed and what mistakes are being looked for. They will come in the order mentioned above, as that is also the chronological order of when the analysis scripts for these packets were created.

The script also uses the packets to gather information about the UE and other configuration parameters. These are stored in a dictionary called `ue_info`. Of course, it does require the assumption that at least some information sent between the UE and eNodeB is accurate. As such, we assume the attach request to contain complete and accurate information about the UE, as this is a message on which all other messages in the attach procedure rely. If this information appears to be inaccurate, however, the analysis for specific phases in the initial attach will probably catch it.

Every time an abnormality is detected in a packet, a (usually) one-sentence description of the error will be added to a string that is stored in the `Packet` class, together with the severity of the abnormality. We have defined two kinds of abnormalities: warnings and errors. Warnings occur

```

--- Software Radio Systems EPC ---
Couldn't open , trying /root/.config/srsran/epc.conf
Reading configuration file /root/.config/srsran/epc.conf...
Couldn't open user_db.csv, trying /root/.config/srsran/user_db.csv
MSE Initialized.
MME S11 Initialized
MME GTP-C Initialized
MME Initialized, MCC: 0xf001, MNC: 0xff01
SPGW GTP-U Initialized.
SPGW S11 Initialized.
SP-GW Initialized.
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB Id: 0x19b
S1 Setup Request - MCC:001, MNC:01
S1 Setup Request - TAC 7, B-PLMN 0xf110
S1 Setup Request - Paging DRX V128
Sending S1 Setup Response
Initial UE message: L1BLTE_MME_MSG_TYPE_ATTACH_REQUEST
Received Initial UE message -- Attach Request
Attach request -- M-TMSI: 0x79f89616
Attach request -- eNB-UE SIAP Id: 1

Built in Release mode using commit 5275f3336 on branch master.
Opening 1 channels in RF device=zmq with args=fall_on_disconnect=true,tx_port=tc
p://*:2000,rx_port=tcp://localhost:2001,ld=enb,base_srate=23.04e6
Available RF device list: zmq
  CHX base_srate=23.04e6
  CHX ld=enb
  Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 declination)
  CHX rx_port=tcp://localhost:2001
  CHX tx_port=tcp://*:2000
  CHX fall_on_disconnect=true

=== eNodeB started ===
Type <t> to view trace
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 declination)
Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 declination)
Setting Frequency: DL=2600.0 MHz, UL=2500.0 MHz for cc_idx=0 nof_prb=50
RACH: tti=341, cc=0, preamble=1, offset=0, temp_crtti=0x46
User 0x46 connected
Disconnecting rnti=0x46.
^Cstopping ..
Saving MAC PCAP (DLT=149) to /home/thomas/enb.pcap
Saving SIAP PCAP file (DLT=150) to /home/thomas/enb_siap.pcap

Available RF device list: zmq
  CHX base_srate=23.04e6
  CHX ld=ue
  Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 declination)
  CHX rx_port=tcp://localhost:2000
  CHX tx_port=tcp://*:2001
  Waiting PHY to initialize ... done!
  Attaching UE...
  Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 declination)
  Current sample rate is 1.92 MHz with a base rate of 23.04 MHz (x12 declination)
  Found Cell: Mode=FDD, PCI=1, PRB=50, Ports=1, CP=Normal, CFO= 0.2 KHz
  Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 declination)
  Current sample rate is 11.52 MHz with a base rate of 23.04 MHz (x2 declination)
  Found PLMN: ID=00101, TAC=7
  Random Access Transmission: seq=1, tti=341, ra-rnti=0x2
  RRC Connected
  Random Access Complete. c-rnti=0x46, ta=0
  Network attach successful. IP: 172.16.0.2
  Software Radio Systems RAN (srsRAN) 11/1/2022 15:3:50 TZ:0
  ^Cstopping ..
  Received RRC connection Release (releaseCause: other)
  RRC IDLE
  Saving MAC PCAP (DLT=149) to /home/thomas/ue_mac.pcap

```

Figure 4.1: Using srsRAN with ZeroMQ on Ubuntu 20.04

when the packet is not necessarily erroneous according to the specification, but the abnormality still should be looked at. An example could be the fact that ciphering algorithm “`eea0`”¹⁹ is chosen in the security mode command. Errors occur when the packet contains faulty data or has an error in the configuration according to specifications. An example of this could be if RLC is not performing in acknowledged mode during the initial attach procedure. In 3GPP specification 36.331 it is stated that this should be the case.[4]

4.1 Experimental setup

During the development of ELVIs, we have consistently used the “open source 4G and 5G software radio suite” srsRAN.[27] We first attempted to use a USRP B205mini, as this would allow us to connect our own mobile devices to srsRAN. However, due to unresolved technical difficulties, we instead used ZeroMQ, a networking library which can also act as a virtual radio interface. Using srsRAN in combination with ZeroMQ exactly as described in the srsRAN documentation,[28] we can use three terminals to simulate a mobile device connecting to a mobile network, as seen in figure 4.1.

While diverting from the original setup felt like a rough change at first, it turned out that ZeroMQ was a better solution all along. Using ZeroMQ and

¹⁹A “null-ciphering” algorithm, meaning that data will not be encrypted.

the UE protocol stack implemented by SRS we could change the workings of existing protocols in ways we could not if we were using our own phones. Details on the changes made to the srsRAN source code are described in appendix C.

Apart from the patches made here, there were changes to the configuration files srsRAN uses. These files are `ue.conf`, `enb.conf` and `epc.conf`. If there were changes made, the changes will be explained in their relevant sections. All the packet captures generated can be found in the `input/` folder of the ELVis source code, and the relevant captures will be referred in their own sections.

4.2 Analysing the SecurityModeCommand

The SecurityModeCommand exchange, during the initial attach, seemed like a good place to start, as the packets are relatively small. This allows for very easy analysis and seemed a good way to experiment with different methods of analysis. This analysis is performed in the `smc.py` file, and analyses all six messages of the SecurityModeCommand process – three for the NAS protocol and three for the RRC protocol. Both are checked for the same errors.

The main error in the SecurityModeCommand would be that the security capabilities do not match the command given by the core network. During analysis there will be a check if the replayed security capabilities in the NAS SecurityModeCommand message are similar to what was present in the attach request. There will also be a check if ciphering and integrity protection algorithms are used. Integrity protection algorithms must be used according to 3GPP specifications, while ciphering algorithms should be used.[2] By default ciphering is disabled in srsRAN, otherwise the packet captures would just contain encrypted data. However, we will still warn the user if ciphering is disabled.

We have tried to work with different security options, in all the configuration files. In `ue.conf`, we have set the UE’s capabilities to not allow “`eea0`”, the null ciphering algorithm, and results of those configurations can be seen in `enb_nas_security_mismatch.pcap` and `enb_rrc_security_mismatch.pcap`. Forcing the network to use ciphering resulted in `enb_nas_ciphering.pcap` and `enb_all_ciphred.pcap`.

To show how one of these messages is analysed, we take the NAS SecurityModeCommand message as an example. The code for the analysis of this message is found in listing B.4. This message both replays the security capabilities of the UE, provided in the attach request, and sends the command to use a certain ciphering algorithm and integrity protection algorithm.

The analyser first checks if the replayed UE security capabilities are the same compared to the ones sent in the attach request. If this is not the case,

the analyser will add this to the packet as a warning.

Afterwards, the script checks if the UE is actually capable of using the chosen security algorithms. If it is not, the error is reported in the analysis. It will also check if a failure message has been sent back. This is also a security mode message, and these failure messages are thus also analysed by the script. If no failure message has been sent back, this is also added to the analysis.

Finally, the script checks if the ciphering and integrity protection algorithms are actually used. If the MME chooses “`eea0`” or “`eia0`” as security algorithms, this means that the security algorithms are not actually used, which could lead to potential attacks on the confidentiality or integrity of the communication.

4.3 Analysing the identity request

The identity request happens before the security mode command, and is also a very simple part of the attach procedure. It asks the UE for either the IMSI or a IMEI value, however, we have only implemented the analysis when the UE asks for the IMSI, as this was the only implementation provided by srsRAN. Due to this sparse implementation of what is already a very small part of the connection establishment process, no packet captures have been generated for this part of the analysis. The analysis for the identity request happens in `identity.py`.

The identity request is only checked for the value it asks. If the query value is not the IMSI, the analysis will provide a warning that this deviates from the srsRAN implementation. Since this program was designed and tested using packet captures from srsRAN, there is no guarantee the format will be the same for other LTE implementations.

The identity response is checked if the value it returns matches the value that was queried. It would be an error if a IMEI was returned while the MME asked for the IMSI. If the user has provided the IMSI of the SIM-card, and the MME queried the IMSI, the analyser checks if the IMSI provided matches the IMSI sent to the MME. If this is not the case, a warning is provided and the IMSI gets updated to always be the value sent to the MME. The code used for this analysis can be found in listing B.5.

4.4 Analysing the authentication process

The authentication process, similar to the processes mentioned in earlier subsections, only consists of a few messages between the MME and UE. However, the analysis is a bit more involved, as we also want to check if the calculation of authentication values, such as RES and MAC-A, is calculated cor-

rectly.²⁰ The authentication procedure is analysed in `authentication.py`.

An attack on the attach procedure was described by Hussain *et al.* When analysing the authentication process we also look for signs of attack *A-3* as described in [16]. This disruption of service attack, dubbed the *numb attack* in the paper, causes the attach procedure to abort instantly, by sending a authentication reject message to the UE without any cause. Analysis will also find signs of this attack.

The effect of the *numb attack* can be found in `enb_auth_reject.pcap`. Other irregularities, such as a miscalculated MAC-A or RES value, can be found in `enb_auth_failure.pcap`, `enb_auth_failure_rand.pcap` and `enb_auth_reject_correct.pcap`. They were all generated using the authentication patch described in appendix C.2.

Calculating the RES value was easier than expected, as an open-source Python module called CryptoMobile[22] already existed to calculate the different MILENAGE output using the inputs received.²¹ The XOR algorithm was not implemented in this module, but using the srsRAN source code and the 3GPP specifications, we were able to easily translate the algorithm to Python. The code used for analysing the RES value in case the MILENAGE algorithm is used can be seen in listing B.6.

The analysis checks for signs of the *numb attack* by checking if an authentication reject happens before an authentication response message was sent. If the authentication reject was sent after an authentication response, it will check if the rejection was justified – if the sent RES value truly did not match the expected RES value.

4.5 PDCP MAC invalidation

Another disruption of service attack is found in the PDCP integrity protection. The integrity protection algorithm is chosen in the RRC security mode command. Integrity protection is done via a Message Authentication Code (MAC), which is usually a hash or cipher of a certain key and input variables. Before the algorithm calculating this MAC is chosen, PDCP still fills the MAC field in its messages, however this always equals `0x00000000` and is never looked at by the receiving end of the message. After an integrity protection algorithm is chosen, this MAC *is* verified. This is done by calculating the MAC again and checking if it matches the value contained in the message. If the two MACs are not the same, the MAC is considered invalid.

²⁰Elaboration on how authentication happens in LTE can be found in appendix A.

²¹Note that CryptoMobile is part of the ELVis source code. This is the only external module included in the source code as a large part of the module was unnecessary for our purposes, and it was the only Python module that could not be installed via `pip` or `apt`. Each file is copied directly from its repository, and every attempt has been made to clarify that we did not make this module.

It was found that, if the MAC is invalid during the attach procedure, the attach will simply halt. Because RLC is still in Acknowledged Mode during this procedure, there will be an acknowledgement packet sent back, making the sender believe that the packet was accepted by the receiving end. However, the attach procedure is not continued, and both parties will sit idly waiting for another message to arrive.

An example of what happens when the MAC is invalidated in the right message can be found in `enb_mac_invalid.pcap`. This file was generated using the patch described in appendix C.1. The analysis itself happens in `attach.py`, along some other minor analyses.

To analyse the capture file for this behaviour, we first checked if the attach procedure is complete. This is marked by the `RRC Connection Reconfiguration` message. If this message is not present in the capture file, we check if the capture might be incomplete. If the last message is a part of the attach procedure, then we provide a warning which says that, if the capture file is a complete capture, there might be an invalid MAC in the last attach message. If the last message is not a part of the attach procedure (for instance an `ACK` packet or an `RRC Connection Release` packet), we assume that this is because of an invalid MAC in the last packet.²² For reference, the code used to detect this behaviour can be found in listing B.7.

²²Another approach would be to calculate the MAC for each packet we analyse, however this turned out to be impossible. The reason for this is discussed in section 5.1.3.

Chapter 5

Discussion

Since 4G research and development is a fairly specialised field of study, there were some limitations that we needed to adhere to during the development of ELVis. In this chapter we will describe these limitations, the impact they had on the product in its current state and what future research and development could be done to improve the tool. Afterwards we will describe what we envision the tool could be used for and who would benefit the most from it.

5.1 Limitations

First, we will discuss the limitations of ELVis. We tried to make the list as comprehensive as possible, however the possibility that a bug or problem with the code has been missed can not be overlooked. After discussing all the limitations, there will be a discussion on other vulnerabilities that we could not make an analysis for, but are mentioned in other research papers.

5.1.1 Limitations on parsing

The parsing module used for loading the packet capture into the program makes use of Tshark, which itself is a terminal-based version of Wireshark. This means that, to parse the data correctly, the user needs to have both Wireshark and Tshark installed on their machine. While this is not the only dependency that ELVis has, and while most people interested in 4G packet analysis probably already have this network analysis tool installed, it still feels a bit disappointing to have to install a wholly different application just to run this parser.

Even with Wireshark and Tshark already in place, there still needs to be some configuration to be able to use the 4G/LTE dissectors. ELVis can do this for the user, but this does require that the user specifies where the Wireshark configuration files are located, unless the default path for Ubuntu is used. This does require that Wireshark is launched at least once

before starting ELVis. This might be confusing for someone who simply downloaded Wireshark to use ELVis, and reinforces the idea that it would be better if there simply was a module or library doing all this for us.

There simply exists no module or library in any language that supports 4G/LTE packet dissecting. Possible alternative Python modules were already discussed in section 3.2, but these only support Ethernet and WiFi dissecting. To develop a parser for 4G/LTE traffic would mean either extending modules or libraries already in place or building a wholly new one from the ground up. No matter what, it would be quite a bit of work, but would greatly increase the ease with which people can view and analyse their 4G network traffic.

Pyshark is also very inefficient. With ELVis in its current state, every packet in a capture file is checked three times before parsing is complete – one for the packet itself, one for the one-line summary, and one for the raw bytes data. This is a lot of redundancy, and leads to a very slow parsing if large files are analysed.

5.1.2 Limitations on visualising

The visualisation of the packet flow turned out alright, but there are definitely still things that could be improved.

The first one is the overall look of the program. Referring back to figure 3.2, it is clear that the visual style of the program is very basic. This is not necessarily a bad thing, however if the program is going to be used by a larger audience, some visual touch-ups might make the program a bit easier on the eye.

Another limitation is the window itself. Right now, only five packets can be viewed at once on the screen. The idea was to make the window resizable, however due to limitations in time and knowledge of Tkinter, this was not yet implemented. For now it is not an issue, as the connection establishment process should only concern the first two dozen packets in a capture file, however if other parts of 4G/LTE traffic will be analysed in a future version of ELVis, such as a handover procedure or connection release, a solution for this problem will have to be created.

5.1.3 Analysis limitations

The main limitation for analysis is the availability of data. Currently, the user has the option of providing SIM data to the analyser, however this is not mandatory for analysis to be performed. The catch is that, without this data, analysis is fairly limited, as it is required to check the correctness of some parts of the information transfer. Without the SIM data, there can be no calculation of keys and thus no correctness analysis of the authentication protocol. ELVis also can not check if the identity request provides the correct IMSI value without this.

In case traffic on an eNodeB device is analysed, the analyser only supports analysis of communication with one UE device. This is because data used for analysis is gathered from an attach request, and distinguishing between different mobile devices felt out of scope for this project. The analyser thus also assumes the attach request to contain correct information, otherwise the analyser barely has any reference points to compare the rest of the capture to.

The setup is also something that limits the usability of this application. This tool was developed on an Ubuntu system, and was developed with that in mind. Although this program might work with Windows or OSx as well, there was no testing performed on those operating systems. A similar note holds for the packet captures used in testing the software. Each packet capture was generated using srsRAN, and thus that implementation is assumed. While srsRAN adheres to the 3GPP specifications, they do not implement everything entirely.²³

5.1.4 Future analysis opportunities

There has been a lot of research into attacks on the LTE network, and all the research combined leaves an extensive list of vulnerabilities existing in the fourth generation of mobile networks. While the ultimate scenario would have us incorporating all these vulnerabilities into ELVis, the reality is less optimistic. This section will document attacks that were found that either were slightly outside the scope we set for ourselves in this project, or attacks that were not possible to detect with the data that was given.

The most immediate example is the MAC invalidation DoS attack that is discussed in section 4.5. This attack can sadly not be completely analysed as the computation of the MAC requires a certain key which can not be obtained by ELVis.²⁴ While it is still possible to detect behaviour in the packet capture that might be caused by an invalid MAC, this behaviour could also have other causes (such as simply an incomplete attach procedure being captured).

Fei *et al.* described an attack which could leak the IMSI to an attacker.[12] It involves setting up a rogue eNodeB and tampering the MIB and SIB messages to nudge the UE into connecting to the rogue eNodeB. They can then perform an identity request during the attach procedure to get the IMSI. This requires an extra SIB message (SIB5) which has a field called "cellReselectionPriority" which, when at its highest value, urges the UE to switch to a different cell to connect to. However, there might be other reasons to have

²³An example of this is the identity request, where srsRAN will only ever ask the IMSI of a connecting UE, whereas the IMEI may also be asked according to 3GPP specification 24.301.[1]

²⁴The key required is K_{eNB} , which is periodically regenerated using K_{eNB}^* . This key requires the physical cell ID and target physical cell downlink frequency, two variables which are neither fixed or transmitted through messages.

such a high priority, so it would not do to add a warning in analysis every time a maximal priority is found in an SIB5 message. This attack could not be analysed, however, as srsRAN only implements SIB1, SIB2 and SIB3.

Kumar and Lakshmy described paging attacks which, while not exactly part of the connection establishment process, can still find the location of a UE in a 2km² area and thus might be an interesting and important attack to try and analyse in the future.[29]

Shaik *et al.* have written about vulnerabilities related to the UE capability information exchange.[25] UE capability information can be accessed without authentication having taken place. This means that a rogue base station can get access to capabilities from a UE, leading to privacy risks, and being able to know exactly what devices are present in a specific area. If these UE capabilities are sent before the security setup is complete for both RRC and NAS protocols, a man in the middle could tamper with the capabilities sent by the UE, leading to downgrade attacks²⁵ and even possible accelerated battery draining for certain devices. We have added a warning if UE capabilities are enquired before security is setup, but there is no seeing if these capabilities have been tampered with.

5.2 Use cases

The introduction already discussed some motivation behind the development of this tool; namely that current analysis tools of mobile network traffic is fairly limited. Tools that do allow an insight in 4G network traffic, such as Wireshark, are limited in their capabilities and require great understanding of 4G to understand and look through.

ELVis attempts to improve on the accessibility and readability of 4G network traffic, by presenting an analysis which already shows where flaws in the traffic reside. This chapter will summarise the possible use cases for both developers interacting with mobile networks on a low level and individuals interested in 4G traffic analysis. This chapter assumes that ELVis is in a complete state, with every aspect of 4G communication analysed perfectly. This is not yet the case as of the completion of this thesis project.

5.2.1 Developers and Mobile Network Operators

The E-UTRAN LTE Visualiser would be most interesting for people who are already familiar with the architecture of mobile networks. This holds for both individual developers, creating their own operating system of baseband processor, who require to develop the workings of the low-level protocols present on the 4G/LTE stack. ELVis would allow developers to quickly see where their implementation fails according to official specifications.

²⁵Attacks leading to the UE being downgraded to an older, slower generation of mobile networks, such as 3G or GSM.

For Mobile Network Operators (MNOs) this same principle holds. However, these companies might be interested in other types of analysis as well; namely analysis of the data travelling over their network. While research into the kind of traffic flowing over the network has been performed[20], together with performance analysis of 4G[18], there exists no (public) tool which analyses what traffic is generated by a specific UE device. While this is an interesting field of study, ELVIs is not nor will ever be a tool suited for this purpose.

To further exemplify how ELVIs could be useful for developers, let us explain how we found a flaw in the srsRAN implementation of an eNodeB using our own developed application. During the generation of the packet captures used for the analysis of the SecurityModeCommand, as described in section 4.2, we found that the RRC protocol did not adhere to the capabilities sent by the UE. The packet capture in which this is visible is `enb_rrc_security_mismatch.pcap` in the ELVIs source code.

In the UE configuration of srsRAN we set the `nas.eea` field, which lists the supported encryption algorithms, to "1,2". This means that the null ciphering algorithm "eea0" is not supported by the UE.²⁶ This fact is also reflected in the attach request.

The problem lies in the RRC SecurityModeCommand message. This message contains the encryption algorithm the UE must use, and in this packet capture it is visible that the AS encryption algorithm should be "eea0", even though the UE does not support it. The UE still accepts the command, however, and "eea0" will still be used. This could lead to issues if "eea0" truly did not appear in the source code, and is something the developers might need to fix. Without ELVIs it would have been harder to notice that this mismatch occurred, and there are probably a lot of other errors or issues we could not think of that would be very useful to have analysed.

5.2.2 Interested individuals

While this tool, on surface level, might be interesting to individuals wishing a better understanding of the communication happening on their own mobile device. While ELVIs is meant for this exact purpose, these individuals might be disappointed when attempting to use this tool in its current state.

While it is a Python application anyone with a basic understanding of a command line interface should be able to use, the workings of the application itself are slightly more convoluted. Users are expected to provide packet captures themselves, which could be generated using applications like srsRAN. But while developers are more experienced in the field and probably know how to generate packet captures of traffic themselves, the

²⁶Note that the 3GPP specifications do not explicitly say that "eea0" must be supported, however it is advised.

average individual probably is not. Even if they manage to get srsRAN installed, they will have trouble actually connecting to the service, since this might mean reprogramming a SIM card.²⁷

Even if an interested individual managed to get a packet capture loaded into the program, together with supplemental data like the key and IMSI of a SIM card, the usefulness of the information might be limited. While a line in analysis like "Data is not encrypted over air interface." might be very easy to understand and useful to know, a line like "RES (0048124f2c50c327) does not match expected value (dd48124f2c50c327)" does not convey any information the individual really needs to know.

If we want ELVis to be more accessible and useful for individuals, it would be better to run it as a service on the UE they want to analyse. The service could provide notifications to the user if there is something they ought to know (such as missing integrity/ciphering options) while the more technical details could be spared unless the user desires to be notified of them. As it is right now, though, an individual with only basic knowledge of network structure and cryptography will not have much use of this tool.

²⁷Not to mention the fact that the radio devices needed to connect to srsRAN are often very expensive.

Chapter 6

Conclusions

The E-UTRAN LTE Visualiser, or ELVIs for short, is a tool that parses, visualises and analyses 4G network traffic. This tool is necessary because, according to prior research, 4G communication does not always happen according to the official specifications, without the user being aware of this fact. With ELVIs, the aim is to provide more accessibility to gain insight into where communication might not happen as specified.

As of the finalisation of this thesis project, ELVIs only analyses the connection establishment between a UE device and a mobile network, from the sending of the `RRC Connection Request` by the UE to the reception of the `RRC Connection Setup Complete` by the UE. The most important parts that were analysed are the security mode command, authentication request and some general attach procedures. Signs for a few disruption of service attacks are also analysed and correctly pointed out.

During development, it was found that the analysis capabilities were fairly limited, as finding the cause of some problems, such as an abrupt end of the attach procedure caused by an invalid MAC on the PDCP protocol, were hard to detect as the analyser only has limited information to work with. Other important limitations that arose during development were the testing capabilities. As we worked with srsRAN to provide capture data, we relied on this implementation of an LTE network for testing our analysis. This resulted in not being able to test some edge cases that were mentioned in the official specifications, like the EPC asking for the IMEI in the identity request, whereas srsRAN only ever asks for the IMSI.

While ELVIs is meant for gaining insight into where 4G/LTE communication goes wrong according to the specifications, for now it is mostly useful for developers aiming to implement protocols on the second and third layer of the 4G/LTE protocol stack. For interested individuals attempting to gain a better understanding of how their mobile device interacts with a mobile network, the software has too many requirements and the analysis is probably too complicated to understand.

Bibliography

- [1] 3GPP. TS24.301 *Universal Mobile Telecommunications System (UMTS); LTE; 5G; Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3*.
- [2] 3GPP. TS33.401 *Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 3GPP Systems Architecture Evolution (SAE); Security architecture*.
- [3] 3GPP. TS34.108 *Universal Mobile Telecommunications System (UMTS); LTE; Common test environments for User Equipment (UE); Conformance testing*.
- [4] 3GPP. TS36.331 *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol Specification*.
- [5] 3GPP. TS35.206 *Universal Mobile Telecommunications System (UMTS); LTE; 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f_1 , f_1^* , f_2 , f_3 , f_4 , f_5 and f_5^* ; Document 2: Algorithm specification (version 10.0.0)*, April 2011.
- [6] Mohammad Al Shinwan and Kim Soo. A Flat Mobile Core Network for Evolved Packet Core Based SAE Mobile Networks. *Journal of Computer and Communications*, 5, 03 2017. doi:10.4236/jcc.2017.55006.
- [7] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm Game: Automatic Discovery of LTE Vulnerabilities Through Documentation Analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1197–1214, 2021. doi:10.1109/SP40001.2021.00104.
- [8] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. LTE Security Disabled: Misconfiguration in Commercial Networks. *WiSec '19*, page 261–266, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3317549.3324927.

- [9] Sakshi Chourasia and Krishna M. Sivalingam. SDN based Evolved Packet Core architecture for efficient user mobility support. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2015. doi:10.1109/NETSOFT.2015.7116148.
- [10] Jeffrey Cichonski, Joshua Franklin, and Michael Bartock. Guide to LTE security. Technical report, National Institute of Standards and Technology, 2016.
- [11] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.
- [12] Teng Fei and Wenye Wang. LTE Is Vulnerable: Implementing Identity Spoofing and Denial-of-Service Attacks in LTE Networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019. doi:10.1109/GLOBECOM38437.2019.9013397.
- [13] Frédéric Firmin. The Evolved Packet Core, year n.a. Last accessed: November 10th 2021. URL: <https://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core>.
- [14] Ushasree Gorrepati, Pavol Zavarsky, and Ron Ruhl. Privacy Protection in LTE and 5G Networks. In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, pages 382–387, 2021. doi:10.1109/ICSCCC51823.2021.9478109.
- [15] Dor Green. pyshark. <https://github.com/KimiNewt/pyshark>, 2013–2021.
- [16] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. *Network and Distributed Systems Security (NDSS) Symposium 2018*. URL: <https://par.nsf.gov/biblio/10055689>.
- [17] KP. LTE: Authentication Response, Nov 2011. Last accessed: December 11th 2021. URL: <http://howltestuffworks.blogspot.com/2011/11/authentication-response.html>.
- [18] Fidel Krasniqi, Arianit Maraj, and Emin Blaka. Performance analysis of mobile 4G/LTE networks. In *2018 South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference (SEEDA-CECNSM)*, pages 1–5, 2018. doi:10.23919/SEEDA-CECNSM.2018.8544937.
- [19] Anna Larmo, Magnus Lindström, Michael Meyer, Ghyslain Pelletier, Johan Torsner, and Henning Wiemann. The LTE link-layer design. *IEEE Communications Magazine*, 47(4):52–59, 2009. doi:10.1109/MCOM.2009.4907407.

- [20] Feng Li, Xiaoxiao Jiang, Jae Won Chung, and Mark Claypool. Who is the King of the Hill? Traffic Analysis over a 4G Network. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018. doi:10.1109/ICC.2018.8422958.
- [21] Yadong Li, Danlan Li, Wenqiang Cui, and Rui Zhang. Research based on OSI model. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, pages 554–557, 2011. doi:10.1109/ICCSN.2011.6014631.
- [22] ‘mitshell’ (GitHub username). CryptoMobile. <https://github.com/mitshell/CryptoMobile>, 2017–2021.
- [23] David Rupprecht, Kai Jansen, and Christina Pöpper. Putting LTE Security Functions to the Test: A Framework to Evaluate Implementation Correctness. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.
- [24] Jaeku Ryu. Subframe number, SFN and HFN. Last accessed: January 7th 2022. URL: https://www.sharetechnote.com/html/Handbook_LTE_SFN.html.
- [25] Altaf Shaik, Ravishankar Borgaonkar, Shinjo Park, and Jean-Pierre Seifert. New Vulnerabilities in 4G and 5G Cellular Access Network Protocols: Exposing Device Capabilities. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, WiSec ’19*, page 221–231, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3317549.3319728.
- [26] Statista. Number of mobile 4G/LTE subscriptions worldwide by region from 2011 to 2026, 2021. Last accessed: November 9th 2021. URL: <https://www.statista.com/statistics/521572/4g-5g-mobile-subscriptions-worldwide/>.
- [27] Software Radio Systems. srsRAN. URL: <https://srslte.com>.
- [28] Software Radio Systems. srsRAN Documentation. https://docs.srsran.com/_/downloads/en/latest/pdf/. Last accessed: December 14th 2021.
- [29] K. Vignesh and K.V. Lakshmy. Averting Paging Related Attacks in 4G LTE Communication System. *International Journal of Recent Technology and Engineering (IJRTE)*, 8, July 2019. doi:10.35940/ijrte.B1029.0782S419.

Appendix A

LTE authentication

LTE uses an Authentication and Key Agreement protocol, which happens during the attach procedure. Most of the values are either stored on the SIM-card or provided by the MME, which takes values like the expected result (XRES) from the HSS, which has the key of every IMSI.

The AKA used to authenticate the UE and generate ciphering and integrity keys is called MILENAGE. This protocol is specified in 3GPP specification 35.206.[5] Since this thesis is following the SRS implementation of LTE, and this implementation uses version 10 of TS35.206, we will follow this version of the document as well.

There also exists a test algorithm for authentication. This algorithm is specified in 3GPP specification 34.108,[3] but should *never* be used in practice, due to very low confidentiality in the key derivation, which is easily visible following the definition in section A.2.

A.1 The MILENAGE protocol

While the specifications of MILENAGE do not require a certain block cipher, it does provide examples with Rijndael.[11] These examples were followed in SRS as well, and thus will also be followed in our explanation of the algorithm. We define the notation $AES_K(x)$ as encrypting x with Rijndael using key K .

The algorithm starts with providing some definitions. Values OPc and TEMP is defined, using the Operator Code stored on the SIM and the random value RAND:

$$\begin{aligned} \text{OPc} &= \text{OP} \oplus AES_K(\text{OP}) \\ \text{TEMP} &= AES_K(\text{RAND} \oplus \text{OPc}) \end{aligned}$$

An input field $IN1$ is defined as $SQN||AMF||SQN||AMF$. AMF is defined later and can be used in combination with input parameter $AUTN$ to find the SQN .

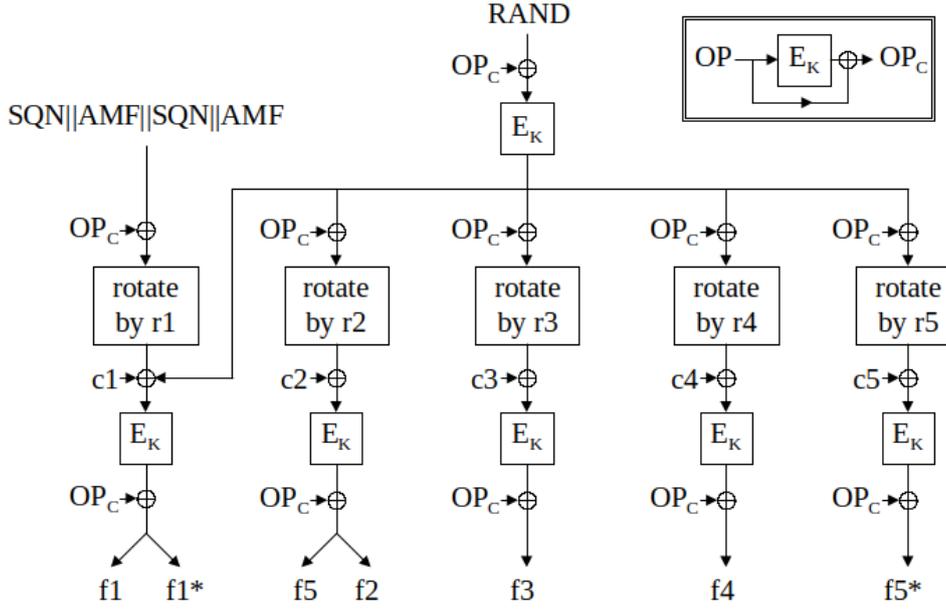


Figure A.1: The MILENAGE algorithm[5]

Five constants c_n are defined as the following 128-bit numbers:

$$\begin{aligned}
 c_1[i] &= 0 \text{ for } 0 \leq i \leq 127 \\
 c_2[i] &= 0 \text{ for } 0 \leq i \leq 127, \text{ except for } c_2[127] = 1 \\
 c_3[i] &= 0 \text{ for } 0 \leq i \leq 127, \text{ except for } c_3[126] = 1 \\
 c_4[i] &= 0 \text{ for } 0 \leq i \leq 127, \text{ except for } c_4[125] = 1 \\
 c_5[i] &= 0 \text{ for } 0 \leq i \leq 127, \text{ except for } c_5[124] = 1
 \end{aligned}$$

There are also five constants r_n which are defined as $r_1 = 64, r_2 = 0, r_3 = 32, r_4 = 64, r_5 = 96$, which will later be used for rotation. Rotation is defined as "The result of cyclically rotating the 128-bit value x by r bit positions towards the most significant bit." [5] So if $x = x[0]||x[1]||\dots||x[127]$ and y is x rotated by r bits, then

$$y = x[r]||x[r+1]||\dots||x[0]||x[1]||\dots||x[r-2]||x[r-1]$$

All these defined values are used to compute seven different outputs, as can be seen in figure A.1. In figure A.1, E_K represents AES_K .

Here is all the output computed by the MILENAGE algorithm:

- f_1 : MAC-A, an authentication code.
- f_1^* : MAC-S, a resynchronisation authentication code.
- f_2 : RES, the resulting value that gets sent in an authentication response.

- f_3 : CK, a confidentiality key used for the ciphering algorithm chosen in the security mode command.
- f_4 : IK, an integrity key used for the integrity algorithm chosen in the security mode command.
- f_5, f_5^* : AK, an anonymity key used in computing the AUTN value.

A.2 The XOR protocol

The XOR protocol is a lot simpler in implementation. The input is the same, however it simply performs a series of XOR operations to arrive at a RES, CK, IK and AK value. This is still secure when RAND and AUTN remain fresh, but it can be broken on repeat values. The CK and IK are also derived very easily from RES.

First an array called ‘xdout’ is created, with its contents being calculated as follows:

$$\text{xdout}[i] = \text{key}[i] \oplus \text{RAND}[i]$$

This array is used as output for the RES value, the CK, the IK and the AK, which are calculated as follows:

$$\begin{aligned} \text{XRES}[i] &= \text{xdout}[i] \\ \text{CK}[i] &= \text{xdout}[(i + 1) \bmod 16] \\ \text{IK}[i] &= \text{xdout}[(i + 2) \bmod 16] \\ \text{AK}[i] &= \text{xdout}[i + 3] \text{ for } 0 \leq i \leq 6 \end{aligned}$$

It is very easily visible that the CK, IK and AK are easily derived from the RES value, meaning that anyone intercepting the authentication response can derive the CK and IK themselves. This is dangerous, as anyone can then access the proceeding packets as if the ciphering and integrity algorithm were not there.

Appendix B

Code snippets

This appendix shows bits of code discussed in chapter 3 and 4, as to not clutter the discussion presented in those chapters.

Listing B.1: Reading a packet capture

```
1 import pyshark
2 capture = pyshark.FileCapture('enb.pcap', custom_parameters={'-C': '<4Gprofile>'})
```

Listing B.2: The parse_pcap function

```
1 pyshark.FileCapture.SUMMARIES_BATCH_SIZE = 4
2 raw_capture = pyshark.FileCapture(path, custom_parameters=
  options)
3 summaries = pyshark.FileCapture(path, custom_parameters=options
  , only_summaries=True)
4 assert len(raw_capture) == len(summaries)
5 capture = []
6 for packet, summary in zip(raw_capture, summaries):
7     if len(packet.layers) > 2 and vars(packet.layers[2])['_layer_name'] == 'mac-lte':
8         vars(packet.layers[2])['_layer_name'] = 'mac_lte'
9         sentence = summary.summary_line
10        capture.append(Packet(packet, sentence, 0))
```

Listing B.3: Parsing arguments

```
1 options = {'analyse': True, 'visualise': True}
2 parse_options = {'-C': '4GLTE'}
3 i = 1
4 while i < len(sys.argv):
5     if sys.argv[i][0] != '-':
6         i += 1
7         continue
8     if sys.argv[i] == '-analyse' or sys.argv[i] == '-a':
9         options['visualise'] = False
10    # ...
11    if (sys.argv[i] == '-filter' or sys.argv[i] == '-f') and i
    + 1 < len(sys.argv):
```

```

12     if sys.argv[i + 1][0] != '-' and not sys.argv[i+1].
    endswith('.pcap'):
13         parse_options['-f'] = sys.argv[i+1]
14     else:
15         print('WARNING: Filter option is set, but no filter
    string is provided.')
16         print('Program will be run without filter.')

```

Listing B.4: Analysing NAS EMM message #93: Security Mode Command

```

1 # check for accurate security capabilities (matching from
    attach request)
2 if not packet_info == ue_info['security_capabilities']:
3     packet.add_analysis(
4         'Security capabilities in NAS security mode command do
    not match capabilities in attach request.', 1)
5
6 # check if UE capable of ciphering algorithm
7 if ue_info['security_capabilities'][ca] == '0':
8     packet.add_analysis('UE not capable of using NAS chosen
    ciphering algorithm.', 3)
9     nas_smc_fail_sent(packet, packets)
10
11 # check if UE capable of integrity algorithm
12 if ue_info['security_capabilities'][ia] == '0':
13     packet.add_analysis('UE not capable of using NAS chosen
    integrity algorithm.', 3)
14     nas_smc_fail_sent(packet, packets)
15
16 # check if ciphering and integrity protection are used
17 smc_algo_used(packet, ca, ia)

```

Listing B.5: Analysing NAS EMM message #86: Identity response

```

1 # check if requested type is response type
2 response_type = packet.data.layers[2].get('gsm_a.ie.
    mobileid_type')
3 if not ue_info['identity_request_type'] == response_type:
4     packet.add_analysis('Identity response does not contain
    queried value by MME.', 3)
5
6 # check if response matches known value
7 imsi = safe_dict_get(ue_info, 'imsi')
8 if imsi and not imsi == packet.data.layers[2].get('e212.imsi'):
9     packet.add_analysis('IMSI in response does not match value
    from SIM configuration.', 1)
10    packet.add_analysis('Changing stored value to value read
    from identity response.', 0)
11 ue_info['imsi'] = packet.data.layers[2].get('e212.imsi')

```

Listing B.6: Analysing the RES value in a NAS authentication response.

```

1 # create cipher
2 if op := safe_dict_get(ue_info['sim_info'], 'op'):
3     cipher = Milenage(op)

```

```

4 else:
5     opc = bytes.fromhex(safe_dict_get(ue_info['sim_info'], 'opc
6     '))
7     cipher = Milenage(op)
8     cipher.set_opc(opc)
9
10 # get RES, CK, IK and AK values
11 xres, ck, ik, ak = cipher.f2345(key, rand)
12
13 # check if authentication should be successful by comparing RES
14 # and XRES
15 res = ''.join(packet.data.layers[2].get('nas_eps.emm.res').
16 # split(':'))
17 xres = bytes.hex(xres)
18 if res != xres:
19     packet.add_analysis(f'RES ({res}) does not match expected
20 # value ({xres})', 4)
21     if not get_attach_message(packets, '92'):
22         packet.add_analysis('No authentication failure message
23 # sent after mismatch in expected value.', 3)
24
25 # update keys
26 # ...

```

Listing B.7: Searching for behaviour caused by an invalid PDCP MAC.

```

1 # find if RRCConnectionReconfiguration occurred, if not, attach
2 # procedure is incomplete
3 complete = False
4 for packet in packets:
5     if 'RRCConnectionReconfiguration' in packet.summary:
6         complete = True
7
8 if complete or safe_dict_get(ue_info,
9 # 'rrc_ca') != 'eea0': # if AS
10 # ciphering is enabled, we will not be able to know if attach
11 # finished
12 return
13
14 # find if attach packet is last packet of capture
15 last_attach = attach_packets[-1]
16 if last_attach == packets[-1]:
17     # send warning for possible incomplete file or invalid PDCP
18     # MAC
19     last_attach.add_analysis(
20 # 'Attach procedure incomplete.\nIf there are no possible
21 # causes listed in this packet, it might be because of an
22 # invalid MAC.',
23 # 1)
24
25 else:
26     # send error for probable invalid PDCP MAC
27     last_attach.add_analysis('Attach procedure incomplete.\n
28 # This might be because of an invalid PDCP MAC.', 3)

```

Appendix C

Patching srsRAN

Several changes in the srsRAN source code were made to generate packet captures containing errors. This involved a patch which changes the MAC for AS integrity protection, used for an attack described in section 4.5, or a patch tampering with RES values during the authentication procedure. The structure for most of these patches are the same, and will all be described in detail in the following sections.

C.1 Invalidating the PDCP MAC

This first patch involves the file `lib/src/pdcp/pdcp_entity_lte.cc` of the srsRAN source code. Specifically, the `write_sdu()` function is being changed here. The patch starts by opening a configuration file which holds variables stating if we want to invalidate the MAC of a certain packet, and if so, which packet that should be.

After the file has been read, it tests if we are currently writing a packet that needs the MAC to be changed. If it is, the program changes the MAC to be `0x01010101` so it is easily recognisable in a packet capture. The patched code can be found in listing C.1.²⁸ The results of using this patch are seen in the `enb_mac_invalid.pcap` file in the ELVis source code.

```
1 bool invalidate_mac = false;
2 uint32_t tx_count_comparison = 0;
3
4 FILE *fptr;
5 if ((fptr = fopen("/home/thomas/.config/srs-test/mac_invalidate
6   .txt", "r")) != NULL) {
7     char buff[1024];
8     char *pch;
9     int line_cnt = 0;
10    while (fgets(buff, 1024, fptr) != NULL) {
11      int word_cnt = 0;
12      pch = strtok (buff, " ");
```

²⁸In this and all following listings, the logging functions have been removed for a clearer overview of what was changed.

```

12     while (pch != NULL) {
13         if (word_cnt == 2 && line_cnt == 0)
14             invalidate_mac = static_cast<uint32_t>(std::stoul(pch))
15         ;
16         else if (word_cnt == 2 && line_cnt == 1)
17             tx_count_comparison = static_cast<uint32_t>(std::stoul(
18 pch));
19         pch = strtok (NULL, " ");
20         word_cnt = word_cnt + 1;
21     }
22     line_cnt = line_cnt + 1;
23 }
24
25 if (invalidate_mac == 1 && tx_count == tx_count_comparison){
26     // invalidating mac
27     uint8_t invalid_mac[4] = {0x01, 0x01, 0x01, 0x01};
28     append_mac(sdu, invalid_mac);
29 }
30 else {
31     // standard procedure
32     append_mac(sdu, mac);
33 }

```

Listing C.1: MAC invalidation patch

C.2 Patching authentication

srsRAN was patched in multiple ways to break authentication, both on the MME's side and on the UE's side. On the network side, in the source file `srsepc/src/mme/nas.cc`, in the `pack_authentication_request()` function, three patches were made. The first two involve the variables that are sent to the UE. Both the MAC and the RAND value can be manipulated so that authentication should fail. The third involves the *numb attack* explained in [16]. If a variable in the configuration file is set to any value other than 0, it will send a authentication reject instead of an authentication request.

On the mobile device's side, in the source file `srsue/src/upper/nas.cc`, in the `send_authentication_response()`, one patch is made which alters the RES value from its normally calculated value, to see if the network would pick up on it and send a rejection back. The patch for the authentication request is visible in listing C.2. The packet captures generated using these patches can all be found in the `input/` folder of the ELVis source code.

```

1     // file reading, similar to MAC invalidation patch
2
3     if (auth_failure_mac != 0){
4         m_sec_ctx.autn[15] = 0;
5     }

```

```

6
7  if (auth_failure_rand != 0){
8      m_sec_ctx.rand[0] = 0;
9  }
10
11  if (auth_reject == 0) {
12      // standard procedure
13      LIBLTE_MME_AUTHENTICATION_REQUEST_MSG_STRUCT auth_req;
14      memcpy(auth_req.autn, m_sec_ctx.autn, 16);
15      memcpy(auth_req.rand, m_sec_ctx.rand, 16);
16      auth_req.nas_ksi.tsc_flag =
17      LIBLTE_MME_TYPE_OF_SECURITY_CONTEXT_FLAG_NATIVE;
18      auth_req.nas_ksi.nas_ksi = m_sec_ctx.eksi;
19
20      LIBLTE_ERROR_ENUM err =
21      liblte_mme_pack_authentication_request_msg(&auth_req, (
22      LIBLTE_BYTE_MSG_STRUCT*)nas_buffer);
23      if (err != LIBLTE_SUCCESS) {
24          m_logger.error("Error packing Authentication Request");
25          srsran::console("Error packing Authentication Request\n")
26          ;
27          return false;
28      }
29      return true;
30  }
31  else{
32      // send authentication reject
33      return pack_authentication_reject(nas_buffer);
34  }

```

Listing C.2: Authentication patch (MME)