# Passive Security Analysis of Current TLS Implementations and Configurations in the eduroam EAP-TLS environment

University of Bremen, Faculty 3 Mathematics and Computer Science

Jan-Frederik Rieckers

August 1, 2021

A thesis to be submitted to fulfill the requirements for the degree of
*Bachelor of Science*

1. Gutachter: Dr. habil. Karsten Sohr
2. Gutachter: Prof. Dr. Carsten Bormann

## Eigenständigkeitserklärung

Ich versichere, den Bachelor-Report ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 01.08.2021

- Jan-Frederik Rieckers -

## License notice



Version notice: This PDF was build from commit ID 482c19b (2021-11-14 13:01:46 +0100)

# Acknowledgements

I would like to give special thanks to

**Dr. habil. Karsten Sohr**, my first reviewer

for mentoring this thesis, reviewing many draft versions and all the advice he gave.

**Prof. Dr. Carsten Bormann**, my second reviewer

for mentoring this thesis and connected work and giving advice.

**Birgit and Dirk Rieckers**, my parents

for supporting me for the whole time of my studies.

**Niels Pollem, Markus Germeier** and all my other former colleagues at the ZfN at the University of Bremen

for the opportunity to learn about eduroam, providing an appreciative environment at work, supporting this thesis, the access to the infrastructure, and pushing me towards finishing this thesis.

**Ralf Paffrath** and the whole DFN

for giving me access to the data at the DFN, and providing resources for the analysis of the data.

**Dr. Martin Pauly and Yassin Alshater**, Employee and student at University Marburg

for providing TLS Client Hellos to identify specific clients.

**My close friends** (unnamed, you know who you are)

for supporting me in writing this thesis by motivating me, proofreading and enduring my occasional rants.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Many research institutions all over the world offer their members free Internet access. The first wireless access networks were locally deployed, allowing only members of their own institution. Over the years, some institutions have created links between their networks, allowing members of other institutions in the same city to access the network. Starting in 2002 eduroam was developed, to provide a worldwide roaming network where members of participating institutions can access the network of other participating institutions with the credentials of their home institution.

The eduroam network connects institutions using the protocols RADIUS (Remote Authentication Dial In User Service) and EAP (Extensible Authentication Protocol) to ensure a personalized login into the network while keeping the separation between Identity Providers (IdP), which hold account information, and Service Providers (SP), which provide Internet access. In most cases, an IdP also acts as SP, but there are a number of SPs, which just provide Internet access without being an IdP.

The combination of RADIUS and EAP in eduroam allows each IdP to choose their EAP authentication method independently, since the routing is done via the RADIUS protocol. To ensure the privacy of the users and to keep the credentials secret from the visited institutions, the login has to be encrypted. Most EAP methods use the well-known security protocol Transport Layer Security (TLS) to achive this.

The most commonly used EAP methods are TTLS (Tunneled TLS) and PEAP (Protected EAP), which both rely on EAP-TLS. EAP-TLS specifies the usage of TLS inside EAP.

Like for most other federated networks, where the members have to trust each other, the security of the whole network depends on the security of the weakest link.

In EAP-TLS, there are two relevant classes of devices for the security analysis: The supplicants (acting as TLS clients) and the authentication servers (acting as TLS servers).

First observations in the eduroam network at the University of Bremen have shown several issues with the current TLS implementations of the EAP-TLS clients. Especially older clients have shown issues such as the usage of old TLS versions and insecure cipher suites. Some current devices are also missing support for security measures such as OCSP stapling (Online Certificate Status Protocol), which allows to check the revocation status of the certificate without need for an active Internet connection.

With the observations on the client side it might also be assumed that the TLS implementation of the TLS

servers show certain flaws. This begins with the support for different TLS versions or the implementation of TLS extensions, but also includes configuration parameters as supported cipher suites or supported curves for elliptic curve key exchange algorithms.

## 1.2 Research Target and scope

This thesis gives an overview of the current operational practices in the usage of TLS in EAP-TLS and aims to determine if security problems exist.

This will be achieved by a passive analysis of the EAP and TLS handshake messages exchanged in the process of the eduroam login. By passively analyzing the traffic, one can learn about the capabilities of the client, since the client sends its capabilities in the Client Hello message of the TLS handshake. The server on the other hand reacts on the Client Hello. To assess the server fully, one has to simulate different clients and capture the corresponding answer. Since this thesis focuses on a passive analysis, it deals with the current status of client implementations and configurations. The reaction of the servers is captured and analyzed, but the complete analysis of the servers will be part of a future work.

The parameters analyzed on the client side will include the TLS version, cipher suites, supported elliptic curves and extensions.

In order not to raise false expectations about the content of this thesis, we would also like to point out the limits of its scope at this point.

This thesis will not deal with the verification of the server certificate by the client. This is the biggest weakness of the eduroam infrastructure in the eyes of the author, nonetheless this topic had been discussed by a good deal of research. [1, 2]

Also, this thesis will not try to find weaknesses in the used encryption algorithms itself or target flawed implementations of the encryption algorithms. For the purpose of this thesis, all used encryption algorithms are considered secure, except for previously known flaws, e.g. the weak key strength of 40-bit ciphers or weaknesses in RC4.

Since TLS itself has evolved and is still evolving, this thesis also has to limit the scope to the analyzed TLS versions. This will be limited to the versions TLSv1.0, TLSv1.1 and TLSv1.2. Previous versions, namely SSLv2 and SSLv3, will not be considered. Later TLS versions, namely currently TLSv1.3, will also be considered out of scope, since the usage of TLSv1.3 within EAP-TLS has not been fully specified at the time of writing.

## 1.3 Structure of this thesis

In the beginning of this thesis the basics will be provided which are needed to understand the following technical topics discussed. This will cover an overview over the relevant protocols, their target and basic operation and a description of the way these protocols are used in the eduroam environment.

The following chapter covers the data collection. The analysis tool which collects the data used for the statistical analysis will be explained and known errors will be presented. It also includes an assessment of privacy implications brought about by the tool.

In chapter 4 the statistical analysis is performed. This starts with a short summary of observed irregularities in the lower protocol layers. In the following sections, the different aspects of the TLS handshake are analyzed separately. The chapter is concluded by a summary of the performed analyses and how the

analysis results have already been used.

The final chapter gives a conclusion of the performed analysis. It also gives an overview of possible configuration changes which could be executed by RADIUS server administrators to improve the security. It closes with an overview of limitations, open topics and an outlook on planned improvements of the analysis tool and further research topics.

# Chapter 2

# Basics

This chapter will give an overview of the different protocols and infrastructures referenced in this thesis. If the reader is already basically familiar with the described protocols, they may skip reading this chapter. Important details of the protocols and their effect on the work presented in this thesis will be repeated in the according chapter.

It is assumed that the reader is already familiar with the basic concepts of symmetric and asymmetric cryptography.

## 2.1 Communication Protocols

### 2.1.1 RADIUS and RADIUS over TLS

RADIUS (Remote Authentication Dial In User Service) is specified in RFC 2865 [3]. The RADIUS protocol is widely used in a number of different scenarios. It is a central part of many AAA infrastructures as it is capable of handling authentication, authorization and accounting. In the eduroam context, the accounting capabilities of RADIUS are not used on federation level. The authorization capability of RADIUS on federation level is usually reduced to a strictly Boolean response, whether the user is authorized to use eduroam; locally the authorization capability may be used to differentiate between the IdP's own users and roaming guests. The details of federation in eduroam is described in section 2.2 on page 10.

RADIUS builds on UDP and follows a strict request-response communication flow. The communication partners are the RADIUS server and the RADIUS client. The RADIUS client is usually also referred to as Network Access Server (NAS) and can be seen as Policy Enforcement Point (PEP) according to the Policy-based Admission Control Framework specified in RFC 2753 [4]. The RADIUS server generally is the Policy Decision Point (PDP), although it might proxy the request to another RADIUS server, e.g. if the user is a roaming guest.

The basic protocol flow is shown in figure 2.1 on page 6. The RADIUS packets consist of the Packet Type, an Identifier, an Authenticator and a variable number of attributes, which are formatted as Type-Length-Value (TLV). The detailed packet format is described in section 3 of RFC 2865 [3]. The Packet Type can be one of four types:

1: Access-Request, which indicates a request sent to the RADIUS server

2: Access-Accept, which indicates a successful authentication

3: Access-Reject, which indicates a failed authentication

11: Access-Challenge, which indicates that the authentication has not been completed yet

The Identifier is used to match the responses to their corresponding requests as the Identifier of the response is set to the Identifier of the request the packet is a response to. Since not all authentications can be handled in only one request and response, the RADIUS server can send an Access-Challenge packet as response, to which the client responds with another request. In order to be able to identify the requests which are sent in response to a challenge, the RADIUS server may include a State attribute, whose value is then mirrored back in the next request of the client.

The communication partners have a shared secret which is used to authenticate the peer in combination with a MAC mechanism. This shared secret is also used to encrypt a user password if RADIUS is used without additional mechanisms.

Since UDP does not provide an acknowledgment mechanism, which would reveal if packets went missing, RADIUS packets may be retransmitted by the client if the answer of the RADIUS server was not received in time. The contents of the retransmission are equal to the previous packet, thus allowing the server to identify retransmissions.

In a RADIUS Accept message the attributes may contain cryptographic secrets for a WPA2 connection. This implies that the RADIUS packets must not be sent unencrypted over insecure connections. This encryption may be achieved with the RADIUS over TLS protocol, also known as RADSEC, which is specified in RFC 6614 [5]. With this protocol the communication partners establish a TLS connection and send the RADIUS packets through this encrypted tunnel. Since the authentication of the peers is ensured by exchanging certificates during the TLS Handshake, the RADIUS secret is not needed and is set to a fixed value to calculate the value of the authenticator and additional attributes.

### 2.1.2 EAP

EAP (Extensible Authentication Protocol) is specified in RFC 3748 [6]. As the name of the protocol already suggests, it can be used to add different authentication types on top of it. The EAP specification itself only specifies the message format and agreement on the EAP method. It can be used on top of RADIUS and provides mechanisms which are transparent for proxies. Especially in eduroam this is an advantage, since all Identity Providers can choose their own preferred EAP method.

In EAP the terms for request and response are the opposite of the RADIUS terminology. EAP also introduces some new terminology for the different actors. The primary communication partners in EAP are called authenticator and peer. The authenticator is the entity initiating the EAP authentication, the peer responds to the authenticator's requests. The authenticator may relay the EAP communication to the EAP server, which performs the authentication of the peer. In eduroam the authenticator and the EAP server are usually different devices, as the authenticator is the Access Point, the EAP server is included in the RADIUS server.

The peer is also known as supplicant in the IEEE-802.1X standard and is in the eduroam environment usually the device of a user.

EAP can be used for different purposes. In the IEEE 802.1X Standard [7] the EAP over LAN (EAPoL) protocol is defined, which specifies the use of EAP to gain access to a network access port, usually a port

```
RADIUS Client         RADIUS Server
      |                     |
      | Access-Request      |
      |  ID: n              |
      |====================>|
      |                     |
 +-------------------------------+
 |    |                     |    |
 |    |    Access-Challenge |    |
 |    |             ID: n+i |    |
 |    |          State: s_k |    |
 |    |<====================|    |
 |    |                     |    |
 |    | Access-Request      |    |
 |    |  ID: n+i+1          |    |
 |    |  State: s_k         |    |
 |    |====================>|    |
 |    |                     |    |
 |                               |
 | (Optional, possibly repeated  |
 |     multiple times until      |
 | authentication is completed)  |
 |                               |
 +-------------------------------+
      |                     |
      |       Access-Accept |
      |            or       |
      |       Access-Reject |
      |             ID: n+k |
      |<====================|
      |                     |
RADIUS Client         RADIUS Server
```

Figure 2.1: Protocol flow of RADIUS

on a switch. In the IEEE 802.11i Standard [8] the use of EAPoL in WPA2 is specified, this protocol is sometimes referred to as EAP over Wireless (EAPoW), although this is not the official protocol name. The use of EAPoW adds a requirement to the used EAP method, that the method must be able to provide a way of generating cryptographic material which can then be used in the WPA 4-Way-Handshake.

### 2.1.3 EAP-TLS

The EAP methods which are currently most commonly used in the eduroam environment are EAP-TLS based methods. The EAP-TLS protocol is specified in RFC 5216 [9] and defines the use of TLS over EAP. The term EAP-TLS is ambiguous, as it describes the protocol defined in RFC 5216 as well as a specific EAP method where the client authenticates using its own certificate. EAP-PEAP and EAP-TTLS are EAP methods, which build on the EAP-TLS protocol, but only use the server authentication capability of TLS, the created encrypted channel is then used to transmit the authentication messages. In this thesis, EAP-TLS will always refer to the EAP-TLS protocol, unless stated otherwise.

The EAP-TLS protocol is designed to provide a reliable, in-order data stream over EAP, as this is required by TLS. The basic packet structure of an EAP-TLS packet contains a Flags byte, optionally 4 bytes indicating the length of the packet and the TLS payload. As the TLS payload may exceed the maximum fragment size, the EAP-TLS protocol provides a fragmentation mechanism. The TLS payload is split into fragments, which are then transmitted in sequence. After each fragment, the communication partner needs to acknowledge the reception of the fragment by sending an empty EAP-TLS packet without any flags. This ensures the reliable and in-order data stream. Since RADIUS is a strictly Request-Response based protocol, the next fragment can only be sent when the previous fragment has been acknowledged.

### 2.1.4 TLS

The TLS protocol is a security protocol designed to provide integrity, confidentiality and authentication for secure communication over insecure channels.

The current version of TLS used in EAP-TLS is TLSv1.2, which is specified in RFC 5246 [10]. The previous versions are specified in RFC 4346 [11] (TLSv1.1) and RFC 2246 [12] (TLSv1.0). The TLS protocol is an advancement of the Secure Socket Layer Protocol (SSL), which is deprecated. As of the time of writing this thesis TLSv1.3 has already been specified in RFC 8446 [13], but the usage of TLSv1.3 in EAP-TLS is still in the specification process [14]. This thesis will therefore focus on TLS versions from TLSv1.0 up to TLSv1.2. There is no research known to us which proves or refutes that SSL is still used by some actors in the eduroam environment. Since the initial EAP-TLS specification in RFC 2716 [15] was published after the publication of TLSv1.0 and does not specify the use of SSL over EAP-TLS, the use of SSL may be the result of the use of outdated cryptographic libraries. This might be an aspect of future research, but is considered out of scope for this thesis, as stated in section 1.2 on page 2.

The encryption is based on a hybrid cryptosystem, where asymmetric cryptography is used to authenticate the communication partners and key agreement and symmetric cryptography is used for data encryption.

For this, TLS provides a handshake for agreement on algorithms, authentication of the communication partners and sharing of the symmetric cryptographic keys used for the encryption in the data records.

A very basic TLS handshake is shown in figure 2.2 on the left side.

The client initiates the communication with a TLS Client Hello. In the Client Hello the client announces its capabilities to the server. This includes a list of supported cipher suites and a set of TLS extensions.

```
     TLS Handshake without PFS                          TLS Handshake with PFS

Client                          Server          Client                          Server
  |                                |                |                                |
  | * ClientHello                  |                | * ClientHello                  |
  | -> Client Random               |                | -> Client Random               |
  | -> Supported Ciphers           |                | -> Supported Ciphers           |
  | -> Extensions                  |                | -> Extensions                  |
  |------------------------------->|                |------------------------------->|
  |                                |                |                                |
  |              * ServerHello     |                |            * Server Hello      |
  |              -> Chosen Cipher  |                |            -> Chosen Cipher    |
  |              -> Extensions     |                |            -> Extensions       |
  |              * Certificate     |                |            * Certificate       |
  |                                |                |            * ServerKeyExchange |
  |                                |                |            -> DH Parameters    |
  |                                |                |            -> Signature        |
  |              * ServerHelloDone |                |            * ServerHelloDone   |
  |<-------------------------------|                |<-------------------------------|
  |                                |                |                                |
  | * ClientKeyExchange            |                | * ClientKeyExchange            |
  | -> Encrypted PMS               |                | -> DH Parameters               |
  | * ChangeCipherSpec             |                | * ChangeCipherSpec             |
  | * Finished (encrypted)         |                | * Finished (encrypted)         |
  | -> MAC of prev. packets        |                | -> MAC of prev. packets        |
  |------------------------------->|                |------------------------------->|
  |                                |                |                                |
  |        * ChangeCipherSpec      |                |        * ChangeCipherSpec      |
  |        * Finished (encrypted)  |                |        * Finished (encrypted)  |
  |        -> MAC of prev. packets |                |        -> MAC of prev. packets |
  |<-------------------------------|                |<-------------------------------|
  |                                |                |                                |
```

Figure 2.2: Overview of a TLS Handshake

These extensions add functionality to the TLS protocol which might not be included in the initial specification or provide functionality which is not needed in all use cases.

The server then answers with its Hello message, which consists of different handshake records. The Server Hello contains the chosen cipher suite, which has to be one of the cipher suites offered by the client.

The Certificate record contains the server certificate, potentially along with additional certificates needed to verify the server's certificate. The server proves its identity by demonstrating the possession of the private key which corresponds to the publicy key contained in the certificate.

Finally, the server sends a Server Hello Done Handshake record.

After reception of the Server Hello Done, and the verification of the validity of the server certificate, the client answers with a Client Key Exchange record, where the client sends a Pre Master Secret (PMS) to the server. The PMS is encrypted with the public key of the server, so only the server can decrypt the message with its private key.

After the key exchange step every entity has enough data to start the encryption. From the PMS and information from the Client and Server Hello the Master Secret is then derived. This is achieved by using a pseudo random function (PRF), usually a cryptographic hash function like SHA. Which PRF is used is determined by the chosen cipher suite for TLSv1.2 or by the specification of TLS for earlier TLS versions. The Master Secret is then the base for the different encryption and authentication keys.

The client and server indicate the finalization of the key material calculation by sending a Change Cipher Spec record.

After calculation of the Master Secret, the client and server exchange a Finished message, which contains a MAC of all previous packets. The Finished message is already encrypted and authenticated by using the chosen cipher suite. This step prevents a downgrade attack, where a middle person tries to degrade the encryption, e.g. by deleting strong cipher suites from the supported ciphers in the TLS Client Hello.

This basic version of a TLS handshake is sufficient to reach all described security objectives of TLS (authentication, confidentiality, integrity), as long as the private key of the server is secret. If the private key of the server is compromised, a previously recorded TLS communication can be decrypted afterwards. To mitigate this problem, in contrast to the previous described handshake, the Pre Master Secret is not sent directly by the client, but determined by a key agreement algorithm such as Diffie-Hellman.

Diffie-Hellman as one key agreement algorithm is based on private and public parameters on both sides. The parameters are regenerated for each connection.[1] The public parameters are exchanged by the communication partners. After exchanging the public parameters, the communication partners can derive the same result using their private parameters and the public parameters of the communication partner. This result then can be used as Pre Master Secret in TLS. Since the knowledge of the private parameters is needed for calculating the shared key, an eavesdropping attack does not reveal enough information to calculate the key.

A version of the TLS handshake including Diffie-Hellman is shown in figure 2.2 on the right side.

To be able to use Diffie-Hellman in the handshake, the client indicates support by choosing respective cipher suites and including a TLS extension indicating support for different Diffie-Hellman variants. The major changes in the protocol are in step two and three. In step two, in addition to the Server Hello and Certificate, now a Server Key Exchange is also transmitted. The Server Key Exchange contains the server's Diffie-Hellman parameters. To ensure authenticity of these parameters, the server cryptographi-

[1] Static Diffie-Hellman parameters are also possible, but since this does not solve the problem of subsequent decryption, this case is not considered here.

cally signs these parameters using its private key. The client can then authenticate the parameters using the public key contained in the certificate.

In step three, the client sends its Diffie-Hellman parameters, together with a Change Cipher Spec record, since it has now enough information to derive the Master Secret. The client's Diffie-Hellman parameters do not need to be authenticated or encrypted, due to the structure of the Diffie-Hellman algorithm. Its authenticity and integrity is proven by the Finished messages. If the parameters had been modified, the result of the Diffie-Hellman calculation would differ for the communication partners, resulting in Finished messages the other cannot decrypt.

After calculation of the Pre Master Secret, the private Diffie-Hellman parameters are discarded. A previously recorded TLS communication with a Diffie-Hellman key exchange cannot be decrypted, since the knowledge of the private parameters is needed to calculate the shared secret.

## 2.2 The eduroam infrastructure

eduroam (short for education roaming) is a federated infrastructure designed to give researchers internet access around the globe. This purpose is primarily achieved though access to a WPA2-Enterprise network, but can also be used for wired access (802.1X). The eduroam infrastructure consists of different entities, which will be explained in this section.

To get an overview of the general setup, we start with a local deployment of eduroam as WPA2-Enterprise network.

In contrast to WPA2-PSK (Pre-Shared-Key) wireless networks, where the authentication and authorization is achieved by a shared secret, WPA2-Enterprise allows an individual authentication and authorization. It is especially used in enterprise environments, where the usage of a shared secret is out of question, e.g. because the network is used by thousands of users, and the access should be individually revocable. Additionally, WPA2-Enterprise allows a distinguished authorization for different wireless parameters such as bandwidth, limitations of the usage time span or the VLAN the client should be connected to.

In the following sections, the shown Identity Providers will always use an EAP-TLS based authentication method, since this method is the subject of this thesis.

```
  +--------------+               +--------+
  | Wired Client |<=============>| Switch |----------+
  +--------------+               +--------+          |         RADIUS server
                                                     |   backend authentication server
    Supplicant,             Network Access Server    |         EAP-TLS server
  EAP-TLS Client                Authenticator        |
                                RADIUS Client        |
                                                     |     +------------------+
+-----------------+           +--------------+       +---->|   RADIUS server  |
| Wireless Client | <-------->| Access Point |------------>|   e.g. running   |
+-----------------+           +--------------+             |    FreeRADIUS    |
                                                             +------------------+
```

Figure 2.3: Simple WPA2-Enterprise deployment

### 2.2.1 Terminology in WPA2-Enterprise and Single Site Deployment

A simple deployment consists of three different component classes, as shown in figure 2.3:

The **supplicant** is the device of the user. It is also acting as **TLS Client** in the EAP-TLS phase. Inside EAP-TLS this device is referred to as **peer**.

The supplicant connects to the **Access Point (AP)**. The AP starts the authentication process by sending the first EAP request as **authenticator**. After reception of the first EAP response it starts a RADIUS communication as a **RADIUS Client**. The AP is also referred to as **Network Access Server (NAS)**. In wired deployments the NAS could also be a switch, where connections to the physical ports require a login.

APs are in most university deployments connected to a controller, which e.g. coordinates selection of the radio band and handles the exchange of key material on a handover to another access point in the same domain. In terms of the WPA2-Enterprise structure and the underlying protocols for EAP, the Access Point and Controller do not need to be distinguished. When referring to communication of Access Points in this thesis, it might also mean that not the Access Point itself but the connected controller is the communication partner.

The **RADIUS server** is then replying to the NAS. EAP refers to this entity also as **backend authentication server**. In most cases, the RADIUS server also acts as **EAP server**, since the software answering the RADIUS requests also handles the EAP protocol directly.

The protocols are used as shown in figure 2.4 on the following page. The AP and supplicant initiate an *EAP* communication. Depending on the link this is either *EAP over Wireless* for WPA2-Enterprise networks or *EAP over LAN* for 802.1X. The NAS then starts a *RADIUS* communication with the RADIUS server. The figure shows a federated deployment; in a local deployment, the request is not proxied and the initial RADIUS communication is between the AP and the RADIUS server of the IdP. The *EAP* messages are encapsulated in RADIUS attributes. After establishing the EAP-Communication on top of EAP the *EAP-TLS* protocol is used, which is a container for *TLS*. TLS then acts as a transparent tunnel between the peer and EAP server, inside which other protocols like MS-CHAP-V2 or PAP can be used to complete the authentication.

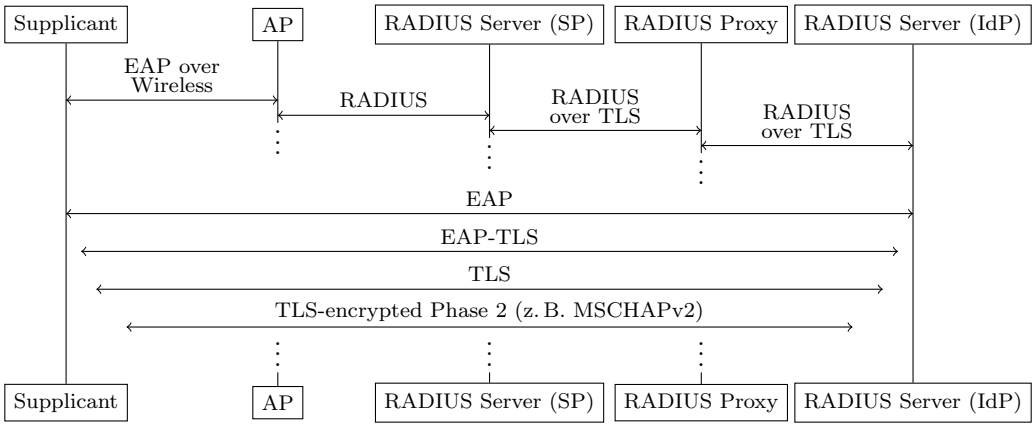


Figure 2.4: Protocol stack in a federated WPA2-Enterprise deployment
*This model uses RADIUS over TLS for proxying between the SP and IdP servers, as deployed for the DFN. This could also be plain RADIUS if the connection is considered secure.*

### 2.2.2 The Federated WPA2-Enterprise Network eduroam

eduroam is not a single-site setup, but a network of several research facilities. In eduroam the entities holding the account information and executing the authentication are called *Identity Providers (IdP)*. The network can then be accessed at every entity providing an eduroam network, in eduroam also called *Service Providers (SP)*. The basic infrastructure design and some basic infrastructure decisions are published in RFC 7593 [16].

In order to give access to users of an Identity Provider at a different Service Provider, while keeping the credentials secret, the access requests must be proxied from the SP to the IdP. The proxying can be done in several ways and through different protocols. eduroam uses a hierarchical structure, as shown in figure 2.5 on the next page. The local IdPs and SPs are connected to a *National Roaming Operator (NRO)*. The NROs are usually also the national research network operators, for Germany this is Deutsches Forschungsnetz e. V. (DFN). The NROs have then a connection to the top-level RADIUS servers, and they may as well be interconnected to reduce the load on the top-level RADIUS servers.

The connection between the different actors is usually done via the RADIUS over TLS protocol.

For choosing the correct proxy destination of a request the realm part of the user name is used. The realms are usually based on the DNS domain of the facility. A user from the University of Bremen will use a user name like `rieckers@uni-bremen.de` as their user name. An SP usually proxies all requests it gets to the connected NRO. If the SP is also an IdP, the requests for its realms will be handled completely locally. In the given example, the RADIUS server for the University of Bremen would handle the request with the aforementioned user name itself. If the user would connect to a network of a different service provider, e.g. the Phillips-University of Marburg, the request would be proxied to the DFN. The NROs then hold a list of realms which are connected locally. If the realm is not in the list of known realms, the request will be proxied again. By using the DNS names, the next proxy can be determined by the top level domain (TLD) of the realm, if it is a country-specific domain. For this to work the NROs have to have a connection between them. If this is not the case or the realm has a generic TLD (gTLD, e.g. `.eu` or `.edu`), the next proxy is the federation-level proxy. The federation-level proxy is connected to all NROs and holds a list of their assigned TLD and gTLD realms.

With this federation infrastructure an eduroam user can access eduroam at any connected SP. The addition

of new SPs is trivial, since only a connection to the NRO has to be established. A new IdP can be easily deployed, if a country TLD is used. For the addition of a gTLD domain the federation proxy operators need to be informed.

```
Federation Level
                  +--------------------+
                  |     Radsecproxy    |
                  |        GÉANT       |
                  | (Federation Proxy) |
                  +-A--------A-------A-+
                    |        |       |
      +-------------+        |       +---------------+
      |                      |                       |
      |                      |                       |
      |       +---------+--- | -------------+        |
      |       |         |    |              |        |
National Roaming Operators   |              |        |
      |       |         |    |              |        |
  +---V-------V-+     +-V----V------+     +-V--------V-----+
  | Radsecproxy |     | Radsecproxy |     | Other national |
  |     DFN     |     |     SURF    | ... |    research    |
  |     .de     |     |     .nl     |     |    networks    |
  +--A---A------+     +---------A---+     +---------A------+
     |   |                      |                   |
     |   +-------------+        +-------+           +----+
     |                 |                |                |
Local Service/Identity Providers        |                |
     |                 |                |                |
+----V-----------+   +-V---------+  +---V-------+     +--V----+
|    RADIUS +    |   |    ...    |  |    ...    |     |  ...  |
|   Radsecproxy  |   |           |  |           | ... |       |
| @uni-bremen.de |   | @<...>.de |  | @<...>.nl |     |       |
+----------------+   +-----------+  +-----------+     +-------+
```

Figure 2.5: WPA2-Enterprise Roaming Infrastructure

# Chapter 3

# Capturing and analysis process overview

This chapter will give an overview of the design and functionality of the capturing and software stack of the analysis tool. The target of the tool is to perform an analysis of the used protocols. This is achieved by a capture of all RADIUS authentication operations. The information gained is then processed and stored for a later statistical analysis.

## 3.1 History of Development

The analysis tool was written by the author before starting this thesis. An early version of this tool with limited capturing abilities was written in C. Capturing results of this early version have been used in a submission to an IAB workshop [17] and a presentation at an operational meeting of the DFN [18].

The current version is written in Ruby. Results of this version have already been presented, too. A basic overview of the capabilities and first results have been presented at an operational meeting of the DFN [19]. The basic structure of the tool and some results and conclusions have been published at a DFN journal [20].

During the processing time of this thesis, small additional features of the analysis tool have been implemented. Additionally, the data was processed to enable simpler and more efficient analyses. This was done by modifying every data item and adding data points derived from already stored data. The added features include primarily statistics output to assess possible errors present in the analysis tool and a few performance improvements necessary to process the large quantity of input at the DFN.

## 3.2 Software Stack

The software stack of the analysis tool is divided in the three parts capturing, data processing and storage and statistical analysis.

### 3.2.1 Capturing

The capturing part provides the input of the analysis tool.

The RADIUS packets can be captured at different points inside the eduroam infrastructure.

**RADIUS Packets over raw socket**

The first option to capture packets is via a raw socket. Using the gem packetfu[1], which uses the Unix library `libpcap`, the complete traffic on a specific interface is captured. These captured packets are then filtered for UDP packets on port 1812, which is the standard port for RADIUS.

Every captured packet is then parsed and inserted into an internal packet queue, where a second thread matches the packet to a previous captured packet of the RADIUS communication.

This capturing process allows an analysis by simply wiretapping. To separate the analysis infrastructure from the production system, the packets can be simply copied and sent to the analysis host, e.g. by using the `iptables -j TEE` option.

The RADIUS capturing option can be used by Service Providers or Identity Providers to analyze their respective clients.

**Capturing inside Radsecproxy**

The second option to capture packets is via a capturing interface inside the software radsecproxy. As described in subsection 2.1.1 and section 2.2, radsecproxy is used to proxy RADIUS requests and responses between service and identity providers. Although radsecproxy is capable of handling RADIUS over UDP or Radius over TCP, which would allow a simple analysis by wiretapping, the majority of communication is done via RADIUS over TLS. The encryption provided by TLS rules out the possibility of wiretapping.

In order to still be able to analyse the packets proxied though, the source code of radsecproxy has been extended. The changes have been published on Github[2].

The modified radsecproxy opens a Unix socket, to which the analysis tool can connect. After the analysis tool connects, a copy of all proxied RADIUS packets is sent to the Unix socket.

In order to be able to match a RADIUS packet to a previous RADIUS communication, the socket output contains some additional information. This includes the source and destination of the packet and the information if the packet is a request or response.

To prevent a disruption of the production environment by the analysis tool, the output of the packet copies is realised with a non-blocking write. If the analysis tool is not able to handle the number of packets sent by radsecproxy, packets are silently discarded.

In the analysis tool, the packet and its additional information are parsed and inserted into the internal packet queue.

[1] `https://rubygems.org/gems/packetfu`
[2] `https://github.com/Janfred/radsecproxy/tree/DFN/outputsocket`

### 3.2.2 Data processing

The data processing inside the analysis tool is separated in three different parts

- Packet matching
- RADIUS communication analysis
- Finalization before storage

To improve the performance and add scalability, the steps are done in separate threads.

**Packet matching**

In this first step of the analysis process, the previously captured packets have to be matched to a complete RADIUS communication.

Depending on the capturing method this takes into account the source and destination IP address and UDP port for the raw capturing or the client/server information for the capturing inside radsecproxy.

The RADIUS packets contain an identifier, which maps a response to a request. This information, in addition to the source/destination information, can be used to match a response to a previous request.

Since the EAP-TLS authentication cannot be completed in only one request and response roundtrip, the RADIUS communication consists of several roundtrips. As the RADIUS packet structure itself does not provide a mechanism for matching a previous response to a following request, additional means have to be used. In the RADIUS standard the use of the RADIUS attribute State is specified [3, section 5.24]. The State attribute is set by the RADIUS server and is mirrored back by the RADIUS client when answering the server's response. The value of the State attribute in responses is saved when inserting a response into the temporary storage object for the ongoing RADIUS communication. To match a request to a previous response, the analysis tool can now simply search for a response with the respective State attribute value.

Unfortunately for the analysis tool, the use of the State attribute is not mandatory for RADIUS servers. If a State attribute is not set, the matching of a RADIUS request to a previous response has to be done by other means. This is achieved by matching other RADIUS attributes which are set in every request by the RADIUS client. This includes the attributes User-Name and Calling-Station-Id.

Limitations of this matching method are discussed in section 3.4 on page 20.

Upon reception of a final RADIUS packet like an Access-Accept or Access-Reject, the data object containing the RADIUS communication is inserted into the parsing queue. If a communication did not receive a packet for a period of time, currently set to ten seconds for the radsecproxy capture and 60 seconds for the raw socket capture, it is assumed that the communication has timed out and the previously captured packets are discarded without parsing.

**RADIUS communication analysis**

After the packets have been matched to a complete RADIUS communication, the communication, in the analysis tool referred to as stream, is parsed. This parsing is following the protocol layers present in the RADIUS stream.

Initially the RADIUS packets are analyzed and meta information such as size of packets and total number of packets are extracted. If the RADIUS packets include EAP Message attributes, the attribute values are concatenated to an EAP message.

Once the RADIUS parsing is finished, the EAP communication is parsed. This consists mainly of parsing the initial EAP handshake, where the EAP identity is transmitted and the EAP method is negotiated. The

following EAP messages are copied and passed on to the next parsing step.

Depending on the chosen EAP method, the parsing continues. As this thesis focuses on EAP-TLS based methods, only these methods are parsed in the tool. In future versions of the analysis tool this might be extended to other EAP methods like TEAP, a different TLS based EAP authentication method, or EAP-PWD, which allows for mutual authentication using a shared secret without need for certificates.

The EAP-TLS protocol contains a fragmentation functionality, as the specification of TLS requires the underlying transport to be a reliable in-order data stream. In order to parse the TLS records afterwards, the analysis tool first merges the EAP-TLS fragments to EAP-TLS messages, which then can be parsed by the next step.

Finally, the TLS packets are parsed. This contains the parsing of TLS Client and Server Hello, the Certificate and Server Key Exchange. The information present in these TLS handshake messages is extracted.

**TLS Client Handshake messages**

For the **TLS Client Hello** record, the following information is extracted:

**TLS version** – This signifies the maximum TLS version the client supports[3].

**Cipher suites** – This is the list of cipher suites the client supports.

**Compression** – The client indicates its support for different compression methods here.

**Extensions** – With extensions the client can indicate support for additional functionality or send additional information needed for the connection. Since not all TLS extensions are relevant for the security analysis, only the content of the following are analyzed.

**RENEGOTIATION_INFO** – This extension indicates the client's support for secure renegotiation as defined by RFC 5746 [21].

**SERVER_NAME** – This extension allows the client to indicate the host name it wants to connect to. This extension is primarily used in HTTPS, where many different websites share a single IP address. It is defined in RFC 6066 [22].

**EXTENDED_MASTER_SECRET** – This extension indicates the client's support for an improved calculation of the Master Secret used for encryption. It is defined in RFC 7627 [23].

**SUPPORTED_VERSIONS** – This extension was defined for TLSv1.3 in RFC 8446 [13] to indicate support for TLSv1.3. As this thesis does not consider TLSv1.3 it is only parsed to determine if the client seems to be capable of TLSv1.3.

**SUPPORTED_GROUPS (previously ELLIPTIC_CURVES)** – With this extension the client sends a list of supported elliptic curves and Diffie-Hellman groups for the Diffie-Hellman key agreement. This extension was initially defined in RFC 4492 [24] and later updated by RFC 7919 [25] and RFC 8422 [26].

**STATUS_REQUEST** – This extension indicates the client support for OCSP Stapling, where the server sends a proof of validity along with the certificate. This method is intended to be used in environments like EAP-TLS where the client does not have an active Internet connection (yet), so the revocation status of a certificate cannot be checked against OCSP servers of the CA. The extension is defined in RFC 6066 [22].

**SIGNATURE_ALGORITHMS** – This extension was added for TLSv1.2 to indicate supported algorithms for the signature of the Diffie-Hellman server parameters.

[3] or will accept in this communication

The choice and order of cipher suites, the supported extensions, ECDHE curves or signature algorithms primarily depends on the TLS implementation used. This circumstance can be used to fingerprint the different TLS implementations and library versions used by different operating systems. The fingerprinting might not be very accurate, as different devices might have a similar TLS fingerprint. Nonetheless, this information can be used to obtain an overview of the variety of devices present in eduroam.

Some of these fingerprints were obtained by us ourselves, e.g. by observing the behavior of own devices. A large number of known fingerprints originate from captures provided by the University of Marburg, where Yassin Alshater performed a separate analysis of the behavior of EAP-TLS clients, focused on verification of server certificates.[27]

**TLS Server Handshake messages**

For the **TLS Server Hello**, **Certificate**, **Server Key Exchange** and **Certificate Status** records the following information is extracted:

**TLS Server Hello** – This record is the first record sent by the server and contains the encryption method agreement.

**TLS version** – This is the TLS version actually used. It must not be higher than the version offered by the client.

**Server random** – The server random is used for generating the Master Secret, but might also include specific signaling values, if the TLS version supported by the server is higher than the version requested by the client. This is used to prevent downgrade attacks.

**Chosen Cipher** – This is the cipher suite chosen by the server, indicating which algorithms will be used for encryption, authentication, key agreement and integrity checks.

**Compression** – This defines the compression method to be used for the connection.

**Extensions** – Like in the Client Hello, the server now indicates support for functionality. The server must only send extensions the client has sent previously (with one exception). The content of the following extensions is parsed:

**RENEGOTIATION_INFO** – This indicates the server's support for secure renegotiation. This extension might be sent without the extension being present in the Client Hello, if the client indicated support for secure renegotiation by sending a Signaling Cipher Suite Value (SCSV).

**EXTENDED_MASTER_SECRET** – This extension indicates the server's support for the improved calculation of the Master Secret.

**SUPPORTED_VERSIONS** – For TLSv1.3 this extension contains a specific value, indicating TLSv1.3 although the Server Hello appears to be a TLSv1.2 Server Hello. Since TLSv1.3 is out of scope of this thesis, the parsing might not be accurate.

**Certificate** – The Certificate record contains the server certificate as well as additional certificates to validate the server certificate. The analysis tool parses all sent certificates and includes the subject fields in the output data. The certificates themselves are stored separately for later analysis.

**Server Key Exchange** – This record contains the Diffie-Hellman parameter for the Diffie-Hellman key agreement. Two pieces of information are parsed:

**Named Curve** – This is the ECDHE curve used for the Diffie-Hellman key agreement algorithm

**Signature Algorithm** – Here the used signature algorithm is indicated. This field is not present in TLSv1.0 or TLSv1.1, as the signature was always done by using SHA-1 as hash algorithm and the encryption algorithm was determined by the certificate. In TLS from version 1.2 the signature of the ECDHE parameters can be done by a variety of algorithms.

**Certificate Status** – This record contains an OCSP staple to prove the validity of the presented certificate. In the current version of the analysis tool, only the presence of this record is determined, not its content.

After encountering a **Change Cipher Spec** record, the following communication is encrypted. This marks the finish of the parsing.

Once the parsing of the protocols present in the stream has been finished, the results are inserted into the storage queue to be picked up by the storage thread.

### 3.2.3 Storage and statistical analysis

The final step inside the analysis tool is the storage of the previously parsed data.

To ensure a unique identification as well as privacy of the users, a fingerprint is calculated over the user name and MAC address which then serves as an identifier. Details of this calculation are discussed in section 3.5 on page 25. After the calculation of the fingerprint, some data items are deleted or anonymized.

Additionally, based on the parsed data, other data items are added. First, the device vendor is determined based on the first three bytes of the device's MAC address. Second, the TLS fingerprint calculated by the previous step is looked up and the corresponding information is inserted. A detailed overview of the complete data structure can be found in appendix C

After processing, the data is then inserted into the database. An Elasticsearch server[4] is used for data storage. Elasticsearch was chosen as it allows storage and efficient analysis of structured data. The Elasticsearch infrastructure contains the frontend Kibana, which allows for a visual analysis of the stored data.

## 3.3 Experiment setup

The data basis used in this bachelor thesis was obtained from two different capturing points.

The first capturing point is located in the network infrastructure of the University of Bremen. The local RADIUS servers handle all authentication requests for clients on campus as well as proxied requests for members of the University of Bremen visiting other institutions. In order to prevent disruption of the production environment, the analysis is performed on a separate server.

To send the data to the analysis machine, the RADIUS packets are copied using `iptables`. An example rule set can be found in figure 3.1 on the following page. This rule set copies all packets sent to or from UDP port 1812 to the analysis server. The UDP port 1812 is the default port for RADIUS as defined by RFC 2865 [3, section 3], which the servers at the University of Bremen use.

On the analysis server, the analysis tool is running with the capturing over the raw socket. Using a raw socket enables the tool to capture all packets sent to the server, regardless of the original destination of the packet. Since the packets are simply duplicated and then sent to the indicated server, the original IP address also remains intact. This is needed for a reliable assignment of packets to different ongoing communications, especially if multiple RADIUS servers send their data to the analysis machine.

Since the destination IP address of the packet does not match the IP address of the analysis server, it must not have IP forwarding enabled. This would lead to the analysis server resending the packets to the

[4] https://www.elastic.co/elastic-stack/

original server, leading to an infinite loop.

This capturing point was the initial source of data for the initial version of the analysis tool and has been running for approximately two years now. The data captured over the time has been erased in May 2020, when the current major version of the analysis tool was deployed.

```
# IP-Addr of the RADIUS server: 192.0.2.1
# IP-Addr of the analysis server: 192.0.2.10
-A PREROUTING  -d 192.0.2.1 -p udp -m udp --dport 1812 -j TEE --gateway 192.0.2.10
-A POSTROUTING -s 192.0.2.1 -p udp -m udp --sport 1812 -j TEE --gateway 192.0.2.10
```

Figure 3.1: Example IPTables-Rules for the capturing at the University of Bremen
*Extract from output from* `iptables-save`

The second capturing point is located in the network infrastructure of the DFN. As described in subsection 2.2.2, the DFN operates the federation proxies for all IdPs and SPs located in Germany. Again, just like at the University of Bremen, the analysis is performed on a different server.

To capture the needed data, the modified version of radsecproxy mentioned in subsection 3.2.1 is running on the production servers. The radsecproxy software then opens a UNIX socket, to which the analysis tool can connect to. This is achieved by using an SSH tunnel from the analysis server to the production servers. The analysis tool connects to the forwarded UNIX socket on the local machine.

This capturing setup has been running since July 2020, initially only with a small number of clients. Since October 2020, the capturing is running on all production servers.

To ensure a consistent data basis, the captured data has been exported to an isolated system, where the statistical analysis for this thesis has been performed. The time range has been set to 2021-01-01 to 2021-07-20. Unless stated otherwise, the statistical analysis always refers to this time range.

## 3.4 Statistical relevance and known errors

Statistical analyses are carried out in the later chapters of the bachelor thesis. This section will address the statistical relevance of the data used in the later analysis, as well as it will point out possible sources of errors or imprecision.

### 3.4.1 Captured parts of the RADIUS communication and TLS handshake

The analysis tool captures all RADIUS communication. For the RADIUS, EAP and EAP-TLS layers, all packets are captured and analyzed. For the TLS layer inside EAP-TLS, only the unencrypted handshake messages can be analyzed. Every message after the CHANGE_CIPHER_SPEC record is ignored in the parsing process, since it is encrypted.

Although all TLS communications are captured, only those with a complete TLS handshake are included in the output of the analysis tool. This behavior entails that clients offering TLS parameters which are incompatible with the TLS configuration of the server are not included in the data. This will affect primarily old devices which are not capable of the encryption level enforced by the servers, e.g. in terms of TLS version or offered Cipher Suites.

This behavior was decided in the early development stage of the analysis tool, to prevent data contamination from incomplete handshakes, where a previous completed handshake has been captured. Especially

in the WLAN environment, an aborted/timed out handshake is possible if the client is at the edge of radio coverage at the time of login.

Due to the fact that only completed handshakes are included in the data, the analysis performed in this thesis is susceptible to confirmation bias, especially when estimating compatibility of security measures with existing clients.

A separate analysis of only TLS Client Hellos might be included in a future version of the analysis tool.

### 3.4.2 Capturing difference between University of Bremen and DFN

The analysis tool captures data from two different locations with different capturing profiles.

At the University of Bremen the tool captures all RADIUS traffic handled by their RADIUS servers. At the DFN the tool connects to the federation level proxies, which forward all roaming traffic from and to German IdPs and SPs. This difference has quite a number of implications to the statistical relevance of the data.

**Number of clients and servers**

The first and foremost difference in the capture is the number of captured clients and observed servers/IdPs. The federation servers of the DFN forward around one million authentications daily, where the University of Bremen only has around eight thousand authentications.

This leads to a huge difference in the unique clients that have been observed (approx. 1.4M at DFN, 29k at Uni).

Since the capture at the federation server is the central router for all roaming traffic in Germany, there is also a huge variety of realms and IdPs present in the data. At the university only realms of those IdPs are present, whose members were roaming visitors at the university. For the University of Bremen these have been mostly members of the other research and teaching facilities in Bremen. Especially due to the pandemic situation, there is a very limited number of other realms. Over the specified time range at DFN 5,531 realms have been observed, where at the University only 767 realms have been seen.

**Client profile**

Due to the different location of the capturing, the client profile differs between the university and the DFN data.

The capture at the university captures all requests to the RADIUS server, therefore all clients are analyzed, whether they are on- or off-campus. This gives a good overview of the actual client distribution in the IdP network.

In contrast, the capture at the DFN only captures roaming clients. The clients are logging in at an SP network and the request is proxied to their respective IdP. This causes a distorted client ratio, because mostly mobile clients are captured. Especially at locations where many SPs and IdPs exist, the devices of the members will automatically attempt a login when the eduroam network comes in range. This has been observed by us even during train rides through Berlin. Clients, which stay at the IdP network, e.g. desktop computers, or clients, which are not always active, like laptops, will not be captured.

In figure 3.2 on the next page the ratios of the client operating systems are shown. For Apple the fingerprinting does not allow a differentiation between phones, tablets or laptops. Since the fingerprinting database does not contain a lot of fingerprints for Windows or Linux, we assume that a larger proportion of the unknown devices may be Windows or Linux machines.

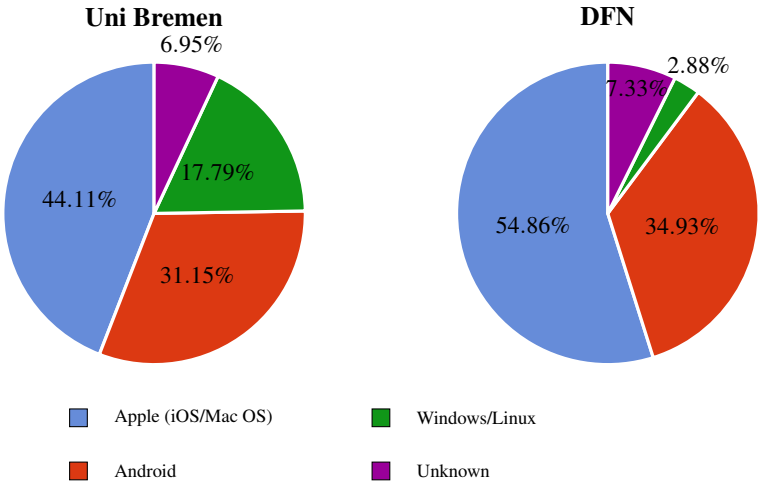


Figure 3.2: Comparison of captured client operating systems
*Sample Size: 28,980 (Uni), 1,421,345 (DFN)*

### 3.4.3 Multiple capture of the same client

In order to have substantive data, multiple authentications of the same client should not result in multiple database entries.

Multiple database entries might be useful for assessing the actual impact of online attacks, since they are commonly more effective if the attacked communication happens more often. For the target of this thesis, an overall security analysis of the eduroam infrastructure, this behavior is not desired.

In order to limit the entries of the same client to one, the clients must be identified by the capturing and their entry in the database updated. This is achieved by using the MAC address. Transmitting the MAC address is a requirement of the eduroam Service Policy [28]. Privacy implications of this method are discussed in section 3.5 on page 25.

This method still produces errors, if the client changes its MAC address. To preserve the privacy of their users, the operating system manufacturers have introduced random MAC addresses for the wireless connection. Depending on the implementation in the operating system, the MAC address is stable for the same SSID. This would allow the identification of the same device, as long as the same SSID is used. Some SPs add a suffix to the eduroam SSID, e.g. if multiple SPs are in close proximity. In this case a device might be added multiple times to the database. If the MAC address is changed randomly, even with the same SSID, the identification method cannot recognize the device. It would then be introduced to the database again each time the MAC address is changed.

To assess the impact of this error, the logs of the RADIUS server at the University of Bremen have been analyzed twice, each time over a period of one week. This analysis has shown that not more than 40 out of 10,780 users ($<0.4\%$) with a total of 312 of 14,640 MAC addresses ($\approx 2.13\%$) seem to use randomized MAC addresses.[5]

The effect of randomized MAC addresses needs to be considered especially for special cases, where the device in question has a unique TLS behavior. This case can be suspected in the case of an unusually high

[5] It is assumed, that a user may possess up to four devices, therefore from five MAC addresses per user it is assumed the MAC address is randomized.

number of identical TLS fingerprint hashes not present in the fingerprint database while other fingerprints only occur in small numbers.

In addition to the randomized MAC addresses of the devices, an additional error source is an incorrectly transmitted MAC address. Some SPs, in violation of the eduroam Service Policy, do not include the client's MAC address in requests. In this case, the analysis tool is unable to map the current authentication to a previous captured authentication. It then falls back to a default MAC address. In addition with a predefined outer identity, this might result in capturing different devices under the same identifier.

### 3.4.4 Unmapped requests and capturing errors

To increase the confidence in the data, the analysis tool contains some statistics features. It records the number of packets captured, analyzed, filtered and written to the database, as well as statistics about the number of EAP streams and error causes.

Here the capturing has shown some irregularities that we cannot explain fully at this time.

A known limitation in the analysis tool is the handling of retransmission. Currently, the analysis tool does not contain any methods to recognize a RADIUS packet retransmission, which can occur for several reasons, especially if the supplicant operates at the edge of the wireless range of the access points and some packets are dropped. Currently, the analysis tool treats these messages as normal messages, which implies that a new communication stream is created, if the packet is a RADIUS Request. This can lead to split communications, where answers to a retransmitted packet are added to the newly created communication instead of the original one. Though this bug might explain some errored packets and communications, this does not explain all errors.

In a testing setup with one radsecproxy connected to 10 different RADIUS-Servers with a throughput of roughly 100 Access-Accepts per second in total, only about 85% of the requests were correctly analyzed. The remaining 15% of requests have either timed out or were matched incorrectly. Although some unmapped requests might be explained by a buildup in the queues for captured packets and streams to be analyzed, the assumption remains that the matching algorithm still contains flaws.

This observation is substantiated by the statistics generated by the analysis tool on the DFN radsecproxy servers. Figure 3.3 on the next page shows an extract of the generated statistic of captured, analyzed and timed out packets. The figure shows, that approximately 25% of all packets are not matched to a complete communication stream and are timed out. A small number of these packets might be retransmissions which are matched incorrectly or the communication is aborted due to the client leaving the radio coverage.

Due to the high throughput of packets at the radsecproxy servers in combination with time constraints, a manual analysis was not performed. This might be conducted for a future version of the analysis tool.

Especially under higher load the analysis tool has crashed several times, as can be seen in the statistic on Wednesday, which is suspected to be a result of a limited calculation capacity.

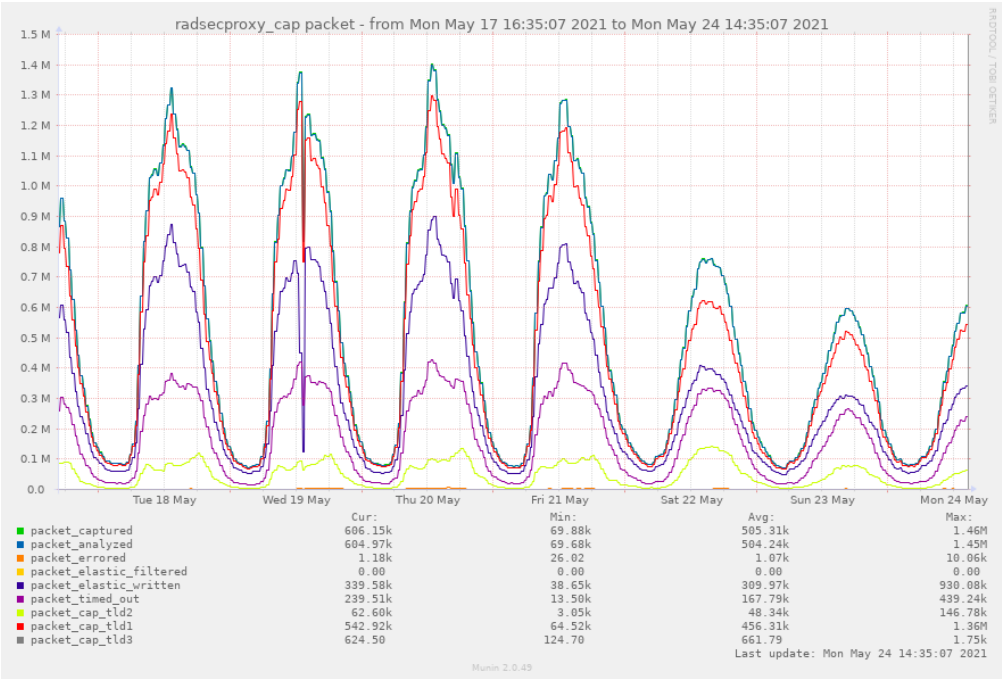


Figure 3.3: Extracted statistic of captured, analyzed and timed out packets
*Capture details of DFN, time range: 7 days*

### 3.4.5 Conclusion of the statistical errors

Despite the aforementioned known errors and inaccuracies, the data used can be considered clean enough to perform analyses on it. Especially the large number of clients lead to a sufficient representativeness of clients active in the eduroam environment. The large number of connected realms with different security settings counteracts the mentioned confirmation bias regarding outdated TLS clients, although it does not rule it out completely.

The capturing differences between the two capturing instances will be considered especially when making assumptions based on realms. Those assumptions are only performed for the DFN data, since the number of realms present in the University data is not enough to provide representativeness, especially because most realms have been observed with only one or two clients.

The output provided by the debug logs and additional statistics has not shown that a structural problem exist, that would result in the loss of data for a large number of clients or clients with special behavior.

## 3.5 Privacy considerations

The described capturing process allows a unique identification of users. Since this enables the capturing entity to create movement profiles or the knowledge of security issues with user's devices may be used, the captured data has to be treated carefully. This is based on moral research on the one hand, and the EU regulations of the GDPR (General Data Protection Regulation) on the other.

In general, in the processing and storage of the data two requirements have to be weighed against each other. The first requirement is the recognition of the same device. When a client reconnects, e.g. because it was shortly out of range, there must not be a second entry in the database. This would distort the data and a qualitative analysis would no longer be possible. The second requirement is the preservation of the user's privacy. A direct assignment between user and device is not necessary and should therefore, if possible, be avoided.

Practically, the dissociation between user and device is achieved by introducing a stable pseudonym. The pseudonym is generated through a cryptographic hash of the complete user name and the MAC address of the device as transmitted in the Calling-Station-Id attribute. After calculation, the local part of the user name and the last three octets of the MAC address are stripped. The pseudonym is stable per device, which ensures that the device exists only once in the database.

Calculating this pseudonym without a salt or pepper is either not possible or still opens up the possibility of a brute force attack. A salt, an individual random string to be added to each data before hashing, would not be usable, since the result of the hash is used as identifier and this result would change when choosing a random string. Searching through all previous captured data to recalculate the hash for each possible salt is not practical. A pepper, a static random string to be added before hashing, is usually configured inside the software. This would add some difficulty to reverse the calculation for attackers, who obtained the captured data, but since the pepper is known to the capturing entity, it could still reverse the calculation easily. The possible value range for the last three octets of the MAC address is limited ($2^{24}$), so finding possible devices to a given user name is a simple task. The other direction, finding out the user name of a given MAC address, is not easily achieved, since the value range for a Brute force now also includes all possible user names.

Since some IdPs use a static outer user name (e.g. `anonymous@REALM` or `eduroam@REALM`), finding a device by their MAC address is possible, as it must be to ensure the first requirement. Adding a pepper to the hash calculation solves this problem, if the pepper is not known to an attacker. In any case, this pseudonym would still allow the capturing entity to identify a user. Since the probability of attacks by external attackers was considered low, the pepper mechanism has not been implemented. This may be done in future versions of the analysis tool.

In order to increase the privacy while still keeping data integrity, after a period of time, the data could be completely anonymized. The pseudonym could be replaced by a random identifier, which then does not allow the mapping to a specific user or MAC address. Since this random identifier, as required, does not allow recognition of the same device, the period of time after which the pseudonym is replaced with the random identifier requires weighing between privacy of the users and duplicate client detection.

For analysis purposes, the replacement of pseudonyms with random identifiers has not been activated, for a long-term production environment we would recommend a period of three months. This would most likely cover the lecture-free period between semesters, so devices of students, who are not at campus for this period of time, will not be introduced twice.

# Chapter 4

# Analysis of EAP-TLS handshakes

In this chapter, the captured data will be analyzed to determine if structural security problems exist.

As guideline to assess the security implications and deviations from current best practices, two main documents will be used: RFC 7525 [29] was published in May 2015 and gives recommendations for a secure usage of TLS. The German Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI) issued a technical guideline for the use of TLS with recommendations for cryptographic algorithms, their key lengths and additional TLS parameters [30].

The analysis will be performed separately for the different captured data. Afterwards an overview of the overall security in eduroam will be given.

## 4.1 Observations in lower layers

Before the TLS handshakes are analyzed, this section will show some observations of the lower protocol layers.

### 4.1.1 RADIUS Protocol and eduroam Service Policy violations

During the analysis period several policy and protocol violations have been observed. These observations are just a byproduct of the analysis performed in this thesis, as the RADIUS packets have to be parsed anyway.

For the RADIUS protocol, limitations in the attribute count for the different packet types are defined in Section 5.44 of RFC 2865 [3]. Additional policies for the eduroam environment are defined in the eduroam service policy [28].

The majority of the RADIUS protocol violations will most likely not affect the functionality or security of the eduroam infrastructure, nonetheless this indicates possible configuration errors.

Some policy and protocol violations and their effects are explained below:

**Multiple User-Name attributes in Access-Request** (RADIUS protocol violation):
The User-Name attribute may only occur once in an Access-Request. If it occurs more than once, the RADIUS server cannot determine which attribute value is correct. This may also affect the identification

method of the analysis tool. The tool always uses the first User-Name attribute.

**No Calling-Station-Id attribute present** (eduroam service policy violation):
According to the eduroam service policy, the Calling-Station-Id attribute must contain the MAC address of the connecting client. The service policy does not include reasons for this requirement, but from the operational experience, the MAC address of the client is used in a number of scenarios. This includes accountability and possible legal actions across the federation, configurations based on the device's MAC address or an allow-listing of clients. Since the Calling-Station-Id attribute is required by the service policy, Identity Providers may reject requests without it. For the analysis tool, a missing Calling-Station-Id attribute makes the identification of specific clients impossible.

**Multiple User-Name attributes in Access-Accept** (RADIUS protocol violation):
This indicates a possible configuration error. The User-Name attribute value in an Access-Accept is used for the following accounting requests, if accounting is used. Multiple values may lead to unpredictable behavior of the NAS.

**Multiple State attributes** (RADIUS protocol violation):
The State attribute is used to match an Access-Request packet to a previously sent Access-Challenge packet. If more than one State attribute is present, this might lead to unpredictable behavior on the supplicant's side, thus resulting in sending back the wrong State attribute in the Access-Request.

In addition to the aforementioned violations, the following RADIUS protocol violations are logged:

- User-Name attribute in Access-Reject or Access-Challenge
- State attribute in Access-Rejects
- Multiple Calling-Station-Id attributes in Access-Request
- Calling-Station-Id in other than Access-Request

### 4.1.2 Use of insecure or unfavorable EAP methods

As described in subsection 2.1.2, the EAP method agreement is led by the EAP server. The server sends the first EAP payload for the default EAP method, which is usually configurable. If the client wants to use a different EAP method, it has to send a negative acknowledgment (NAK) EAP message with its requested EAP methods. If the requested method is supported by the server, it then answers with the first payload of this method. This adds a roundtrip to the communication.

The default configuration of FreeRADIUS, a widely used RADIUS server implementation, defines the EAP method 'MD5-Challenge' as default EAP method. Approximately 6.5% of the observed realms offer MD5-Challenge as their initial EAP method. Since FreeRADIUS will not start unless at least a basic configuration for MD5-Challenge is present, this indicates that the configuration has not been disabled. Depending on the further configuration, this could allow a login via MD5-Challenge, which provides no privacy and is vulnerable to attacks against the password. This login method will fail for WPA2-Enterprise logins, as they require the EAP method to provide cryptographic material, but it may still be used for a wired login.

## 4.2 TLS analysis

In this section the captured TLS handshake data will be analyzed.

As the passive analysis only is able to observe what clients are requesting and how the servers answer to that, the primary focus of the analysis will be the clients. Where possible and reasonable, the corresponding server behavior will be analyzed as well.

### 4.2.1 TLS Versions

The use of a current TLS version is essential for the overall security of the following connection. As described in subsection 2.1.4, the TLS protocol has evolved during the years. The differences in the protocol versions are outlined in the Section 1.1 of RFC 4346 [11] (TLSv1.0 to TLSv1.1) and Section 1.2 of RFC 5246 [10] (TLSv1.1 to TLSv1.2). The major change is the use of a different hash algorithm in TLSv1.2. In TLSv1.0 and TLSv1.1 the PRF for generating the Master Secret and signature of elements was a combination of MD5 and SHA-1. In TLSv1.2 this is replaced with a cipher-suite-specified PRF.

The usage of TLS versions 1.0 and 1.1 has been discouraged by RFC 7525 in May 2015. In March 2021 the usage of these TLS versions has been finally deprecated by RFC 8996 [31]. Also, the BSI discourages the usage of TLS versions before 1.2 in their technical requirement document.

Nonetheless, TLSv1.0 and TLSv1.1 are still used. The analysis has shown that still a number of clients request TLS versions 1.0 or 1.1, as shown in table 4.1. According to the known fingerprinting of TLS clients, these devices are mostly old Android or iPhone devices. A detailed comparison is shown in figure 4.1 on the next page.

| | Uni | % | DFN | % |
|---|---|---|---|---|
| TLSv1.3 | 114 | 0.38 | 2,015 | 0.14 |
| TLSv1.2 | 28,939 | 97.53 | 1,379,038 | 97.02 |
| TLSv1.1 | - | - | 253 | 0.02 |
| TLSv1.0 | 620 | 2.09 | 40,040 | 2.82 |
| Total | 29,673 | | 1,421,346 | |

Table 4.1: Distribution of clients TLS versions

For the server side, the analysis has shown that a number of Identity Providers do not support TLSv1.2. Approximately 16.6% (DFN) of all realms do not negotiate TLSv1.2, even if the client indicates support for this version.

Since TLSv1.2 was specified in 2008, this indicates that a number of servers use long outdated cryptographic libraries. The missing support for current protocol versions might also lead to incompatibilities with updated clients, which do not support the old protocol versions. An analysis of possible incompatibilities has not been performed, it could be detected by evaluating TLS alerts.

To improve the overall security of the local network, RADIUS server administrators may choose to disable TLSv1.0 and TLSv1.1 in the near future. However, this step requires a weighing between excluding users, whose clients are not up-to-date and increasing the overall security of the local network by rejecting these clients.

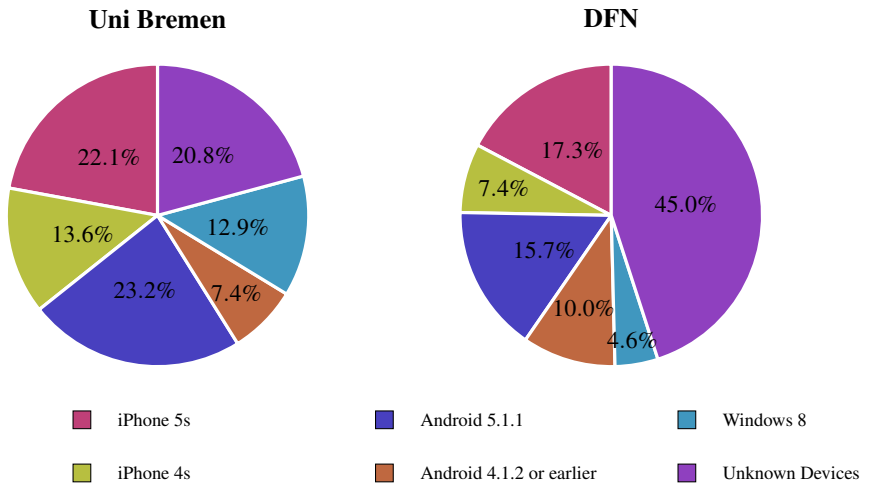


Figure 4.1: Distribution of clients supporting only TLSv1.0 or TLSv1.1

### 4.2.2 Cipher Suites

The essential part for the encryption is the negotiation of the cipher suite. It defines the used encryption and authentication methods.

Each cipher suite is mapped to a two byte cipher suite value, which are registered with the IANA[1]. The client sends a list of cipher suites it is capable of, ordered in its preference. The server can then choose any cipher suite from this list as chosen cipher in their Server Hello.

The cipher suite usually contains four parts:[2]

**Authentication:** This determines which algorithms are used to authenticate the identity of the server. Currently, this is usually done by using the RSA algorithm. Depending on the key exchange method, the server proves the possession of its private key either by decrypting the Pre Master Secret sent by the client or by signing the key exchange parameters sent to the client.

**Key exchange:** This determines which algorithms are used to exchange the Pre Master Secret, used to calculate the symmetric keys used in the following encryption. This can be the same algorithm used in the authentication, in which case the client generates a Pre Master Secret and encrypts it with the server's public key. In this case the authentication and key exchange are done with the same step, as the server proves the possession of the private key by decrypting the Pre Master Secret. Alternatively, this can also be a Diffie-Hellman key exchange, either with natural numbers (DH) or over elliptic curves (ECDH). The DH algorithms are also separated into algorithms with static keys (DH, ECDH) and exchanges with short-lived parameters, also known as ephemeral (DHE, ECDHE). To achieve perfect forward secrecy, only DHE or ECDHE cipher suites should be used, as Diffie-Hellman key exchanges with static keys can also be decrypted afterwards.

**Encryption:** This determines the algorithm used for the symmetric encryption of the payload. It includes the algorithm as well as the used key length, if the algorithm can use different key lengths. Currently, this is usually either AES-128 or AES-256. If the chosen symmetric algorithm is a block cipher, the definition

[1] `https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4`

[2] As this thesis does only analyse TLSv1.0-TLSv1.2, the modifications made for cipher suites in TLSv1.3 are not mentioned here.

also includes an operation mode like cipher block chaining (CBC) or Galois/Counter Mode (GCM). This could also define an AEAD cipher suite, which includes an authentication method of itself, e.g. CCM (Counter mode with CBC-MAC).

**Pseudo Random Function:** This determines the algorithm to be used as pseudo random function. Unless otherwise defined by the cipher suite, TLS uses an HMAC mechanism for integrity checks. If HMAC is used, the function stated here is used for HMAC. In TLSv1.0 and TLSv1.1, this is usually either MD5 or SHA-1. TLSv1.2 defines SHA-256 as default, which can be overwritten by the cipher suite. This is especially useful for achieving an even level of security, e.g. SHA-384 with AES-256. In TLSv1.2, this also defines the PRF used for the generation of key material like the Master Secret, where previous TLS versions used a combination of MD5 and SHA-1.

The goal for the server and the client is to offer a set of cipher suites that matches their needed security level and yet remain compatible with older peers. If the intersection of the cipher sets of client and server is empty, the TLS handshake will fail, preventing the peers from communication.

Since the selection of the server's cipher suite inevitably depends on the client's cipher suites, the following section focuses primarily on a security assessment of the client's offered cipher suites. If the security of the server is to be analyzed, TLS Client Hellos with different cipher sets must be actively sent to the server, because this is the only way to learn about all the cipher suites supported by the server. However, there are a few assessments that can be made about the security of the servers, e.g. based on the choice of a non-PFS cipher suite, when PFS capable cipher suites have been offered. Depending on the variety of clients, it could be tried to find out if the server follows its own preference of cipher suites or that of the clients. Since this assessment would be highly speculative, especially if only a small variety of clients have been observed for this specific realm, this analysis was not performed.

In the majority of cases (67.5% at DFN, 70.3% at Uni), the list of offered cipher suites from the client contains 21 cipher suites, not counting SCSV. The most commonly offered cipher suite is `TLS_RSA_WITH_AES_128_CBC_SHA`, which uses RSA as authentication and key exchange method, AES-128 in CBC mode for encryption and SHA-1 as hash function for the calculation of the HMAC. This is also the mandatory to implement cipher suite specified in the TLSv1.2 specification [10, section 9]. The most commonly chosen cipher suite is `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, which uses RSA for server authentication, ephemeral elliptic curve Diffie-Hellman as key exchange, AES-256 in GCM mode for encryption and SHA-384 for the HMAC.

The list of most commonly offered cipher suites can be found in table 4.2 on the following page. Since an analysis of all different sets of cipher suites can not be performed effectively, the analysis will concentrate on each aspect of the offered encryption algorithms separately and only put them in the context of the other offered cipher suites where practical.

#### Usage of insecure or outdated algorithms

A number of encryption methods that have been used for long years are now considered insecure. Some of them contain design flaws, which reduces their security level, although their key length is sufficient. This is especially true for RC4 [32]. For some other encryption methods the key length is not sufficient, either because it has been intentionally shortened, as in the case of `EXPORT` ciphers, or because the specification does not allow for longer keys. This applies to DES, which uses a 56 bit key.

Here the analysis has shown, that 0.7% (DFN)/0.6% (Uni) of all clients offer cipher suites on `EXPORT` level. According to the TLS fingerprinting, these are Android devices up to Android 5.1.1.

| **Most common cipher suites at Uni** | # of Clients | % |
|---|---|---|
| TLS_RSA_WITH_AES_128_CBC_SHA | 29,672 | >99.99 |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | 29,669 | 99.99 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | 29,669 | 99.99 |
| TLS_RSA_WITH_AES_256_CBC_SHA | 29,669 | 99.99 |
| TLS_RSA_WITH_3DES_EDE_CBC_SHA | 29,271 | 98.65 |
| TLS_RSA_WITH_AES_128_GCM_SHA256 | 29,037 | 97.86 |
| TLS_RSA_WITH_AES_256_GCM_SHA384 | 29,037 | 97.86 |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | 28,868 | 97.29 |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | 28,868 | 97.29 |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA | 28,806 | 97.08 |
| Total nunber of clients: | 29,673 | |

| **Most common cipher suites at DFN** | # of Clients | % |
|---|---|---|
| TLS_RSA_WITH_AES_128_CBC_SHA | 1,421,213 | 99.99 |
| TLS_RSA_WITH_AES_256_CBC_SHA | 1,419,889 | 99.90 |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | 1,419,578 | 99.88 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | 1,419,572 | 99.88 |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA | 1,419,559 | 99.87 |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA | 1,419,553 | 99.87 |
| TLS_RSA_WITH_3DES_EDE_CBC_SHA | 1,403,342 | 98.73 |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | 1,381,077 | 97.17 |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | 1,381,077 | 97.17 |
| TLS_RSA_WITH_AES_128_GCM_SHA256 | 1,381,036 | 97.17 |
| Total number of clients: | 1,421,346 | |

Table 4.2: The 10 most commonly offered cipher suites

A considerable higher percentage of clients (14.0% at DFN/13.5% at Uni) offer cipher suites with outdated algorithms. This includes the algorithms `RC2`, `RC4`, `DES` and `IDEA` for encryption and `MD5` as hash function. With these clients, it is worth noting that some current Android devices with Android 10 are also among these clients, as they offer the cipher suite `TLS_RSA_WITH_RC4_128_SHA`, although this differs in different Android 10 versions.

#### Level of security

The BSI specifies a minimum security level of 100 bits in their technical guidelines. They plan to raise it to 120 bits from 2023 [30, section 3.1.2]. RFC 7525 does not specify any minimum requirement on the security level, although it recommends the usage of cipher suites with either AES-128 or AES-256 as encryption method [29, section 4.2]. The RFC7525bis draft, which is currently discussed in the IETF and will, if ratified, replace RFC 7525, requires a minimum security level of 112 bits, and recommends a minimum security level of 128 bit [33, section 4.1]. However, the content of the RFC7525bis draft may be subject to change.

For the clients, only 0.1% (DFN)/0.01% (Uni) of all clients do not include a cipher suite with a security level of 256 bits. All of those clients use TLSv1.0, and most of them only offer two cipher suites: `TLS_RSA_WITH_3DES_EDE_CBC_SHA` and `TLS_RSA_WITH_AES_128_CBC_SHA` All other clients include cipher suites with AES-256.

On the other side, a large number of clients offer cipher suites with low security levels. 14.0% (DFN) / 13.5% (Uni) of all clients offer a cipher suite with a security level below 112 bit, mostly because RC-4 is included in their cipher suites.

#### Requested and negotiated encryption methods

Almost all clients observed support AES-128 and AES-256. Only a negligible amount of clients does not support it, most of which only offer the cipher suite `TLS_RSA_WITH_3DES_EDE_CBC_SHA`. A detailed listing of requested encryption algorithms can be found in table 4.3 on the next page.

The stream cipher ChaCha20 is requested primarily by Android devices, starting with Android 7. It is designed for efficient calculation on processors without hardware support for AES. Since most embedded processors nowadays have support for AES, the advantage of using ChaCha20 over AES is presumably negligible.

The broken stream cipher RC4 is also mostly requested by Android devices, up to some current Android 10 devices, as previously stated.

For the negotiated cipher suite, almost in all cases an AES variant is chosen. The detailed overview is shown in table 4.4 on the following page. The small fraction of negotiated RC4 cipher suites originate in realms that have several additional findings. Especially the lack of some other security measures, described in the following sections, lead to the conclusion that long outdated software is being used.

#### Requested key exchange methods

Almost all observed clients offer at least one cipher suite with RSA as well as ECDHE as key exchange method. The detailed overview of the client's offered key exchange algorithms can be found in table 4.5 on page 34.

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| AES-128 | 29,672 | > 99.99 | 1,421,227 | 99.99 |
| AES-256 | 29,669 | 99.99 | 1,419,902 | 99.90 |
| 3DES | 29,271 | 98.65 | 1,403,344 | 98.73 |
| ChaCha20 | 9,190 | 30.97 | 482,654 | 33.96 |
| RC4-128 | 3,994 | 13.46 | 199,121 | 14.01 |
| DES | 194 | 0.65 | 10,600 | 0.75 |
| RC4-40 | 190 | 0.64 | 10,546 | 0.74 |
| DES-40 | 190 | 0.64 | 10,540 | 0.74 |
| RC2-40 | 190 | 0.64 | 10,488 | 0.74 |
| Camellia-128 | 46 | 0.16 | 9,959 | 0.70 |
| Camellia-256 | 46 | 0.16 | 9,959 | 0.70 |
| SEED | 44 | 0.15 | 8,836 | 0.62 |
| IDEA | 6 | 0.02 | 358 | 0.02 |
| NULL* | - | - | 63 | <0.01 |
| UNKNOWN** | 1 | < 0.01 | 25 | < 0.01 |

Table 4.3: Requested encryption algorithms

**cipher suites offering no encryption, also known as `eNULL`*
***cipher suites not specified/included in IANA list*

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| AES-256 | 21,856 | 73.66 | 1,072,739 | 75.47 |
| AES-128 | 7,660 | 25.81 | 344,588 | 24.24 |
| ChaCha20 | 154 | 0.52 | 2,517 | 0.18 |
| 3DES | 3 | 0.01 | 1,438 | 0.10 |
| RC4-128 | - | - | 64 | < 0.01 |

Table 4.4: Negotiated encryption algorithms

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| RSA | 29,673 | 100.00 | 1,421,339 | > 99.99 |
| ECDHE | 29,669 | 99.99 | 1,419,588 | 99.88 |
| DHE | 10,414 | 35.10 | 300,097 | 21.11 |
| ECDH | 277 | 0.93 | 22,184 | 1.56 |
| SRP SHA | 44 | 0.15 | 3.958 | 0.28 |
| DH | 20 | 0.07 | 1,700 | 0.12 |
| UNKNOWN | 1 | < 0.01 | 25 | < 0.01 |

Table 4.5: Requested key exchange algorithms

The small number of clients not offering RSA as key exchange (7 clients at DFN) use up-to-date cipher suites and appear to implement all security relevant TLS features discussed in this thesis. This may be a result of a special manual configuration of the supplicant by technically skilled persons.

The other side, clients not offering ECDHE as key exchange method, indicate outdated clients. Almost all these clients support only TLSv1.0 and do not have support for secure renegotiation (see section 4.2.4 on page 37) or Extended Master Secret (see section 4.2.5 on page 39). Most of these clients can not be identified by the TLS fingerprint, only four out of 1,762 (DFN and Uni combined) devices were identified as Android 2.2. Some clients (17.2% at DFN) did offer a DHE cipher suite. If the DHE cipher suites are not offered, the clients are not capable of negotiating a TLS connection with perfect forward secrecy, which renders them incompatible with servers with a PFS-only cipher set.

Static ECDH cipher suites are offered by 1.6% (DFN) / 0.9% (Uni) of all devices, which are identified as Android 4 and Android 5, if the fingerprint was found in the database. All these clients offered an ECDHE cipher suite, which ensures the capability for PFS-only servers.

A small percentage of clients (0.28% at DFN, 0.15% at Uni) offer a cipher suite with key exchange over the Secure Random Password protocol, which was specified in RFC 5054 [34]. This method allows for the client and server to perform a key exchange based on a calculation like Diffie-Hellman, but using a password in addition to the exchanged parameters to calculate the Pre Master Secret. Cipher suites containing SRP are offered by almost exclusively Android 4.1.2 devices. Since the Client Hello of these clients does not contain the SRP extension needed to transmit their identity, a negotiation of this key exchange method would require a reconnect after the TLS server indicated support for SRP [34, section 2.5.1.2].

On the server side, the majority of connections use ECDHE as key exchange method. Only 3.2% (DFN) / 0.3% (Uni) of all connections are negotiated using RSA for key exchange, thus not having perfect forward secrecy. An overview of all negotiated key exchange methods can be found in table 4.6 on the following page. The details of the ECDHE key exchange and the chosen elliptic curves are discussed in section 4.2.6 on page 41.

Due to the limited number of DHE key exchanges, this key exchange is not analyzed in detail in the analysis tool. Future versions may collect additional data for these handshakes.

#### Requested authentication methods

All observed clients support the RSA authentication method. As the authentication method needs to match the certificate parameters, a client not supporting RSA for authentication will not be able to connect to a server with only an RSA certificate. An overview of supported authentication algorithms can be found in table 4.7 on the following page.

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| ECDHE | 29,570 | 99.65 | 1,373,727 | 96.65 |
| RSA | 100 | 0.34 | 45,149 | 3.18 |
| DHE | 3 | 0.01 | 2,469 | 0.17 |
| None† | - | - | 1 | $< 0.01$ |

Table 4.6: Negotiated key exchange algorithms

† *This is a TLSv1.3 handshake. Since the key exchange in TLSv1.3 is negotiated using extensions, the cipher suite does not contain information about it.*

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| RSA | 29,673 | 100.00 | 1,421,346 | 100.00 |
| ECDSA | 28,806 | 97.08 | 1,419,570 | 99.86 |
| DSS | 551 | 1.86 | 26,370 | 1.86 |
| NULL‡ | - | - | 36 | $< 0.01$ |
| UNKNOWN | 1 | $< 0.01$ | 25 | $< 0.01$ |

Table 4.7: Requested authentication algorithms

‡*cipher suites offering no authentication, also known as* `aNULL`

Only a small number of clients do not support ECDSA authentication (0.14% at DFN, <0.01% at Uni[3]). The vast majority of devices not capable of ECDSA authentication also do not offer any cipher suites with PFS, which indicates outdated operating systems.

Cipher suites with DSS as authentication are requested by Windows 7 and 8 and Android up to 5.1.1 according to the TLS fingerprinting.

On the server side, RSA is chosen in almost all cases. Only a negligible amount of handshakes was done using the ECDSA algorithm. A detailed overview can be found in table 4.8 on the next page.

Using ECDSA as authentication algorithm has a number of advantages over RSA. First, ECDSA requires a much smaller key length than RSA to achieve the same level of security, which makes computation more efficient. The smaller key length also makes the certificate smaller. An example ECDSA certificate issued by Let'sEncrypt using the secp384r1 curve was 1181 bytes long, where a similar certificate with a 4096 bit RSA key was 1611 bytes long. Considering the usual maximum EAP-TLS fragment size of approximately 1000 bytes, this could save a roundtrip if ECDSA certificates are used.

The certificate details are not captured in the data, but the analysis of the saved certificates has shown that a small number of certificates use 1024 bit RSA keys, which are considered insecure. This will be a topic for further research.

### `aNULL`, `eNULL` and unknown ciphers

Over the period of the analysis, a small number of clients offered cipher suites not included in the list of specified cipher suites or use the range reserved for private use. Since the details of the used algorithms for these cipher suites are unknown, their security level and implications of usage cannot be assessed.

[3]The data of the University of Bremen appears to be poisoned by a client with a randomized MAC address, as described in section 3.4. For this specific analysis, these clients are considered as one device. If the clients are considered different, the percentage for the university data rises to 2.92% as listed in table 4.7.

| **Algorithm** | Uni | % | DFN | % |
|---|---|---|---|---|
| RSA | 29,669 | 99.99 | 1,420,491 | 99.94 |
| ECDSA | 4 | 0.01 | 854 | 0.06 |

Table 4.8: Negotiated authentication algorithms

Especially the usage of cipher suites in the range for private usage may lead to incompatibilities if the same cipher suite value is used by different vendors for different algorithms.

Additionally to the unknown cipher suites, a small number of clients have offered `aNULL` or `eNULL` cipher suites.

`eNULL` cipher suites do not offer any encryption. They only ensure the authenticity and integrity of the data, but the payload is sent in plaintext. Using this cipher suite in conjunction with an inner authentication method that sends the password unencrypted makes the password visible to an attacker eavesdropping on the connection. As shown in table 4.3 on page 33, 63 clients in the DFN data offered an `eNULL` cipher suite. The cipher suite offered by all these clients is `TLS_RSA_WITH_NULL_MD5`, which uses RSA for authentication and key exchange and an HMAC based on MD5 for ensuring integrity. Additionally, some clients offered different other `eNULL` cipher suites, which use ECDSA for authentication, DHE or ECDHE for key exchange and an HMAC based on SHA-1 or SHA-2. All clients offering `eNULL` cipher suites also support stronger cipher suites with DHE, RSA and AES-256. The captured MAC addresses for all these clients are registered to Apple, which leads to the conclusion that these are outdated Apple devices. Since the usage of `eNULL` cipher suites does not provide confidentiality, they must not be used in EAP-TLS.

`aNULL` cipher suites do not offer authentication. These cipher suites can be used for protection against simple eavesdropping attacks. Usually the cryptographic keys are exchanged using anonymous Diffie-Hellman key exchanges, where both partners send their parameters without any signature. Since an attacker wiretapping the connection cannot calculate the cryptographic keys from the public Diffie-Hellman parameters sent by the peers, they cannot encrypt the content. However, `aNULL` cipher suites cannot protect against active attackers. Given the fact that any attacker could broadcast an eduroam network and trick clients to connect to the rogue network with a rogue server, clients offering this cipher suites may connect to the malicious server without enforcing it to authenticate itself using a certificate. As shown in table 4.7 on the previous page, 36 clients in the DFN data offered an `aNULL` cipher suite. Again, the majority of devices (29) use MAC addresses registered to Apple, leading to the conclusion that these too are outdated Apple devices. All Apple devices offering `aNULL` cipher suites, did also offer `eNULL` cipher suites. Since the authenticity of the communication partner can not be assured using an `aNULL` cipher suite, it must not be used in EAP-TLS.

### 4.2.3 Compression

The TLS standard includes support for data compression. Within the TLS handshake, the client and server can agree to compress the payload data to save bandwidth. Along with different compression methods, the client can send the NULL compression in its supported compression methods. If the NULL compression is chosen, the payload data will not be compressed.

If an attacker can control some input data, they may afterwards be able to infer other input data, based on the length of the compressed data. This attack is known as CRIME (Compression Ratio Info-leak Made Easy)[35].

RFC 7525 discourages the usage of compression, “unless the application protocol in question has been

shown not to be open to such attacks."[29, section 3.3] For EAP-TLS a known plaintext insertion will be very difficult, since the supplicant only transmits its client certificate or user name and password inside, which an attacker can most likely not modify. Therefore, the usage of compression will most likely not pose a significant security issue.

Nonetheless, the use of compression might reveal some properties of the inner authentication, if plaintext authentication methods (e.g. PAP or GTC) are used inside of the TLS. Here, an attacker might be able to guess a certain similarity between user name and password or repeated characters in the password based on the length of the encrypted data. If the outer user name is not anonymized an attacker has a further knowledge of some plaintext, as it is retransmitted inside the TLS tunnel. This is only a theoretical consideration, an actual evaluation of the possibility of this attack has not been performed.

Since most TLS implementations have removed the compression possibility from their implementation either completely or disabled it by default after the CRIME attack had been disclosed, a compression offer in the Client Hello may indicate that the supplicant is using an outdated TLS library.

Within the captured data, only two clients offered a compression method (DEFLATE), along with the NULL compression. According to the TLS fingerprint, one device was an Android 2.2 device. The fingerprint of the other device was not included in the fingerprint database, but showed only slightly different TLS parameters compared to the aforementioned device.

All other clients, including at least one other device identified as Android 2.2, did not offer any other compression methods than the NULL compression, meaning they will not compress the data sent before encryption.

### 4.2.4 Secure Renegotiation

The TLS protocol allows for a renegotiation of security parameters. This can be initiated by any party during the connection. The TLS handshake is then performed again exactly as the initial handshake, except that the handshake messages are encrypted according to the currently negotiated encryption.

The original specification of the renegotiation contains a design flaw, since it has no cryptographic binding to the previous connection. This enables an attacker to splice in a connection from a client after connecting to a server and sending its own data. The attacker simply forwards the client's handshake messages for its initial connection to the server, after encrypting them with the negotiated keys between current between attacker and server. The server assumes the client wants to renegotiate and sends its own encrypted Server Hello back, which is then decrypted by the attacker and forwarded to the client. The client is also unable to detect the attack, as the Server Hello for renegotiation does not differ from an initial Server Hello. Assuming the TLS library does not notify the application layer about the renegotiation, the data will be treated as coming from the same entity. A detailed description of the attack can be found in RFC 5746 [21].

To circumvent this attack, RFC 5746 specifies the TLS extension `RENEGOTIATION_INFO` along with a Signaling Cipher Suite Value (SCSV). When sending either the extension or the SCSV in the initial TLS Client Hello, the clients indicate support for secure renegotiation. The SCSV value was specified in addition to the TLS extension to provide backwards compatibility with TLS implementations which do not support extensions at all. The original specification of TLSv1.0 did not contain extensions, but required implementations to ignore additional data after the specified protocol fields. Unfortunately, not all implementations followed this requirement and rejected all TLS Client Hellos containing extensions. If the server supports secure renegotiation, it answers with an empty `RENEGOTIATION_INFO` extension. This is the only extension which the server is allowed to send, even if the client did not send it, but only if the client indicated its support for secure renegotiation via the SCSV.

When either side wants to renegotiate, the Client Hello must contain the renegotiation extension with the content of the finished message of the previous handshake as extension data, thus binding the previous and current handshake cryptographically, since the content of the finished message can only be known to the original communication partners. The server answers with the concatenated previous finished messages of the client and the server in its renegotiation extension. Since the finished messages of the renegotiation are built based on the previous handshake message, an attacker is not able to manipulate the client's and server's handshake messages without client and server noticing, thus preventing the previous described attack.

RFC 5746 recommends indicating support for secure renegotiation even in the case renegotiation itself is not supported, which prevents the attack. For these purposes, the SCSV can be included in the Client Hello, as the use of the SCSV is prohibited for renegotiations. In this case the server would recognize the attack and terminate the connection.

For EAP-TLS, TLS renegotiation will not be used in most communications, since the TLS connection is short lived and there is no need for a renegotiation of cryptographic key material. In the EAP-TLS authentication method using client certificates, it might be desirable for privacy reasons not to send the client certificate unencryptedly. Here, renegotiation is used to first establish an encrypted tunnel to the server without the need for a client certificate which is sent afterwards. In the initial handshake the client sends an empty certificate record as response to the server's certificate request. The server sends a hello request to the client immediately after completion of the first handshake, starting a renegotiation over the then already encrypted channel. In this handshake, the client sends its client certificate and the handshake is then completed with mutual authentication.

In the observation, approx. 99.9% (DFN)/99.97% (Uni)[4] of all observed clients did indicate support for secure renegotiation. The operating system of clients without support for secure renegotiation could not be identified by their TLS fingerprint. Based on the distribution of offered cipher suites of the clients, these clients are a mixture of clients with different security settings. Only a small percentage of the clients support TLSv1.2 (1.5%, DFN). For the University this analysis cannot be performed, since only nine devices[5] have been captured, which is not sufficient data to make a statistical statement.

The SCSV is used by Apple devices and Android up to Android 6. Later Android versions use the `RENEGOTIATION_INFO` extension to signal their support for secure renegotiation. The method used has no implication on the security, since the specification explicitly requires the implementations to recognize either. The use of the SCSV requires three bytes less than the use of the extension, which will most likely not affect the underlying protocols at all, since the TLS Client Hello usually can be transmitted in a single RADIUS packet.

On the server side, only a small amount of realms do not support secure renegotiation. The lack of indication may have its origin in the fact that the corresponding servers do not support renegotiation at all, thus not needing to secure the renegotiation. This fact can only be checked with an active probing of these servers, which is out of scope for this thesis. Based on additional data about the affected realms, this fact can be assumed for some realms. Most realms lacking support for secure renegotiation also lack support for Extended Master Secret, indicating deprecated or incomplete TLS implementations.

[4] This percentage is also calculated counting the poisoned clients mentioned in section 4.2.2 as one. If the clients are considered different, the percentage for the university data drops to 97.0%.

[5] Counting the aforementioned device(s) with randomized MAC address as one

### 4.2.5 Extended Master Secret

The Extended Master Secret mechanism circumvents a design flaw in the TLS specification.

In the original specification, the Master Secret is calculated using the client and server random and the Pre Master Secret. If an attacker is able to intercept the connection, they can open a second connection to the original server with the same Master Secret. The attack principle is shown in figure 4.2 on the following page. A detailed description of this attack can be found in the paper 'Triple Handshakes and Cookie Cutters' [36], where it is referenced as Man-in-the-middle TLS proxy server.

The attack relies on the client's acceptance of the attacker's certificate. If the client accepts it, the attacker can then initiate a second connection to the original server, which will subsequentially have the same Master Secret as the connection from the client to the server.

To achieve this, the attacker copies the client random from the client's TLS Client Hello to its own Client Hello. After reception of the Server Hello from the original server, the attacker then copies the server random to its own Server Hello. The server's certificate is replaced with the attacker's certificate for the client. The client encrypts its Pre Master Secret with the public key of the attacker. The attacker then decrypts it with its private key and reencrypts it with the public key of the server. Since the Master Secret is then calculated using the client and server random and the Pre Master Secret, and this information is identical for both connections, the Master Secret is identical.

In EAP-TLS based EAP methods, the Master Secret is then used to calculate the Master Session Key and Extended Master Session Key, which are subsequentially used for generating the encryption keys for the following wireless connection or as initialization vectors for inner authentication methods.

In the MS-CHAP-V2 authentication method, the server challenge is computed of the Master Secret. The attacker can then simply relay the messages from the client to the server without knowledge of its content. Since both the connections share the same Master Secret, the server cannot recognize the middle person attack. When the client authenticates successfully, the attacker can then possibly decrypt the wireless traffic.

The TLS session resumption also relies on the session identifier and Master Secret. Some EAP servers authenticate a session resumption without a following repeated inner authentication. In this case, the attacker can impersonate the client for as long as the session can be resumed.

Since the attacker has full control over its own Client Hello, they can direct the choice of cipher suites and extensions, which complicates countermeasures.

The immediate countermeasure is the usage of the Extended Master Secret mechanism, specified in RFC 7627 [23]. The client signals support for this by sending the `EXTENDED_MASTER_SECRET` extension. If the server signals its support by sending the extension back, the Master Secret is then calculated differently. Instead of using the client and server random, this is replaced by a hash of all previous communication. This ensures, that the previous messages have not been altered, since the session hash would be different for the client and the server otherwise, which would result in different Master Secrets.

This countermeasure can be bypassed by the attacker easily, since they have full control over their own Client Hello and can suppress the client's `EXTENDED_MASTER_SECRET` extension. This can only be circumvented, if the server requires all clients to use the Extended Master Secret mechanism.

Currently, only a small percentage (1.9% at DFN/4.2% at Uni) of clients do not signal support for Extended Master Secret. Unfortunately, the TLS fingerprint database does not contain an identification for a large number of the captured clients. Since the fingerprint database primarily contains mobile phones, it can be assumed that some unknown devices are laptops. According to the TLS fingerprinting, the majority of the known devices are old Android versions (up to 5.1.1, which was initially published in 2015)

```
Client              Attacker             Server
  |                    |                    |
  | Client Hello       |                    |
  | Random: cr         | Client Hello       |
  |===================>| Random: cr         |
  |                    |===================>|
  |                    |                    |
  |                    |       Server Hello |
  |       Server Hello |         Random: sr |
  |         Random: sr |      Certificate S |
  |      Certificate A |        Pubkey: K_s |
  |        Pubkey: K_a |<===================|
  |<===================|                    |
  |                    |                    |
  | PreMasterSecret    |                    |
  | Encrypted with K_a | PreMasterSecret    |
  |===================>| Encrypted with K_s |
  |                    |===================>|
  |                    |                    |
  |   Master Secret:   |   Master Secret:   |
  |  PRF(cr, sr, PMS)  |  PRF(cr, sr, PMS)  |
  |<==================>|<==================>|
  |                    |                    |
Client              Attacker             Server
```

Figure 4.2: Simplified overview of the TLS Proxy attack

as well as old iPhones (identified as iPhone 4s).

The majority of clients without support for Extended Master Secret also request TLSv1.0.

Unfortunately, the support for Extended Master Secret is not as widespread on the server side as it is on the clients side. In the DFN data, only 43.6% of all captured handshakes were negotiated using the Extended Master Secret mechanism. Calculated based on the realm of the username, 52.7% (DFN) of all realms did not negotiate Extended Master Secret, even if the client offered it in its extension.

This may be based on the usage of old software. The support for Extended Master Secret was introduced into OpenSSL in version 1.1.0, released in Aug 2016 [37]. In table 4.9 on the next page the standard OpenSSL version on different Debian-based Linux versions is shown.

Since only outdated Debian and Ubuntu versions come with an OpenSSL version without support for Extended Master Secret, the missing support for Extended Master Secrets on the server side may be an indication of outdated RADIUS server software.

The most immediate recommendation is to update RADIUS servers to a version with support for Extended Master Secret. Since a possible attacker has full control over their Client Hello, this is not sufficient to prevent TLS proxy attacks. To minimize the risk of this attack, servers should require the Extended Master Secret, as suggested in Section 5.2 of RFC 7627. However, this would prevent the small number of clients without support for Extended Master Secret from accessing eduroam. Since the analysis has shown that most of these devices are outdated, this might be an acceptable side effect, since the outdated devices might pose a security risk for the network.

The enforcement of the Extended Master Secret mechanism might not be possible everywhere. This might be caused by the limitations in the used software, as e.g. FreeRADIUS currently does not provide

| OS Version | Status | OpenSSL Version | Support for EMS |
|---|---|---|---|
| **Debian** | | | |
| Debian 10 (buster) | supported (current) | 1.1.1d | ✓ |
| Debian 9 (stretch) | supported (LTS until 06/2022) | 1.1.0l | ✓ |
| Debian 8 (jessie) | not supported (EOL 06/2020) | 1.0.1t | × |
| Debian 7 (wheezy) | not supported (EOL 05/2018) | 1.0.1e | × |
| **Ubuntu** | | | |
| Ubuntu 21.04 (hirsute) | supported (EOL 01/2022) | 1.1.1j | ✓ |
| Ubuntu 20.10 (groovy) | not supported (EOL 06/2021) | 1.1.1f | ✓ |
| Ubuntu 20.04 LTS (focal) | LTS release (EOL 04/2025) | 1.1.1f | ✓ |
| Ubuntu 18.04 LTS (bionic) | LTS release (EOL 04/2023) | 1.1.1 | ✓ |
| Ubuntu 16.04 LTS (xenial) | not supported (EOL 04/2021) | 1.0.2g | × |
| Ubuntu 14.04 LTS (trusty) | not supported (EOL 04/2019) | 1.0.1f | × |

Table 4.9: Overview of OpenSSL versions in the package repository of different Linux distributions
LTS: Long Term Support
Extended Security Maintenance (payed support) not included.
Sources: `https://wiki.ubuntu.com/Releases` and `https://wiki.debian.org/LTS`

a configuration option to enforce Extended Master Secret. Additionally, there may exist devices without support for Extended Master Secret, which still have to be able to access eduroam. In these cases, the next best circumvention of this attack is the use of ECDHE cipher suites, as they are less susceptible to the TLS proxy attack. DHE cipher suites can also be used, if the implementation effectively prevents attacks with specially chosen DHE-Parameters, which would allow a synchronization of the DHE key agreement for both connections. [36, p. 105f, V-B]

### 4.2.6 Supported Groups

When client and server agree on a cipher suite with ECDHE key agreement protocol, client and server also have to agree on a specific elliptic curve to use.

The support for different curves is transmitted in the `SUPPORTED_GROUPS` extension. This extension was renamed from `ELLIPTIC_CURVES` by RFC 7919 [25], which introduced support for finite field Diffie-Hellman parameters in this extension as well.

A number of standardized curves has been summarized and specified by the Standards for Efficient Cryptography Group (SECG) [38]. These curves have either been defined by different entities before or were defined by the SECG. Some additional elliptic curves have been defined in RFC 7027 [39] (brainpool curves) and by Daniel J. Bernstein (x25519) [40] and Mike Hamburg (x448) [41]. The finite fields for Diffie-Hellman have been specified in RFC 7919.

All captured clients which support ECDHE cipher suites support the curves `secp256r1` (also known as `NIST P-256` or `prime256v1`) and `secp384r1`. The curve `x25519` is only supported by 39.5% (DFN) / 47.7% (Uni) of the clients, which are mostly newer Android versions for the known TLS fingerprints. A detailed breakdown of supported groups can be found in appendix D.

The support for the curve `secp521r1` appears to be dropped in higher Android versions (beginning from Android 8). Earlier Android versions beginning with Android 4 supported all curves specified by SECG in their "Recommended Elliptic Curve Domain Parameters" document, including curves with 163 bit keys, which corresponds to a security level of ≈80 bits.

As described in section 4.2.2, over 95% of all observed handshakes used a cipher suite with ECDHE key exchange. The server's selection of curves appears to be very limited. Only handshakes with the curves `secp256r1`, `secp384r1`, `secp521r1` and `x25519` have been observed. Here the DFN data shows over 90% usage of the curve `secp256r1`, which is most likely the result of the default setting of the widely deployed software FreeRADIUS.

The configuration in Bremen allows for different curves with the server preferring `x25519`, 40.0% of all observed handshakes there used the `x25519` curve.

RFC 7525 recommends support for at least the curve `secp256r1` for interoperability reasons, which all observed clients capable of ECDHE support. The technical requirements from the BSI recommend the usage of `secp256r1`, `secp384r1` or the brainpool curves, the `x25519` or `x448` curves are not mentioned in either document.

### 4.2.7 Signature Algorithms

The signature scheme determines the algorithms used to ensure the authenticity of the server's Diffie-Hellman parameters. This mechanism was introduced in TLSv1.2. In previous TLS versions the Diffie-Hellman parameters were signed using an SHA-1 hash together with an MD5 hash when using RSA and an SHA-1 hash when using DSS [12, section 7.4.3].

The original specification of TLSv1.2 contained a strict separation between hash and signature algorithm [10, section 7.4.1.4.1]. All TLS versions from 1.0 to 1.2 used the PKCS #1 standard to create the signature, if RSA was used.

In TLSv1.3 the separation between hash and signature algorithm for the signature scheme in the registry has been lifted. This results in different name schemes in the analysis tool output for the different signature schemes. Signature schemes included in the TLSv1.3 standard are named as specified in the TLS SignatureScheme registry of IANA[6]. Signature schemes not included in TLSv1.3 are named according to their respective names in the TLS SignatureAlgorithm and TLS HashAlgorithm registry, indicated by upper case letters.

All observed clients capable of TLSv1.2 or TLSv1.3 offered the signature algorithms `rsa_pkcs1_sha256` and `rsa_pkcs1_sha384`. Almost all clients also offered `rsa_pkcs1_sha1`, which uses the deprecated SHA-1 algorithm to hash the Diffie-Hellman parameter.

On the server side, the majority of observed handshakes used the signature scheme `rsa_pkcs1_sha256`. The outdated SHA-1 algorithm is used in less than 0.2% (DFN) / 0.1% (Uni) of all handshakes, if TLSv1.2 is used.

Especially interesting is the choice of signature schemes for one specific realm, where the connections made using TLSv1.2 used the `rsa_pkcs1_sha1` signature scheme, although the clients included better signature schemes (e.g. using SHA-256) and preferring those methods. The negotiated cipher suite in this case used SHA-384 as hash function for the MAC, which indicates that the RADIUS server supports this hash function. The usage of SHA-1 for signing the Diffie-Hellman parameters may be a misconfiguration or indicate a flaw in the TLS stack.

[6] `https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-16`

### 4.2.8 Status Request (OCSP)

The Online Certificate Status Protocol (OCSP), specified in RFC 6960, provides a mechanism for a live checking of the revocation status of certificates. The client connects to an OCSP server, usually operated by the corresponding certificate authority, and requests a proof of validity for the certificate sent by the TLS server. The proof is signed by the OCSP server.

This method requires the client to open a second connection while connecting to the first TLS server. In the eduroam environment, the client usually does not have an active Internet connection yet, thus the certificate status cannot be checked directly. For these purposes, the TLS extension `STATUS_REQUEST` can be used, which allows the TLS server to send a proof of validity along with the certificate. This mechanism, also known as OCSP Stapling, is specified in RFC 6066 [22].

The client signals its support for OCSP Stapling by including the extension `STATUS_REQUEST` in its Client Hello. The server then answers with this extension included in its Server Hello. After sending the certificate, the server adds a CertificateStatus record, which includes the OCSP response from the OCSP server. Since the servers certificate does not often change, the server can reuse the staple issued by the OCSP server within its validity period for different clients. This method enables the client to check the revocation status of the servers certificate without the need of a second connection.

Since the usage of OCSP Stapling is usually not mandatory, a client requesting OCSP Stapling will connect to a server without support for it. An attacker, who gained knowledge of the private key, may use the corresponding certificate and simply disable support for OCSP stapling, since the staple would indicate that the certificate is revoked. This can only be circumvented by either using additional information in the certificate, which signals the client to reject the certificate, if the server does not support OCSP stapling (X.509-Extension TLS Feature, also known as OCSP MustStaple) or configuring the client for this behavior. Since this thesis does not focus on the usage of certificates in EAP-TLS, the implications of either behavior will not be discussed here.

Approximately 46.3% (Uni) / 60.6% (DFN) of the clients indicate support for OCSP stapling in their Client Hellos, which are almost exclusively Apple devices. On the server side, the support for OCSP stapling is rare, only ≈7.3% (DFN) of all observed realms support this extension, which is the result of missing support in the server implementations [18, slide 11].

### 4.2.9 Heartbeat

The TLS Heartbeat extension was defined in RFC 6520 [42]. It intended to provide a mechanism to check the liveness of the communication partner and to refresh state in middleboxes like firewalls or NATs, without need for actual payload data to be sent.

The client and server can negotiate usage of the heartbeat mechanism by sending the TLS extension `HEARTBEAT`. If the client included the extension and the server answered with it, the client and server may send Heartbeat records at every stage of the TLS connection. This record can include a payload with variable length. The payload content is then mirrored back to the sender.

In EAP-TLS the connection between client and server is very short-lived, so the support for heartbeats serves no practical purpose.

The heartbeat implementation of the cryptographic library OpenSSL contained an overflow bug. If an attacker sends a heartbeat message with a manipulated length field, OpenSSL returned contents of its memory, due to a missing length check. This bug became known as Heartbleed. [43]

Following the detection of this bug, the usage of the heartbeat extension was discouraged and the support

for the heartbeat extension was dropped in OpenSSL version 1.1.0, released in August 2016 [37].

Since the extension does not serve a purpose in the EAP-TLS environment, it should be disabled. Nonetheless, the observations have shown that a small number of clients (0.6% at DFN / 0.2% at Uni) request the heartbeat extension in their Client Hello. Unfortunately, the devices cannot be identified by their TLS fingerprint. Out of all clients requesting heartbeat, 62.71% (DFN) of the Server Hellos included the heartbeat extension, signaling the support for heartbeat on the server side.

Further research might investigate if the EAP-TLS server indicating heartbeat support are susceptible to the heartbleed bug. Since this would be an active probing of the servers, it is considered out of scope for this thesis.

### 4.2.10 Additional observations

The previous sections analyzed the most important TLS parameters. There are a number of additional observations, which will be summarized shortly here.

#### TLS Extension `ENCRYPT_THEN_MAC`

In RFC 7366 [44], the `ENCRYPT_THEN_MAC` extension was defined. If negotiated, the peers will encrypt their data first and calculate the message authentication code over the encrypted data. This prevents numerous attacks, which rely on different behavior of the implementation if padding errors occur. The BSI recommends the usage of this mechanism, as soon as implementations are available.

The analysis has shown, that only 1.3% (Uni) / 0.7% (DFN) of all clients include the `ENCRYPT_THEN_MAC` in their Client Hello. Most of these clients are identified as Debian 10 or greater. On the server side, only one handshake has been observed where the server included this extension in their Server Hello. Surprisingly, this server negotiated TLSv1.1 with a client capable of TLSv1.2 and supported other security methods like Extended Master Secret or secure renegotiation.

#### Certificates

The analysis tool saves details about the server certificate. Here, a number of realms were observed where the number of sent certificates did not match the expected length. For a successful validation the server needs to include its own certificate as well as all intermediate certificates, which provide a verification path to the trust anchor. The trust anchor is either configured explicitly or the system certificates are used. In both cases, the client has these certificates already stored, therefore the server does not need to send this certificate. Since certificates usually have a size around 1 kB, this adds at least one roundtrip to the EAP communication.

In contrast, realms that only send the server certificate have also been observed. As a result, clients that have not previously saved the intermediate certificate cannot establish a trusted path to the trust anchor.

Since certificates are not the primary focus of the thesis, these observations have not been further investigated.

#### Downgrade detection in the Server Random

The TLS version negotiated should always be the highest possible. Unfortunately, the version negotiation was not handled correctly by all TLS implementations. This led to a possibly dangerous behavior of some

clients: If a server did not answer to a handshake with a higher version or answered with a TLS alert, it tried to negotiate a smaller TLS version. This behavior could be used by an attacker to downgrade the TLS connection by intercepting the first handshake. The client would then negotiate a TLS version the attacker could attack more easily.

To prevent this attack, two countermeasures have been introduced.

First, the client can insert a Signaling Cipher Suite Value (SCSV) into its cipher suite list, indicating that the requested TLS version is not the highest the client is capable of. This mechanism was specified in RFC 7507 [45]. It has not been observed with any client.

The second prevention mechanism was introduced with TLSv1.3 [13, section 4.1.3]. Using a specific value as last bytes of the server random, the server can now signal the client that it is capable of a higher TLS version then the one negotiated. If negotiated TLSv1.0 or TLSv1.1 the server will set its last eight bytes of the server random to the ASCII value of `DOWNGRD0`. If TLSv1.2 is negotiated, TLSv1.3 servers will set this to `DOWNGRD1`.

Since the second downgrade prevention mechanism was introduced in TLSv1.3, which is not yet specified for usage in EAP-TLS, the small percentage of servers using this mechanism is not surprising. All the more surprising is the fact that `DOWNGRD1` was set in 1.8% (DFN) of all observed handshakes. This might lead to the conclusion, that the TLS library used is willing to negotiate TLSv1.3. Tests with the infrastructure at the University of Bremen and the laptops available to us have shown incompatibilities if TLSv1.3 is actually negotiated, due to the unclear behavior which is still being specified.

## 4.3 Summary and utilization of results

Summarizing the previous detailed analysis of the different parts of the TLS handshake, it can be said that the overall effective security level of the eduroam federation appears to be good. Apart from very few exceptions, there are no serious security issues that would open a highly probable attack vector and should be addressed immediately.

On the server side, the overall security level observed is acceptable, although the different observed behavior, especially regarding TLS versions and Extended Master Secret, lead to the conclusion that a considerable number of servers run outdated software. Despite this, the servers are mostly capable of negotiating up-to-date cipher suites with strong cryptographic algorithms.

On the clients side, a large number of outdated algorithms is still offered, which could benefit attacks if an attacker is able to actively interfere with the handshakes. In addition to missing or faulty certificate checking, this could lead to a number of attacks, including password tapping or impersonation. Since eduroam is designed to be a network every user can log in with their private devices, issues found here cannot be as easily addressed as they can on the server side. Although a number of clients is offering outdated algorithms, new security measures are more widely deployed then they are on the server side.

Within the analysis of the data, a number of realms with some security issues showed up. Eight German institutions were contacted and made aware of some of their security issues. The mail contained details about the found security issues as well as some questions regarding the used software and configuration. The content of the mail sent can be found in appendix B. Out of the eight contacted institutions only two replied to the inquiry.

The RADIUS server of the first institution did not support Extended Master Secret or secure renegotiation. They use a Cisco Identity Services Engine[7] with a software version from 2018 as RADIUS server.

[7] `https://www.cisco.com/c/en/us/products/security/identity-services-engine/index.html`

According to the reply, an upgrade is planned in the near future.

The RADIUS server of the second institution did not support Extended Master Secret. Additionally, no PFS cipher suites were negotiated. They use Radiator[8] in the current version. The operating system used `openssl-1.0.2`, which does not support Extended Master Secret. An update of the server operating system was planned for a few weeks later already. The missing support for PFS cipher suite originated in missing configuration statements. After two configuration statements were added, the server negotiated PFS cipher suites.

I would like to take this opportunity to thank the administrators who responded to my request, provided me with details about their local eduroam setup, and with whom I had a very constructive exchange.

During the analysis five realms showed up, that negotiated exclusively RC-4 and 3DES as encryption algorithms, which leads to the conclusion that these RADIUS servers run on heavily outdated software.

[8] `https://www.open.com.au/radiator/`

# Chapter 5

# Conclusions

At the conclusion of this thesis, a final assessment of the results of the analysis will be made and possible further fields of research will be discussed.

## 5.1 General assessment of the security analysis

This thesis wants to answer the question if security problems in the eduroam environment exist. The analysis performed in this thesis has shown that a number of issues exist on both the clients and the server side, but only a few could be considered critical.

On the client side, the analysis has shown a wide support for up-to-date algorithms and a wide implementation of protocol improvements to the TLS protocol. The most important issue found is the client's wide support for outdated or even broken algorithms. Operating system manufacturers may choose to offer these algorithms to ensure backward compatibility with older servers. However, the use of broken algorithms like RC-4 should be removed from the supported algorithms. Especially because outdated algorithms are offered, most clients do not meet the recommendations given in RFC 7525 and the BSI technical requirements in TR-02102-2.

On the server side, the analysis has shown a number of issues, which presumably have their origin in the use of outdated software. The problem existent at most realms is the missing support for Extended Master Secret. As described in section 4.2.5 on page 39, the most probable explanation is the use of an outdated operating system, resulting in outdated libraries. Some additional problems like the missing support for TLSv1.1 or TLSv1.2 may also indicate the use of outdated software. Apart from the aforementioned security problems, the overall security level observed meets the basic requirements for an EAP-TLS environment. The recommendation from RFC7525 and TR-02102-2 are mostly met. Nonetheless, a number of issues indicate that RADIUS servers do not receive as much as attention as web servers.

A number of security measures could be enforced by the server operators to improve the security and minimizing the risk of attacks, which will be discussed in the following section.

## 5.2 Recommendations for RADIUS server administrators

From the analysis performed, a number of recommendations for RADIUS server administrators can be deduced.

First, since the analysis has shown several issues related to outdated software, RADIUS administrators should make sure that their RADIUS servers use up-to-date software. This applies especially to the cryptographic libraries used by the RADIUS software. Most RADIUS server software do not implement the cryptographic functions themselves, but rely completely on the cryptographic library to provide the needed TLS tunnel. This implies that the default settings of the cryptographic library need to meet the security requirements and implement the according countermeasures.

Apart from the use of current software, a number of other settings can be set to improve the security. Where possible, the relevant configuration options for FreeRADIUS are also given.

First, the TLS version should be set to a minimum of TLSv1.2. Once TLSv1.3 is specified for EAP-TLS and implementations support it, it should be used. Since disabling TLSv1.0 and TLSv1.1 will exclude some devices, this should be executed after announcing this first, leaving owners of old devices a bit of time to update their devices. In FreeRADIUS this can be achived by setting `tls_min_version` to `"1.2"`. To prevent the RADIUS server from negotiating TLSv1.3 before the specification is completed, `tls_max_version` should also be set to `"1.2"`, until an update for FreeRADIUS capable of EAP-TLS with TLSv1.3 is installed.

The available cipher suites can be configured in most RADIUS software available. Here, a cipher suite set should be set that allows only PFS cipher suites. Since only a negligable amout of devices does not support PFS cipher suites, this step could be done without prior notification. Clients capable of PFS are then forced to use a PFS cipher suite, even if non-PFS cipher suites are preferred by the client. A possible cipher set in FreeRADIUS can be used by setting `cipher_list` to `"DEFAULT !kRSA !PSK !SRP !SSLv3"`. This enables the default cipher suites from OpenSSL without cipher suites with key exchange over RSA (`!kRSA`), Pre-Shared Key algorithms (`!PSK`), Secure Remote Password (`!SRP`) and cipher suites used in SSLv3 (`!SSLv3`).

To allow clients to choose a specific ECDHE curve for the key exchange, the servers should allow more then one elliptic curve. For FreeRADIUS, the default configuration always uses `secp256r1`. To allow all curves activated by default, the option `ecdh_curve` can be set to the empty string (`""`).

The RADIUS server could also simply reject all clients without support for Extended Master Secret. As the analysis has shown that only a negligible amount of clients do not support Extended Master Secret, hence a Client Hello without the extension present is either originated from an outdated client or an attacker attempting a TLS Proxy attack. Enforcing this is currently not possible in FreeRADIUS. Different RADIUS server software may allow for such a configuration.

The same can be said for the support for secure renegotiation. Here the server could also choose to reject all clients not capable of supporting secure renegotiation. A second countermeasure would be to ensure that the server does not allow renegotiation at all, which would prevent any attacks with a triple handshake. Like the previously discussed countermeasure, this too is not yet configurable in FreeRADIUS.

If the server software supports it, OCSP stapling should be enabled. This currently does not provide a security improvement, since missing OCSP staples will not considered negatively by most clients. However, this could change in the future, if OCSP stapling can be enforced by the certificate or by configuration. FreeRADIUS v3 does not support OCSP stapling, although it supports OCSP to check the revocation status of presented client certificates. The handling of OCSP MustStaple is a good example of the need to introduce security mechanisms bilaterally.

## 5.3 Limitations of this thesis, open topics and future research

Within the thesis, several flaws of the analysis tool and missing features have been mentioned. Some of these have not been implemented due to time constraints, some were considered out of scope for this thesis. Within the thesis, a number of additional topics of research have been mentioned. Discussing those will conclude this thesis.

The first limitation concerns to the error handling of the analysis tool. Although the tool was improved several times, the number of errored packets is still relatively high, as mentioned in subsection 3.4.4. Fixing the bugs and flaws causing these errors would have taken considerable time. This may be done for a future release, but was considered too costly for this thesis.

The analysis tool is currently not capable of analyzing a Diffie-Hellman key exchange not based on elliptic curves. This feature should be added in a future version. Especially the used key lengths could also provide a new insight and may uncover some security issues.

A feature missing for a better analysis, as mentioned in subsection 3.4.1, is the separate capture and storage of unanswered or rejected TLS CLient Hellos. This would prevent the mentioned confirmation bias in the data, as it would include both clients too insecure and too secure to negotiate a TLS connection with their respective server. Additionally, the capturing of different TLS Alerts seen in response to different TLS messages would give additional information e.g. about rejected TLS parameters, invalid certificates, possible implementation errors and several more details.

As this thesis explicitly did not deal with certificates, this would be an additional topic for further research. This would include questions for the used public key algorithms, used key lengths, chosen curves for certificates using elliptic curves, usage of own certificate authorities and several more. To simplify the analysis and to provide a possible monitoring for the eduroam community, a certificate transparency log intigrated into the analysis tool could be used to save the seen certificates. An attempt to implement this has been made by us already but was aborted after a short period of time.

With the parsing of certificates, the sent OCSP staples could also be analysed. This could include the issuing entity, the revocation status and validity time spans.

As the specification of EAP-TLS with TLSv1.3 is in its final stage, a further improvement of the analysis tool would be to support analysing TLSv1.3 handshakes, where possible. Since most parts of the TLS handshake in TLSv1.3 are encrypted, the passive analysis would reveal much less information than with previous TLS versions.

Another field of research may be the assessment of the actual security status of RADIUS servers. As this thesis focuses on a passive analysis, possible misconfigurations of RADIUS servers can remain hidden, if no client ever negotiates these parameters. Testing this would require an active component, which probes the RADIUS servers with different TLS Client Hello messages and assessing their response. Implementing this would also improve the ability for RADIUS server administrators to become aware of issues, especially if this would be made available for the eduroam community. This already exists for many other use cases of TLS (e.g. with HTTPS for web servers[1] or with SMTP for mail servers[2]), but not for EAP-TLS.

[1] e.g. `https://ssllabs.com`
[2] e.g. `https://ssl-tools.net`

# Bibliography


[1] Alberto Bartoli, Eric Medvet, and Filippo Onesti. Evil twins and WPA2 Enterprise: A coming security disaster? *Computers & Security*, 74:1–11, 2018.

[2] Alberto Bartoli, Eric Medvet, Andrea De Lorenzo, and Fabiano Tarlao. (In)Secure Configuration Practices of WPA2 Enterprise Supplicants. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Allan Rubens, Carl Rigney, Steve Willens, and William A. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865, June 2000.

[4] Dimitrios Pendarakis, Dr. Raj Yavatkar, and Dr. Roch Guerin. A Framework for Policy-based Admission Control. RFC 2753, January 2000.

[5] Klaas Wierenga, Mike McCauley, Stefan Winter, and Stig Venaas. Transport Layer Security (TLS) Encryption for RADIUS. RFC 6614, May 2012.

[6] John Vollbrecht, James D. Carlson, Larry Blunk, Dr. Bernard D. Aboba, and Henrik Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748, June 2004.

[7] IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control. *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pages 1–289, 2020.

[8] IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements. *IEEE Std 802.11i-2004*, pages 1–190, 2004.

[9] Daniel Simon, Ryan Hurst, and Dr. Bernard D. Aboba. The EAP-TLS Authentication Protocol. RFC 5216, March 2008.

[10] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[11] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, April 2006.

[12] Christopher Allen and Tim Dierks. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[13] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[14] John Preuß Mattsson and Mohit Sethi. Using EAP-TLS with TLS 1.3 (EAP-TLS 1.3). Internet-Draft draft-ietf-emu-eap-tls13-18, Internet Engineering Task Force, July 2021. Work in Progress.

[15] Daniel Simon and Dr. Bernard D. Aboba. PPP EAP TLS Authentication Protocol. RFC 2716, October 1999.

[16] Klaas Wierenga, Stefan Winter, and Tomasz Wolniewicz. The eduroam Architecture for Network Roaming. RFC 7593, September 2015.

[17] Carsten Bormann and Jan-Frederik Rieckers. WiFi authentication: Some deployment observations from eduroam. May 2019. `https://www.iab.org/wp-content/IAB-uploads/2019/05/p16-bormann-wifi.txt`.

[18] Jan-Frederik Rieckers. (Probleme beim) Widerruf von Zertifikaten für RADIUS-Server. *71. DFN-Betriebstagung*, September 2019. `https://www.dfn.de/fileadmin/3Beratung/Betriebstagungen/bt71/BT71_MobileIT_Zertifikatswiderruf_Rieckers.pdf`.

[19] Jan-Frederik Rieckers. Filtern und Analyse von TLS-(EAP)-Handshakes beim eduroam-Login. *73. DFN-Betriebstagung*, September 2020. `https://www.dfn.de/fileadmin/3Beratung/Betriebstagungen/BT73/BT73_MobileIT_EAP-TLS_Rieckers.pdf`.

[20] Ralf Paffrath and Jan-Frederik Rieckers. Eduroam – ein sicherer Dienst. *DFN Mitteilungen*, 98:41–44, December 2020. `https://www.dfn.de/fileadmin/5Presse/DFNMitteilungen/DFN_98_download.pdf`.

[21] One Way, Marsh Ray, Steve Dispensa, and Eric Rescorla. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746, February 2010.

[22] Donald E. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, January 2011.

[23] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Alfredo Pironti, Adam Langley, and Marsh Ray. Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension. RFC 7627, September 2015.

[24] Bodo Moeller, Nelson Bolyard, Vipul Gupta, Simon Blake-Wilson, and Chris Hawk. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492, May 2006.

[25] Daniel Kahn Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). RFC 7919, August 2016.

[26] Yoav Nir, Simon Josefsson, and Manuel Pégourié-Gonnard. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier. RFC 8422, August 2018.

[27] Mhd Yassin Alshater. Verbesserung der Resistenz gegen Evil-Twin-Angriffe in Enterprise-WLANs im Beispiel eduroam, 2020. Bachelor's thesis.

[28] eduroam Policy Service Definition. *Version 2.8*, July 2012. `https://www.eduroam.org/wp-content/uploads/2016/05/GN3-12-192_eduroam-policy-service-definition_ver28_26072012.pdf`.

[29] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525, May 2015.

[30] Federal Office for Information Security Germany. Technical Guideline TR-02102-2 – Use of Transport Layer Security (TLS), March 2021.

[31] Kathleen Moriarty and Stephen Farrell. Deprecating TLS 1.0 and TLS 1.1. RFC 8996, March 2021.

[32] Andreas Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, 48(3):269–286, Sep 2008.

[33] Yaron Sheffer, Ralph Holz, Peter Saint-Andre, and Thomas Fossati. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). Internet-Draft draft-ietf-uta-rfc7525bis-01, Internet Engineering Task Force, July 2021. Work in Progress.

[34] David Taylor, Trevor Perrin, Thomas Wu, and Nikos Mavrogiannopoulos. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054, November 2007.

[35] Juliano Rizzo and Thai Duong. The CRIME attack. In *ekoparty security conference*, volume 2012, 2012.

[36] Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113, 2014.

[37] Openssl changelog. `https://www.openssl.org/news/changelog.html`. last accessed on 2021-07-12T16:00Z.

[38] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, January 2010. `https://secg.org/sec2-v2.pdf`.

[39] Johannes Merkle and Manfred Lochter. Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS). RFC 7027, October 2013.

[40] Daniel J. Berstein. Curve25519: new Diffie-Hellman speed records, February 2006. `https://cr.yp.to/ecdh/curve25519-20060209.pdf`.

[41] Mike Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `https://eprint.iacr.org/2015/625`.

[42] Michael Williams, Michael Tüxen, and Robin Seggelmann. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. RFC 6520, February 2012.

[43] The Heartbleed Bug. `https://heartbleed.com/`. last accessed on 2021-07-13T14:30Z.

[44] Peter Gutmann. Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7366, September 2014.

[45] Bodo Moeller and Adam Langley. TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks. RFC 7507, April 2015.

# Lists

## List of Figures



## List of Tables

# Glossary

**AAA** Authentication, Authorization, Accounting. Generic term used to describe protocols, procedures or infrastructure capable of doing all three aspects. 4, 55

**EAP** Extensible Authentication Protocol. Specified in RFC . 1, 5

**EAP-TLS** An EAP authentication protocol. Specified in RFC 5216. Can be used either as authentication method itself if used with client certificates or as basis for other authentication methods like EAP-TTLS or EAP-PEAP. 1

**ECDHE** Elliptic Curve Diffie-Hellman with ephemeral parameters. 18

**Identity Provider (IdP)** Entity holding the account of the user. 1, 15, 25

**Message Authentication Code (MAC)** A cryptographic method securing integrity and authenticity of a message using a shared secret. 5

**MS-CHAP-V2** A challenge-response authentication protocol developed by Microsoft. 11

**Network Access Server (NAS)** Device providing access to the restricted resource. In the eduroam environment usually the Access Point. 4

**OCSP** Online Certificate Status Protocol. Specified in RFC 6960. Protocol to check the revocation status of a certificate using a trusted server, usually operated by the respective Certificate Authority. 1, 17

**PAP** Password Authentication Protocol. A simple protocol for cleartext transmission of user credentials. Usually used inside an encrypted tunnel like EAP-TLS. 11

**PEAP** Protected EAP. An EAP Authentication protocol based on EAP-TLS. Developed by Microsoft. 1, 7

**Policy Decision Point (PDP)** Point where the decision about an authorization is made, e.g. if the requesting entity is allowed in the network and if additional restrictions should apply. 4

**Policy Enforcement Point (PEP)** Point where the authorization decision made by the PDP is executed. 4

**Pre Master Secret (PMS)** The secret used in TLS to calculate the master secret. Depending on the key agreement method of the cipher suite, either chosen by the client and sent encrypted to the server or the output of the DH/ECDH function. 9

**Pseudo Random Function (PRF)** A cryptographic function capable of generating deterministic pseudo random output with a given input. Usually achieved by using crypgoraphic Hash functions like MD5, SHA-1 or SHA-2. 9, 28

**RADIUS** Remote Authentication Dial In User Service. Specified in RFC 2865. A protocol from the AAA family. Capable of doing Authentication, Authorization and Accounting. 1, 4

**Secure Socket Layer (SSL)** Predecessor protocol of TLS. Developed by Netscape. Specifications can be found in the RFC 6101 (SSLv3). 7

**Service Provider (SP)** Entity providing Internet access for users. 1, 15

**Signaling Cipher Suite Value (SCSV)** A specific value included in the list of supported cipher suites to signal the server. Used for signaling support for specific features (secure renegotiaton) or a fallback to an earlier TLS version. 18, 30, 45

**SSID** Service Set Identifier. A commonly humanreadable name for a wireless network. 22

**TLS** Transport Layer Security. Specified in RFC 2246 (TLSv1.0), RFC 4346 (TLSv1.1), RFC 5246 (TLSv1.2), RFC 8446 (TLSv1.3). A security protocol to ensure authenticity, integrity and confidentiality of transmitted data. 1

**TLV** Type-Length-Value. Format to store or transmit type-value-pairs with variable length of value. 4

**TTLS** Tunneled TLS. Specified in RFC 5281. An EAP Authentication protocol based on EAP-TLS. 1, 7

**VLAN** Virtual Local Area Network. Allows separation of Local Area Networks while using a shared connection. 10

# Appendix A

# Trademark notices

The eduroam logo and eduroam are registered trademarks of the GÉANT Association.

Apple, iPhone, iPad, mac, macOS are trademarks of Apple Inc., registered in the U.S. and other countries and regions.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license by Apple Inc.

# Appendix B

# Inquiries to facilities - Questions

This is the content of the mail to the different institutions. Since the bachelor thesis was written in cooperation with the German research network (Deutsches Forschungsnetz e. V. DFN), the inquiries were made in German.

> Betreff: Mögliche Sicherheitsprobleme im RADIUS-Server - Fragen für eine Bachelorarbeit
>
> Sehr geehrte Damen und Herren,
>
> im Rahmen meiner Bachelor-Arbeit beschäftige ich mich aktuell mit der Sicherheit im eduroam, im Speziellen mit dem Login über TLS-basierte EAP-Methoden. Ich möchte in der Arbeit einen Überblick über aktuelle Sicherheitsprobleme geben und dabei helfen, diese Probleme zu lösen. Erste Resultate aus dieser Analyse habe ich bereits auf der 73. DFN-Betriebstagung [1] und in einem Artikel im DFN-Mitteilungsblatt [2] veröffentlicht.
>
> Ich schreibe Ihnen nun, weil mir bei Ihrer Einrichtung mögliche Sicherheitsprobleme aufgefallen sind, die weiter unten aufgelistet sind. Diese Probleme sind in der Regel keine kritischen Lücken. Trotzdem möchte ich Sie auf diese Lücken aufmerksam machen, damit die Sicherheit im eduroam weiter verbessert werden kann.
>
> Um einen Überblick über die eduroam-Landschaft in Deutschland zu bekommen, würde ich mich sehr freuen, wenn sie mir ein paar Details zu Ihrem eduroam-Setup zukommen lassen können. In meiner Bachelorarbeit werde ich Handlungsempfehlungen für eduroam-Admins veröffentlichen und möchte daher einen möglichst guten Überblick über genutzte Software, Betriebssysteme oder Konfigurationen erhalten. Die Teilnahme an dieser kleinen Umfrage ist natürlich freiwillig.
>
> Es interessiert mich hierbei vor allem welches Betriebssystem und welche RADIUS-Software in welcher Version Sie einsetzen, sowie die Version der eingesetzten Verschlüsselungs-Bibliothek (meist OpenSSL). Wenn Sie die TLS-Konfiguration des Servers angepasst haben, würde ich mich hier für die Konfiguration interessieren. Für eine Einordnung würde ich mich auch freuen, wenn Sie mir grob die Anzahl an eduroam-Nutzer*innen pro Tag an Ihrer Einrichtung nennen können.
>
> Die Einrichtung werde ich in der Bachelorarbeit natürlich nicht namentlich nennen, weder in der Auflistung von Sicherheitsproblemen noch bei der Beantwortung der nachfolgenden Fragen. Sämtliche Rückmeldungen auf diese Umfrage werden absolut vertraulich behandelt und dienen nur der Forschung im Rahmen meiner Bachelorarbeit. In der Bachelorarbeit werden lediglich aggregierte Informationen aus den Rückmeldungen enthalten sein.

Über eine Antwort bis zum 9.7. würde ich mich sehr freuen, damit ich noch ausreichend Zeit habe, die Resultate dieser Umfrage in meine Arbeit einzuarbeiten.

Sollten Sie Fragen zu den beschriebenben Problemen haben, zögern Sie nicht uns anzuschreiben. Das DFN gibt auch gerne Hilfestellung bei Konfigurationsänderungen, wenn es uns möglich ist.

Ich bedanke mich schon jetzt für Ihre Mühen.
Mit freundlichen Grüßen

Jan-Frederik Rieckers
- stud. Hilfskraft -

[1]:https://www.dfn.de/fileadmin/3Beratung/Betriebstagungen/BT73/BT73_MobileIT_EAP-TLS_Rieckers.pdf
[2]:https://www.dfn.de/fileadmin/5Presse/DFNMitteilungen/DFN_Mitteilungen_98.pdf

Depending on the found security problems the following additional information has been sent in in the footer of the mail:

**Veraltete TLS-Version (1.0)**
Ihr RADIUS-Server handelt nicht die aktuelle TLS-Version für EAP-TLS (TLS 1.2) sondern nur TLS 1.0 aus. Dies deutet auf eine veraltete Server-Software hin.

TLS 1.0 benutzt die veralteten Hash-Algorithmen SHA-1 und MD5 für die Berechnung der kryptographischen Schlüssel und zur Integritätsprüfung. Die Art und Weise der Benutzung der Hash-Funktionen in diesem Kontext sorgt nicht dafür, dass die gefundenen Angriffe gegen diese Funktionen die Sicherheit stark beeinträchtigen würden, nichtsdestotrotz gelten TLS 1.0 und TLS 1.1 als veraltet und Clients könnten die Verbindung ablehnen, wenn nicht TLS 1.2 zur Verfügung steht.

**Veraltete TLS-Version (1.1)**
Ihr RADIUS-Server handelt nicht die aktuelle TLS-Version für EAP-TLS (TLS 1.2) sondern nur TLS 1.1 aus. Dies deutet auf eine veraltete Server-Software hin.

TLS 1.1 benutzt die veralteten Hash-Algorithmen SHA-1 und MD5 für die Berechnung der kryptographischen Schlüssel und zur Integritätsprüfung. Die Art und Weise der Benutzung der Hash-Funktionen in diesem Kontext sorgt nicht dafür, dass die gefundenen Angriffe gegen diese Funktionen die Sicherheit stark beeinträchtigen würden, nichtsdestotrotz gelten TLS 1.0 und TLS 1.1 als veraltet und Clients könnten die Verbindung ablehnen, wenn nicht TLS 1.2 zur Verfügung steht.

**Veraltete TLS-Version (1.1) (inkonsistent)**
Ihr RADIUS-Server handelt nicht die aktuelle TLS-Version für EAP-TLS (TLS 1.2) sondern nur TLS 1.1 aus. Diese Aushandlung geschieht allerdings nur mit einem Teil der Clients. Dies deutet auf eine veraltete Server-Software auf einem/mehrerer Ihrer RADIUS-Server oder inkonsistente Konfigurationen hin.

TLS 1.1 benutzt die veralteten Hash-Algorithmen SHA-1 und MD5 für die Berechnung der kryptographischen Schlüssel und zur Integritätsprüfung. Die Art und Weise der Benutzung der Hash-Funktionen in diesem Kontext sorgt nicht dafür, dass die gefundenen Angriffe gegen diese Funktion die Sicherheit stark beeinträchtigen würden, nichtsdestotrotz gelten TLS 1.0 und TLS 1.1 als veraltet und Clients könnten die Verbindung ablehnen, wenn nicht TLS 1.2 zur Verfügung steht.

**Kein Support für Extended Master Secret**
Ihr RADIUS-Server scheint keine Unterstützung für Extended Master Secret (RFC 7627) zu

haben. Dies deutet auf eine ältere Version der kryptographischen Bibliotheken oder veraltete Server-Software hin.

Das Extended Master Secret erlaubt eine verbesserte Kalkulation der kryptographischen Schlüssel. Ohne diese Erweiterung ist ein TLS-Proxy-Angriff möglich, in dessen Folge ein Angreifer die kryptographischen Schlüssel der Verbindungen zum Client und zum Server synchronisieren kann, sollte der Client das Zertifikat des Angreifers als gültig akzeptieren. Da die WLAN-Verschlüsselung aus den kryptogrpahischen Schlüsseln der TLS-Verbindung abgeleitet wird, könnte ein Angreifer in der Folge die WLAN-Verschlüsselung brechen oder sich bei einer Neuanmeldung als Client ausgeben, falls Session Resumption aktiv ist.

**Kein Support für Extended Master Secret (inkonsistent)**
Einige Ihrer RADIUS-Server scheinen keine Unterstützung für Extended Master Secret (RFC 7627) zu haben. Dies deutet auf eine ältere Version der kryptographischen Bibliotheken oder veraltete Server-Software auf einem oder mehreren Ihrer Server oder eine inkonsistente Konfiguration hin.

Das Extended Master Secret erlaubt eine verbesserte Kalkulation der kryptographischen Schlüssel. Ohne diese Erweiterung ist ein TLS-Proxy-Angriff möglich, in dessen Folge ein Angreifer die kryptographischen Schlüssel der Verbindungen zum Client und zum Server synchronisieren kann, sollte der Client das Zertifikat des Angreifers als gültig akzeptieren. Da die WLAN-Verschlüsselung aus den kryptogrpahischen Schlüsseln der TLS-Verbindung abgeleitet wird, könnte ein Angreifer in der Folge die WLAN-Verschlüsselung brechen oder sich bei einer Neuanmeldung als Client ausgeben, falls Session Resumption aktiv ist.

**Kein Support für Secure Renegotiation**
Ihr RADIUS-Server scheint keine Unterstützung für Secure Renegotiation (RFC 5746) zu haben. Dies deutet auf veraltete Server-Software hin.

Ohne Secure Renegotiation kann ein Angreifer eine TLS-Verbindung von einem Client in seine eigene einspleißen, wodurch der Angreifer Daten an den Server schicken kann, die der Server dem späteren Client zuordnet. Ob hierdurch ein konkreter Angriff auf den eduroam-Login möglich ist, wurde nicht erforscht.

**Verschlüsselung 3DES ausgehandelt**
Ihr Server bietet Verschlüsselung auf Basis von 3DES an und handelt diese Verschlüsselung auch bei Verfügbarkeit von AES-Verschlüsselung auf Seiten des Clients aus. Dies deutet auf veraltete Software oder fehlerhafte TLS-Konfiguration hin.

Die Verschlüsselung 3DES bietet eine theoretische Sicherheit von 112 bit. Auch wenn die Bit-Sicherheit noch als ausreichend angesehen werden kann, wird die Verwendung von 3DES u.a. aufgrund von verschiedenen Schwächen im unterliegenden DES-Algorithmus von der IETF und dem BSI nicht mehr empfohlen.

**Akzeptanz von alten Clients mit 3DES**
Ihr RADIUS-Server bietet Verschlüsselung auf Basis von 3DES an. Diese Verschlüsselung wird augenscheinlich nur ausgehandelt, wenn Clients keine besseren Verschlüsselungsmethoden unterstützen.

Die Verschlüsselung 3DES bietet eine theoretische Sicherheit von 112 bit. Auch wenn die Bit-Sicherheit noch als ausreichend angesehen werden kann, wird die Verwendung von 3DES u.a. aufgrund von verschiedenen Schwächen im unterliegenden DES-Algorithmus von der IETF und dem BSI nicht mehr empfohlen. Clients, die lediglich 3DES als Verschlüsselung anbieten sind daher vermutlich stark veraltet und könnten damit auch ein mögliches Sicherheitsrisiko darstellen.

**Verschlüsselung RC4 ausgehandelt**
Ihr RADIUS-Server bietet die Verschlüsselungsmethode RC4 an. Dies deutet auf veraltete Software hin.

Der RC4-Algorithmus gilt als gebrochen und sollte unter keinen Umständen mehr verwendet werden. Die IETF hat in RFC 7465 die Verwendung von RC4 in TLS untersagt, das BSI rät auch dringend von einer Verwendung von RC4 ab.

**Keine AES-Verschlüsselung**
Ihr RADIUS-Server bietet keine Verschlüsselung auf Basis von AES an. Dies deutet auf veraltete Software oder eine fehlerhafte TLS-Konfiguration hin.

Der Verschlüsselungsalgorithmus AES ist der aktuelle Standard für symmetrische Verschlüsselung und sollte in jedem Fall unterstützt werden.

**Keine Perfect Forward Secrecy**
Ihr RADIUS-Server handelt keine Verschlüsselungsmethoden aus, die Perfect Forward Secrecy unterstützen. Dies deutet auf veraltete Server-Software oder eine fehlerhafte TLS-Konfiguration hin.

Die Nutzung von Perfect Forward Secrecy-Verschlüsselungsmethoden verhindert eine nachträgliche Entschlüsselung von aufgezeichneten Verbindungen, sollte der private Schlüssel des Servers kompromittiert werden. Verschlüsselungen auf Basis von Diffie-Hellmann (ECDHE oder DHE) berechnen den kryptographischen Schlüssel auf Basis von ausgetauschten Parametern, die für jede Verbindung neu generiert und danach verworfen werden. Hierbei ist eine nachträgliche Entschlüsselung auch bei Bekanntwerden des privaten Schlüssels nicht möglich.

# Appendix C

# Data Structure

This is listing of the different fields used to save the data.

## `meta` – Meta-Information about the communication and data storage

| Field name | Type | Description |
|---|---|---|
| `.last_seen` | Timestamp | Time when the client was last seen |
| `.oui` | String | Organizationally unique identifier, equal to the first 3 bytes of the MAC address |
| `.vendor` | String | Organization associated with the OUI, determined from `meta.oui` |
| `.realm` | String | Realm part of the user name |
| `.realm_tld` | String | Top Level Domain of the realm, determined from `meta.realm` |
| `.capture_ver` | Long | Capture version. Increased every time, new data is included in the data. |
| `.scheme_ver` | Long | Scheme version. Increased every time, new fields are added, that can be computed from existing data. |

## `radsec.information` – Information about the RADIUS communication when captured via radsec

| Field name | Type | Description |
|---|---|---|
| `.roundtrips` | Long | Number of packets sent in total[1] |
| `.accept` | Boolean | `true` if the communication ended with an Access-Accept, `false` if Access-Reject. |
| `.time` | Float | Time in seconds from first to last packet. |
| `.max_client_pkt_size` | Long | Size in bytes of the largest RADIUS packet sent by the client. |
| `.max_server_pkt_size` | Long | Size in bytes of the largest RADIUS packet sent by the server |
| `.total_client_pkt_size` | Long | Total bytes sent by the client |
| `.total_server_pkt_size` | Long | Total bytes sent by the server |

## `radius.information` – Information about the RADIUS communication when captured via RADIUS

Equal to the contents of `radsec.information`.

## `eap.information` – Information about the EAP communication

| Field name | Type | Description |
|---|---|---|
| `.initial_eaptype` | String | EAP Type initially offered by the EAP server |
| `.wanted_eaptypes` | Array of Strings | EAP Types offered by the client, if the client sent a EAP-NAK. If the client chose the EAP method offered by the server initially, this field remains empty. |
| `.actual_eaptype` | String | Chosen EAP Type for the following communication |
| `.roundtrips` | Long | Number of EAP packets sent in total[1] |
| `.max_client_pkt_size` | Long | Size in bytes of the largest EAP packet sent by the client. |
| `.max_server_pkt_size` | Long | Size in bytes of the largest EAP packet sent by the server. |
| `.total_client_pkt_size` | Long | Total bytes of EAP data sent by the client |
| `.total_server_pkt_size` | Long | Total bytes of EAP data sent by the server |

[1] The name of this field was chosen incorrectly. To do justice to the name, the current value would have to be divided by 2

## `tls.tlsclienthello` – Analysed data from the TLS Client Hello

| Field name | Type | Description |
|---|---|---|
| `.version` | String | TLS version of the Client Hello |
| `.ciphersuites` | Array of Strings | List of available cipher suites as hex string |
| `.cipherdata` | Hash | Container to save data about the offered cipher suites. See next section for details. |
| `.compression.names` | Array of Strings | List of available compressions |
| `.compression.order` | String | Order of available compressions |
| `.extendedmastersecret` | Boolean | `true` if the client included the Extended Master Secret extension in its client hello |
| `.all_extensions` | Array of Strings | List of all TLS extensions included in the client hello |
| `.extensionorder` | String | Concatenated string of all included extensions in the order of their occurrence in the client hello |
| `.renegotiation` | Boolean | `true` if the client signaled support for secure renegotiation, either by SCSV or extension |
| `.servername` | Array of Strings | List of included Server names in the `SERVER_NAME` extension, if present |
| `.signaturealgorithms` | Array of Strings | List of available signature algorithms |
| `.statusrequest` | Array of Strings | List of requested status (either OCSP or empty) |
| `.supportedgroups` | Array of Strings | List of supported groups for Diffie-Hellman key exchange |
| `.fingerprinting` | Hash | Container for storage of fingerprinting details |
| `.v2` | String | SHA-2 Hash of several details of the Client Hello. |
| `.osdetails.os` | String | Operating System (e.g. Windows/Linux/Android) |
| `.osdetails.os_version` | String | Version of OS, if known |
| `.osdetails.detail` | String | Details of OS-Version, e.g. data source, OpenSSL version |

## `tls.tlsclienthello.cipherdata` – Additional data to support statistical analysis

| Field name | Type | Description |
|---|---|---|
| `.all_auth` | Array of Strings | List of all authentication algorithms offered |
| `.all_auth_list` | String | Concatenated string of previous list |
| `.all_encr` | Array of Strings | List of all encryption algorithms offered |
| `.all_encr_list` | String | Concatenated string of previous list |
| `.all_keyx` | Array of Strings | List of all key exchange algorithms offered |
| `.all_keyx_list` | String | Concatenated string of previous list |
| `.all_mac` | Array of Strings | List of all MAC algorithms offered. (PRF if MAC is HMAC) |
| `.all_mac_list` | String | Concatenated string of previous list |
| `.export` | Boolean | `true` if the offered cipher suites contained cipher suites with export level security |
| `.broken` | Boolean | `true` if the offered cipher suites contained cipher suites with broken algorithms. These are `RC-2`, `RC-4`, `DES` |
| `.outdated` | Boolean | `true` if the offered cipher suites contained cipher suites with outdated algorithms. These are the broken algorithms and `IDEA` and `MD5` |
| `.anull` | Boolean | `true` if the offered cipher suites contained an `aNULL` cipher suite |
| `.enull` | Boolean | `true` if the offered cipher suites contained an `eNULL` cipher suite |
| `.des` | Boolean | `true` if the offered cipher suites contained a cipher suite with `DES` as encryption algorithm. |
| `.rc4` | Boolean | `true` if the offered cipher suites contained a cipher suite with `RC-4` as encryption algorithm |
| `.tripledes` | Boolean | `true` if the offered cipher suites contained a cipher suite with `3DES` es encryption algorithm |
| `.only_pfs` | Boolean | `true` if only cipher suites with perfect forward secrecy were offered. |

## Continuation for `tls.tlsclienthello.cipherdata`

| Field name | Type | Description |
|---|---|---|
| `.pfs_avail` | Boolean | `true` if the offered cipher suites contained cipher suites with support for perfect forward secrecy |
| `.humanreadable` | Array of String | list of all offered cipher suites with their human readable name as published in the IANA list |
| `.cipherset` | String | concatenated string of all offered cipher suites in their hex representation |
| `.cipherset_length` | Long | Number of offered cipher suites including SCSV |
| `.cipherset_length_noscsv` | Long | Number of offered cipher suites, not counting SCSV |
| `.max_sec_lvl` | Long | The maximum level of security (bits of key) offered by the cipher suites |
| `.min_sec_lvl` | Long | The minimum level of security (bits of key) offered by the cipher suites (Caution: `RC4` with 128bit key is treated as 128 bit here, although it is considered broken) |
| `.signature_algorithm_set` | String | Concatenated string of all offered signature algorithms |
| `.supported_group_set` | String | Concatenated string of all supported groups |

## `tls.tlsserverhello` – Analysed data from the TLS Server Hello

| Field name | Type | Description |
|---|---|---|
| `.version` | String | TLS version of the Server Hello |
| `.downgrade` | String | `DOWNGRD1` or `DOWNGRD0` if the server random included downgrade indication, `NONE` otherwise |
| `.cipher` | String | Chosen cipher suite as Hex string |
| `.cipherdata` | Hash | Container to store details about the chosen cipher suite |
| `.FS` | Boolean | `true` if the chosen cipher suite provides perfect forward secrecy |
| `.auth` | String | Authentication algorithm used in the chosen cipher suite |
| `.encry` | String | Encryption algorithm used in the chosen cipher suite |
| `.keyx` | String | Key exchange algorithm used in the chosen cipher suite |
| `.name` | String | Human readable name of the chosen cipher suite |
| `.compression.code` | Long | Code of the chosen compression algorithm |
| `.compression.name` | String | Name of the chosen compression algorithm |
| `.all_extensions` | Array of Strings | List of all TLS extensions sent by the server |
| `.extensionorder` | String | Concatenated string of the names of all extensions sent by the server |
| `.keyexchange.curve_name` | String | Name of the used Elliptic Diffie-Hellman Curve if ECDHE key exchange is used |
| `.keyexchange.sig_scheme` | String | Signature scheme used to authenticate the ECDHE parameters |
| `.renegotitation` | Boolean | `true` if the server indicated support for secure renegotiation |
| `.extendedmastersecret` | Boolean | `true` if the server indicated support for Extended Master Secret |
| `.stapling` | Boolean | `true` if a certificate status was included in the server handshake messages |
| `.certificate` | Hash | Container for storage of certificate details. See next section for details |

## `tls.tlsserverhello.certificate` – Details about the sent certificate

| Field name | Type | Description |
|---|---|---|
| `.public_chain` | Hash | Container for the chain against publicly trusted certificates |
| `.additional_chain` | Hash | Container for the chain against trusted certificates together with previously observed intermediate certificates. |
| *Chain Hash (public and additional chain)* | | |
| `.array` | Array of Strings | Array of the subjects of the certificate |
| `.by_index.<INDEX>` | String | Subjects of the certificate, ordered from the trust anchor to the leaf certificate |
| `.public_trust_anchor` | String | Subject of the trust anchor determined by the public chain. |
| `.additional_trust_anchor` | String | Subject of the trust anchor determined by the additional chain. |
| `.public_trusted` | Boolean | `true` if the sent leaf certificate is trusted using the public chain for verification |
| `.additional_trusted` | Boolean | `true` if the sent leaf certificate is trusted using the additional chain for verification |
| `.complete_public_chain_length` | Long | Length of the certificate chain when validated against the public chain |
| `.complete_additional_chain_length` | Long | Length of the certificate chain when validated against the additional chain |
| `.sent_chain_length` | Long | Length of the certificate chain sent by the server |

# Appendix D

# Absolute numbers for percentages

These tables contain the absolute numbers used for percentage calculations within the thesis.

## D.1 Per-Client data

| Data Item | Uni | % | DFN | % |
|---|---|---|---|---|
| Total items | 29,673 | | 1,421,346 | |
| **Client TLS Versions** | | | | |
| TLSv1.3 | 114 | 0.38 | 2,015 | 0.14 |
| TLSv1.2 | 28,939 | 97.53 | 1,379,038 | 97.02 |
| TLSv1.1 | - | - | 253 | 0.02 |
| TLSv1.0 | 620 | 2.09 | 40,040 | 2.82 |
| **OS Version of Clients not capable of TLSv1.2** | | | | |
| Total | 620 | 2.09 | 40,293 | |
| ↪Unknown | 129 | ↪20.81 | 18,150 | ↪45.05 |
| ↪iPhone 5s | 137 | ↪22.10 | 6,956 | ↪17.26 |
| ↪iPhone 4s | 84 | ↪13.55 | 2,969 | ↪7.37 |
| ↪Android 5.1.1 | 144 | ↪23.23 | 6,343 | ↪15.74 |
| ↪Android 4.1.2 | 44 | ↪7.10 | 3,943 | ↪9.79 |
| ↪Android 4.0.4 | 2 | ↪0.32 | 73 | ↪0.18 |
| ↪Android 2.2 | - | - | 4 | ↪0.01 |
| ↪Windows 8.1 | 80 | ↪12.90 | 1,855 | ↪4.60 |
| **Length of cipher suite list (if >1%)** | | | | |
| 21 | 20,849 | 70.26 | 958,775 | 67.46 |
| 15 | 3,971 | 13.38 | 227,019 | 15.97 |
| 30 | 1,873 | 6.31 | 88,704 | 6.24 |
| 20 | 800 | 2.70 | 39,343 | 2.77 |
| 31 | 658 | 2.22 | 34,701 | 2.44 |
| 27 | 524 | 1.77 | 22,118 | 1.56 |
| 29 | 265 | 0.89 | 14,940 | 1.05 |

| Data Item | Uni | % | DFN | % |
|---|---|---|---|---|
| Total items | 29,673 | | 1,421,346 | |
| **Properties of offered cipher suites** | | | | |
| Contains EXPORT | 190 | 0.64 | 10,546 | 0.74 |
| Contains outdated algorithms | 3,994 | 13.46 | 199,127 | 14.01 |
| Max Security Level <256 bit | 4 | 0.01 | 1,444 | 0.10 |
| Min Security Level min 112 bit & no RC4 | 25,678 | 86.54 | 1,222,206 | 85.99 |
| **Secure Renegotiation** | | | | |
| Missing support on Client side | 872 | 2.94 | 1,827 | 0.13 |
| ↪Thereof supporting TLSv1.2 or higher | 866 | ↪99.31 | 27 | ↪1.48 |
| **Extended Master Secret** | | | | |
| No Support for EMS (Client) | 1,254 | 4.23 | 27,388 | 1.93 |
| EMS negotiated | 25,696 | 86.60 | 625,393 | 44.00 |
| **Supported Elliptic Curves/DH Groups** | | | | |
| secp256r1 | 29,669 | 99.99 | 1,419,588 | 99.88 |
| secp384r1 | 29,669 | 99.99 | 1,419,587 | 99.88 |
| secp521r1 | 14,774 | 49.79 | 864,480 | 60.82 |
| x25519 | 14,158 | 47.71 | 561,174 | 39.48 |
| **Negotiated Elliptic Curves** | | | | |
| secp256r1 | 16,550 | 55.77 | 1,289,585 | 90.73 |
| x25519 | 11,880 | 40.04 | 43,318 | 3.05 |
| secp384r1 | 980 | 3.30 | 38,013 | 2.67 |
| secp521r1 | 153 | 0.52 | 821 | 0.06 |
| None (no ECDHE) | 110 | 0.37 | 49,609 | 3.49 |
| **Negotiated SignatureScheme Algorithms** | | | | |
| rsa_pkcs1_sha256 | 25,071 | 84.49 | 898,758 | 63.23 |
| rsa_pkcs1_sha512 | 1,922 | 6.48 | 211,004 | 14.85 |
| rsa_pkcs1_sha384 | 863 | 2.91 | 36 | <0.01 |
| rsa_pss_rsae_sha256 | 846 | 2.85 | 129,677 | 9.12 |
| rsa_pkcs1_sha1 | 5 | 0.02 | 2,486 | 0.17 |
| <TLSv1.2, SHA-1/MD5 | 852 | 2.87 | 128,996 | 9.08 |
| None (no ECDHE) | 110 | 0.37 | 49,609 | 3.49 |
| Other | 4 | 0.01 | 780 | 0.05 |
| **Support for OCSP/Status Request** | | | | |
| Client support for OCSP | 13,727 | 46.26 | 861,195 | 60.59 |
| OCSP response from server | 362 | 1.22 | 53,843 | 3.79 |
| **Heartbeat support** | | | | |
| Client support for Heartbeat | 46 | 0.16 | 8,814 | 0.62 |
| ↪Thereof Heartbeat negotiated | 1 | ↪2.17 | 5,527 | ↪62.71 |
| **Additional observations** | | | | |
| Support for ENCRYPT_THEN_MAC | 409 | 1.34 | 10,047 | 0.71 |
| Server Random included DOWNGRD1 | 34 | 0.11 | 25,013 | 1.76 |
| Server Random included DOWNGRD0 | 58 | 0.20 | 8,018 | 0.56 |

## D.2 Per-Realm data

| Data Item | Uni | % | DFN | % |
|---|---|---|---|---|
| Total realms | 767 | | 5,531 | |
| TLSv1.1- negotiated while TLSv1.2+ clients present | 65 | 8.47 | 919 | 16.62 |
| No support for Extended Master Secret | 288 | 37.55 | 2,915 | 52.70 |
| OCSP Stapling active | 32 | 4.17 | 402 | 7.27 |