# eduroam IdP

eduroam IdP

## Obligations for an eduroam IdP

## Selecting EAP types

The decision which EAP type(s) to deploy on your eduroam IdP depends on several factors:

- Capabilities of your Identity management backend
- Types of devices you want to support

### Choices depending on the Identity Management System

Regarding the identity management backend, the most fundamental differentiation between EAP types is the type of credential they support.

- Does your identity management backend support X.509 Client Certificates? Then you can use EAP-TLS.
- Does your identity management backend use username/password combinations?
    - Does it store the passwords as either clear text - or - encrypted as NT-Hash? Then you can use EAP-TTLS, PEAP, EAP-FAST, EAP-PWD and more.
    - Does it store the passwords in a different crypt format? Then you can use EAP-TTLS only.

As you see, the decision is largely dependent on your identity management system; so your choices may be limited. As a more concrete advice for some IdM backends:

- Microsoft ActiveDirectory: stores passwords as NT-Hashes.

### Anonymous outer identities

Almost all EAP types support the use of anonymous outer identities. The primary use of anonymous outer identities is for better preservation of privacy for your users; a properly configured supplicant will then not even reveal the real username of the user to the visited eduroam SP; instead, the username is replaced with a dummy value.

This feature needs protocol support by the EAP type in question; the basic idea is that there have to be two stages of communicating the client identity:

- one identity, the outer identity, is used to route the user's login request from the eduroam SP via the eduroam RADIUS path to the eduroam IdP
- the second, "inner" identity, is only revealed inside a cryptographically protected tunnel to the IdP

Since the outer identity is only needed for routing purposes towards the IdP, the local username part does not have to be accurate and can be obfuscated. The IETF-suggested way of obfuscating the username is to leave it empty; but it can just as well be replaced with "anonymous", "anon" or similar. However, the realm part (i.e. behind the @ sign) always needs to be accurate because it contains the routing information.

The inner identity always needs to be fully accurate, because it is used to authenticate the user. It does not necessarily have to contain an @ sign at all, because that username is local to the IdP and is only seen and evaluated there.

Example:

- Outer identity: anonymous@restena.lu
- Inner identity: stefan.winter

For eduroam request routing, the part @restena.lu of the outer identity is used to route the request to the restena.lu realm and to establish a secure tunnel; while the real username inside this tunnel which is looked up in a user database is "stefan.winter".

Here is a break-down of anonymous outer identity support for some popular EAP types:

| EAP-Type | Support for anonymous outer identites |
|---|---|
| EAP-TTLS | yes |
| PEAP | yes |
| EAP-FAST | yes |
| EAP-TLS | support in protocol, but not typically available in supplicants |
| EAP-PWD | no |

If the EAP type allows for the use of outer identities, it is a client device configuration option to either make use of them or not; there is little you as an IdP can do to force the use of anonymous outer identities (except for providing and encouraging the use of pre-configured installers which will then make all the necessary settings on the client device automatically).

### Choices depending on the envisaged devices

The landscape of wireless-enabled devices is rather heterogenous, and support for EAP types varies. Ideally, you should survey which types of devices you should come to expect among your user base, check the capabilities of these devices, and make an informed decision regarding the EAP type of choice.

However, the EAP protocol is flexible enough to handle multiple EAP types: if your IdM backend can support the use of multiple EAP types, then you can configure all the supported EAP types. In that case, you have to select a "default" EAP type - it should be set to the EAP type with the broadest support in your client base.

Now, assuming you have the option of configuring a range of EAP types *and* your clients support that same range, which of these types should you prefer?

- We suggest the use of PEAP over EAP-TTLS for it does a mild amount of protection of the user password inside the secure tunnel.
- If you cannot support PEAP, consider to allow TTLS-PAP **and** the more unusual variant TTLS-GTC (initially Generic Token Card; also used for passwords which are not savable on the client device). Some older devices (certain Symbian OS builds) support TTLS, but not PAP inside. Enabling TTLS-GTC will allow these devices to connect.

## EAP Server certificate considerations

Almost all EAP types in eduroam (with the exception of EAP-PWD) require an X.509 server certificate with which the RADIUS server identifies itself to the end user before the user sends his credentials to the server.

### Consideration 1: Procuring vs. creating your own server certificate

In a generic web server context, server certificates are usually required to be procured by a commercial Certification Authority (CA) operator; self-made certificates trigger an "Untrusted Certificate" warning. It makes sense for browsers to have a pre-configured trust store with many well-known CAs because the user may browse to any website; and the operator of that website may have chosen any of those well-known CAs for his website. In an abstract notion, one can say: it is required to have many CAs in the list because the user device does not have all required information for certificate validation contained in its own setup; it misses the information "which CA did the server I am browsing to use to certify the genuinity of his website?".

These considerations are not at all true in an EAP authentication context, such as an eduroam login. Here, the end user device is pre-provisioned with the entire set of information it needs to verify this specific TLS connection: the IdP has a certificate from exactly one CA, and needs to communicate both that CA and the name of his authentication server to the end user. A trust store list from the web browser is thus insignificant in this context; certificates from a commercial CA are as valid for EAP authentications as are self-made certificates or certificates from a small, special-purpose CA. For a commercial CA, the installation of the actual CA file may be superfluous in some client operating systems (particularly those who make their "web browser" trust store also accessible for EAP purposes), but marking that particular CA as trusted for this specific EAP authentication setup still needs to be done.

Note that also root CA certificates have an expiry date. Both for commercial and private CAs please be aware that an exchange of the root CA certificate will require re-configuration of all your end-users' devices to accept the new CA. As a consequence: for commercial CAs, check their root CA's expiry date so you can make an informed decision whether you want to buy the certificate from them or not. For your own private-use CA: choose a very long expiry date for the CA. Especially for commercial CAs, keep in mind that if you ever want to switch to a *different* CA as a trust anchor, all your end-user devices again need to be re-configured for that new root.

Configuration tools like eduroam CAT enable to provision the chosen CA(s) and the expected server name(s) into client devices without user interaction. In that light, it does not make much difference whether to procure a server certificate from a commercial CA or to make your own; either way, configuration steps are necessary on the end-user device to enable and secure your chosen setup. With the conceptual differences being small, a number of secondary factors come into play when making the decision where to get a server certificate from:

- Do you have the necessary expertise to create a self-signed certificate; or to set up a private Certification Authority and issue a server certificate with it? Consider in particular the next "Consideration 2" which imposes some properties onto the certificates you need.
- Does your eduroam NRO operate a special-purpose CA for eduroam purposes, so that you could get a professionally crafted certificate without much hassle?
- Do your end-user devices all verify the exact server identity (issuing CA certificate AND expected server name)?

The third question is particularly important these days because some popular operating systems, particularly early Android versions up until Android 7, do not allow to configure verification of the expected server name in their UI. For such operating systems, using a commercial CA for the server certificate opens up a loophole for fraud: anyone with a valid certificate from this CA, regardless of the name in the certificate, can pretend to be the eduroam authentication server for your end-user; which ultimately means the end-user device will send the user's login credentials to that unauthorised third-party. If you use a self-signed certificate or private CA however, which issues only one/very few certificates, and over which you have full control, then no unauthorised third party will be able to get a certificate in the first place, and thus can't fraud your users.

Another factor to consider when making the decision private vs. commercial CA is that of size and length of the EAP conversation during every login: with a private CA, you will be able to construct a certificate chain without intermediary CA certificates; requiring less bytes to be transmitted inside the EAP conversation (see Consideration 3, below). This results in fewer EAP round-trips and thus a faster authentication.

So, as a general recommendation: if you have the required expertise, it is suggested to set up a private CA exclusively for your IdP's eduroam service. This CA should have a very long lifetime to prevent certificate rollover problems. The CA should issue only server certificates for your eduroam IdP server (s). If you do not have that expertise, you should make use of your NROs special-purpose CA if it exists. If none of these work for you, a certificate from a commercial CA is the third option.

> ⚠️ *With great power comes great responsibility. (Voltaire)*

If you choose to use a private CA and deploy it to your users' devices, there may be side-effects after the installation of the CA. Some devices do not differentiate between a CA which is used for Wi-Fi server authentication purposes and, say, web browser TLS encryption.

Your CA may incidentally yield the power on such client devices of your own user base to inspect their web or other traffic (if you actively abuse it and modify your IT infrastructure to enable this). We do not endorse or encourage this in any way.

Having your own trusted root CA in client devices also makes the protection of the private key to this CA an objective of paramount importance.

We recommend that you inform your users how best to restrict the power of the CA (e.g. with CA installation instructions which point to a dedicate Wi-Fi store [Android 4.3+]; or with the advice not to use browsers which use the built-in CA store of the device [MS Windows]).

### Consideration 2: Recommended certificate properties

Various end-user device operating systems impose different requirements on the contents of the server certificate that is being presented. Luckily, these requirements are not mutually exclusive. When creating or procuring a server certificate, you should check with the CA that its certificates satisfy as many of these requirements as possible to ensure broad compatibility with your users' devices. The list below does not include "standard" sanity checks applied to certificates; e.g. well-formedness of the data, validity timestamps etc. These checks are done "as per usual" in every TLS connection.

The most important property of the server certificate is the name of the server. Since this certificate is not for a webserver, there is no necessity to put an actual hostname into the server name. Also, when an Identity Provider uses multiple servers for resilience reasons, then all these servers can and should have a certificate with the same name; and it may well be the identical certificate. Having different names for different servers means that end-user devices must be configured to trust multiple servers, which is more cumbersome than just having to configure one name string.

Some end-user device operating systems might (incorrectly) require the name to be parseable as a hostname; so it is a good idea to use a server name which parses as a *fully-qualified domain name* - the corresponding record does not have to exist in DNS though. The server name should then be both in certificate's Subject field (*Common Name* component) and be a *subjectAltName:DNS* as well.

The following additional certificate properties are non-standard and are of particular interest in the eduroam context:

| Property | Content | Remarks |
|---|---|---|
| X.509 version | 3 | The CA certificate should be an X.509v3 certificate. |
| server name | parses as fully-qualified domain name | Server certificates with spaces, e.g. "RADIUS Service of Foo University" are known to be problematic with some supplicants, one example being **Apple iOS 6.x**. |
| server name | Subject /CN == SubjectAltName: DNS | Some supplicants only consult the CN when checking the name of an incoming server certificate (**Windows 8 with PEAP**); some check either of the two; some new EAP types such as **TEAP**, and **Linux clients configured by CAT 1.1.2+** will only check SubjectAltName:DNS. Keeping the desired name in both fields in sync is a safe bet for futureproofness.<br><br>Only use one CN. If you have multiple RADIUS servers, either use the same certificate for all of them (there is no need for the name to match the DNS name of the machine it is running on), or generate multiple certificates, each with one CN /subjectAltName:DNS pair. |
| server name | not a wildcard name (e.g "*. someidp. tld") | Some supplicants exhibit undefined/buggy behaviour when attempting to parse incoming certificates with a wildcard. **Windows 8 and 8.1** are known to choke on wildcard certificates. |
| signature algorithm | Minimum: SHA-1<br><br>Recommended: SHA-256 or higher | Server certificates signed with the signature algorithm MD5 are considered invalid by many modern operating systems, e.g. ***Apple iOS 6.x and above***. Also **Windows 8.1** and all previous versions of **Windows (8, 7, Vista)** which are on current patch levels will not validate such certificates. Having a server certificate (or an intermediate CA certificate) with MD5 signature will create problems on these operating systems.<br><br>Apparently, no operating system as of yet has an issue with the root CA being self-signed with MD5. This may change at any point in the future though, so when creating a new CA infrastructure, be sure not to use MD5 as signature algorithm anywhere.<br><br>The continued use of SHA-1 as a signature algorithm is not recommended, because several operating systems and browser vendors already have a deprecation policy for SHA-1 support. While the deprecation in browser-based scenarios does not have an immediate impact on EAP server usage, it is possible that system libraries and operating system APIs will over time penalise the use of SHA-1. Therefore, for new certificates, SHA-256 is recommended to avoid problems with the certificate in the future. |
| length of public key | Minimum: 2048 Bit<br><br>Recommended: 3072 Bit or higher | Server certificates with a length of the public key below 1024 bit are considered invalid by some recent operating systems, e.g. ***Windows 7 and above***. Having a server certificate (or an intermediate CA certificate) with a too small public key will create problems on these operating systems.<br><br>The continued use of 1024 bit length keys is not recommended, because several operating systems and browser vendors already have a deprecation policy for this key length. While the deprecation in browser-based scenarios does not have an immediate impact on EAP server usage, it is possible that system libraries and operating system APIs will over time penalise the use of short key lengths. 2048 bit is the most popular and default choice these days. However, some applications already suggest 3072 bit or more, and a longer key length does not have an extra cost. So, it is recommended to create new certificates with 3072 bit keys or higher (4096 has been tested and is also unproblematic) to avoid problems with the certificate in the future. |

| | | |
|---|---|---|
| Extension: Extended Key Usage | TLS Web Server Authentication | Even though the certificate is used for EAP purposes, some popular operating systems (i.e. ***Windows XP and above***) require the certificate extension "TLS Web Server Authentication" (OID: 1.3.6.1.5.5.7.3.1) to be present. Having a server certificate without this extension will create problems on these operating systems. |
| Extension: CRL Distribution Point | HTTP /HTTPS URI pointing to a valid CRL | Few very recent operating systems require this extension to be present; otherwise, the certificate is considered invalid. Currently, ***Windows Phone 8*** is known to require this extension; ***Windows 8*** can be configured to require it.<br><br>These operating systems appear to only require the extension to be present; they don't actually seem to download the CRL from that URL and check the certificate against it. One might be tempted to fill the certificate extension with a random garbage (or intranet-only) URL which does not actually yield a CRL; however this would make the certificate invalid for all operating systems which do evaluate the extension if present. So the URL should be a valid one. |
| Extension: BasicConstraint (critical) | CA: FALSE | Server certificates need to be marked as not being a CA. Omitting the BasicConstraint:CA totally is known to cause certificate validation to fail with **Mac OS X 10.8 (Mountain Lion)**; setting it to TRUE is a security issue in itself. Always set the BasicConstraint "CA" to false, and mark the extension as critical. |
| Certificate Type | Domain-Validated (DV) or Organisation-Validated (OV) | There have been several reports that **ChromeOS** will refuse to accept Extended Validation (EV) certificates. You should avoid these types of certificates if you care about this operating system. |
| Validity Time | 825 days or fewer | **Apple** products as of **macOS 10.15+ and iOS 13+** enforce this limit and consider certificates with a longer lifetime as untrusted. See also<br><br>this Apple article. |

### Consideration 3: Which certificates to send in the EAP exchange

End-user devices need to verify the server certificate. They do this by having a known set of trustworthy anchors, the "Trusted Root Certificates". These root certificates need to be available and activated on the device prior to starting the eduroam login. Therefore, it does not serve any useful purpose to send the root CA certificate itself inside the RADIUS/EAP conversation. It is not harmful to send it anyway though, except that it unnecessarily inflates the data exchange, which means more round-trips during eduroam authentication, and in turn a slower login experience. One possible exception is: there are reports of certain Blackberry devices for which it is advantqageous to send the root CA certificate nontheless; if you expect you need/want to support Blackberry devices, sending the root CA may be of help.

During the EAP conversation, the eduroam IdP RADIUS server always needs to send its server certificate.

One question needs an administrative decision: if there is one or more intermediate CAs between the root CA and the server certificate (such as is the case with, for example, the TERENA Certificate Service (TCS) and many commercial CAs), should the intermediate CA certificates be sent to the end user device during the EAP conversation, or should the devices pre-install the intermediate CAs along with the root certificate?

In any case, for a successful verification of the server certificate, the end-user's device must have the full set of CA certificates available. It does not matter whether the intermediate CAs have been pre-provisioned or are sent during the login phase; but if any one intermediate CA is missing, the verification of the server certificate will fail.

Pre-provisioning the intermediate CAs has the advantage of a relatively small amount of data being sent during the EAP authentication, which means fewer round-trips between the end-user's device and the eduroam IdP RADIUS server. The downsides of this approach are that any changes to intermediate CAs (re-issue, rollover) will also need to be pushed to end-user devices. Also, if end-user devices are not under administrative control of the IdP, there is a danger that some end users do not follow the advice to install all intermediate CAs even though they should, and end up in a situation where the server certificate can not be validated.

Sending the intermediate CAs during the login phase means a longer time to authenticate due to more round-trips, but means that it is sufficient for client devices to install the root CA certificate; if intermediate CAs change, the new ones will always become available to the device during the next authentication data exchange.

For most deployments, it probably makes more sense to include the intermediate CA certificates during the RADIUS/EAP conversation.

## Set up of several popular RADIUS servers

### FreeRADIUS

#### Setting up FreeRADIUS

This section describes how to set up FreeRADIUS for an IdP. It assumes that you have already executed the configuration steps for the eduroam SP configuration of FreeRADIUS. We will expand that configuration to turn FreeRADIUS into a simple IdP. N.B.: even if you are going to have an IdP-only installation, the eduroam SP configuration for FreeRADIUS is still the exact same. You just don't define any own Access Point clients in clients.conf.

Adding IdP support in FreeRADIUS needs several steps to be executed:

- a TLS server certificate needs to be created for EAP methods to work

- the desired EAP types need to be configured.
- the virtual server eduroam needs to be instructed to do tunneled EAP authentication
- a user database needs to be linked to the FreeRADIUS instance to authenticate the users
- a realm needs to be marked as to-be-authenticated-locally in the configuration
- the server needs to be prepared to process incoming requests *from* the upstream FLR server

These steps are explained in detail below. For the user database, this example will use a "flat file" with usernames and passwords. The Goodies section contains examples for MySQL and other types of backend databases.

### TLS server certificate

While it is possible to buy and install a commercial TLS certificate, this is neither necessary (the trust settings of web-browser stores don't apply for EAP, so there are no "recognised" CAs) nor prudent (a commercial CA issues many certificates, and uncautious users might be tempted to accept other certificates from that same CA).

We suggest to create an own certificate. FreeRADIUS makes this very easy by providing an automatic script for that purpose. Execute the

```
/etc/raddb/certs/bootstrap
```

script. It will generate certificates which are suited for EAP authentication, and name them so that the server can find them immediately without further configuration. Later, for the supplicant configuration, you will need to include the generated CA certificate into your supplicant configurations.

### EAP type configuration

The file /etc/raddb/eap.conf defines how EAP authentication is to be executed. The shipped configuration file is not adequate for eduroam use; it enabled EAP-MD5 and LEAP, for example; which are not suitable as eduroam EAP types. Use the following content for eap.conf instead. It enables PEAP and TTLS:

```
eap {
                default_eap_type = peap
                timer_expire     = 60
                ignore_unknown_eap_types = no
                cisco_accounting_username_bug = no

                tls {
                        certdir = ${confdir}/certs
                        cadir = ${confdir}/certs
                        private_key_password = whatever
                        private_key_file = ${certdir}/server.key
                        certificate_file = ${certdir}/server.pem
                        CA_file = ${cadir}/ca.pem
                        dh_file = ${certdir}/dh
                        random_file = /dev/urandom
                        fragment_size = 1024
                        include_length = yes
                        check_crl = no
                        cipher_list = "DEFAULT"
                }

                ttls {
                        default_eap_type = mschapv2
                        copy_request_to_tunnel = yes
                        use_tunneled_reply = yes
                        virtual_server = "eduroam-inner-tunnel"
                }

                peap {
                        default_eap_type = mschapv2
                        copy_request_to_tunnel = yes
                        use_tunneled_reply = yes
                        virtual_server = "eduroam-inner-tunnel"
                }

                mschapv2 {
                }

        }
```

A common question regarding this definition is: "why is TLS also configured? I don't want it, can I disable it?" The answer is: the TTLS and PEAP sections depend on the tls stanza for the definition of which server certificates to use. You cannot delete the stanza, but that doesn't mean you can't effectively disable TLS: the tls stanza contains the ca_file parameter. Only clients with a TLS client certificate from this CA will be accepted. We have just created a brand-new CA with the "bootstrap" script. Simply don't issue nor distribute any client certificates from this CA, then nobody will be able to log in with EAP-TLS. Note that in newer versions of FreeRADIUS (>3.0.14) there is a new tls-config section that allows you to configure the common TLS configuration without configuring the TLS EAP type. The config above is backwards compatable, but if you want to take advantage of the new section you can replace the name of the "tls" block above with "tls-config tls-common" and then reference it from each EAP type with "tls = tls-common" (the example eap config shows you how to do this).

Another question is regarding the mschapv2 section. For all practical purposes, the easy answer is that it is a piece of magic and needs to be there for PEAP to work. If you are curious regarding the gory details, please let us know.

Note that one parameter for both the ttls and peap stanza is "virtual_server = eduroam-inner-tunnel". This means that the inner EAP authentication will be carried out in this other virtual server, which we will define later.

### Virtual server eduroam: enable EAP, make Operator-Name conditional

Compared to the eduroam SP config, you need to additionally mention the "eap" module in both the authorize and authenticate stanza of the file /etc/raddb/sites-enabled/eduroam so that your server can process EAP requests from your own userbase.

You should also make sure to only tag those incoming requests with the Operator-Name attribute which actually originate from your own WiFi gear - as an IdP, your own users roaming elsewhere will also be processed, but they should not carry your own Operator-Name. For the purposes of this wiki, let's assume that you are connected to one FLR server, and it is defined in your clients.conf with the shortname "antarctica-flr-1" (see below for the exact definition).

It will then look like the following:

```
authorize {
                if ("%{client:shortname}" != "antarctica-flr-1") {
                   update request {
                           Operator-Name := "1yourdomain.tld"
                            # the literal number "1" above is an important prefix! Do not change it!
                   }
                }
                auth_log
                suffix
                eap
        }

authenticate {
       eap
}
```

### Virtual server eduroam-inner-tunnel

When the eap module has started with an authentication, it will first establish a TLS tunnel; this is done by enabling the module in the previous "eduroam" virtual server. After the TLS tunnel is established, the content (i.e. the tunneled authentication) is processed separately in this new virtual server. Create the file in /etc/raddb/sites-enabled/eduroam-inner-tunnel and give it the following content:

```
server eduroam-inner-tunnel {

authorize {
        auth_log
        eap
        files
        mschap
        pap
}

authenticate {
        Auth-Type PAP {
                pap
        }
        Auth-Type MS-CHAP {
                mschap
        }
        eap
}

post-auth {
        reply_log
        Post-Auth-Type REJECT {
                reply_log
        }

}
}
```

Let's revisit the modules which this virtual server executes one after another:

- auth_log: logs the incoming packet to the file system. This is needed to fulfill the eduroam SP logging requirements. Note that this log *may* contain the user's cleartext password if TTLS-PAP is used. You can log the packet with omitted User-Password attribute if you prefer; see the "Goodies" section for more details).
- eap: if the EAP authentication contains another EAP instance inside, the module will decode it. This is the case for PEAP.
- files: this module tries to find out the authoritative password for the user by looking up the username in the file
- mschap: this module is in effect only if PEAP-MSCHAPv2 or TTLS-MSCHAPv2 is used. It will mark the packet as to be authenticated with MS-CHAP algorithms later.
- pap: this module is in effect only if TTLS-PAP is used. It will mark the packet as to be authenticated with PAP alogrithms later.
- reply_log: logs the reply packet to the file system

### User database: flat file

By default, the "files" module will use information in the file

```
/etc/raddb/users
```

for authenticating users. This file has a straightforward format

```
icecold@group1.aq       Cleartext-Password := "snowwhite"

otheruser@group1.aq     Cleartext-Password := "swordfish"
```

### Local authentication for your realm

In the SP configuration, all requests were unconditionally forwarded to upstream. We will need to revisit the file "proxy.conf" and mark one realm to NOT proxy. In this example, we will use "@group1.aq" as the local authentication realm. Simply add the following stanza immediately preceeding the "DEFAULT" realm:

```
realm group1.aq {
        nostrip
}
```

Since the stanza doesn't contain a server pool to proxy to, this realm won't be proxied and instead authenticated locally. This stanza works only for users who correctly use the full username format "user123@group1.aq" for their eduroam login.

If the IdP and SP are colocated, it is possible to *locally* also accept users who erronuously omitted their realm (just "user123"). This is NOT permitted by the eduroam policy (read 6.3.2 bullet 6 under AAA Servers of the current service definition document: "The outer EAP identities (and with it, RADIUS User-Name attributes) for the IdP MUST be in the format of arbitrary@realm"). Allowing this also requires further configuration and it is strongly discouraged, because it will give such users a "halfways-working" experience: they will be able to use eduroam when on their own IdP's campus, because no routing information needs to be evaluated, but their account will fail at all other locations. Therefore, this guide does not include instructions for that kind of setup.

### Processing incoming requests

As an eduroam IdP, your users can go to other eduroam hotspots around the globe. They will still be authenticated at your server. In these roaming cases, your upstream FLR servers will send Access-Requests to your server. Surprisingly, it is very simple to configure that: these upstream servers are simply clients - just like an Access Point. So, simply add client stanzas for your FLR servers into clients.conf:

```
 client antarctica-flr-1 {
        ipaddr                          = 172.20.1.2
        netmask                         = 32
        secret                          = secretstuff
        require_message_authenticator   = yes
        shortname                       = antarctica-flr-1
        nastype                         = other
        virtual_server                  = eduroam
}
```

### CUI for eduroam IdP

To use the Chargeable-User-Identity (CUI) you must already use the Operator-Name attribute. This documentation is only for FreeRADIUS 3.0.X release.

**Modify the log module**

Edit "eduroam_cui_log" file in the mods-available/ subdirectory and add the following lines to your virtual inner server :

```
...
linelog cui_inner_log {
#    filename = syslog
    filename = ${logdir}/radius.log
    format = ""
    reference = "inner_auth_log.%{%{reply:Packet-Type}:-format}"
    inner_auth_log {
        Access-Accept = "%t : eduroam-inner-auth#VISINST=%{request:Operator-Name}#USER=%{User-Name}#CSI=%{%
{Calling-Station-Id}:-Unknown Caller Id}#NAS=%{%{Called-Station-Id}:-Unknown Access Point}#CUI=%{%{%{reply:
Chargeable-User-Identity}:-%{outer.reply:Chargeable-User-Identity}}:-Local User}#RESULT=OK#"
        Access-Reject = "%t : eduroam-inner-auth#VISINST=%{request:Operator-Name}#USER=%{User-Name}#CSI=%{%
{Calling-Station-Id}:-Unknown Caller Id}#NAS=%{%{Called-Station-Id}:-Unknown Access Point}#CUI=%{%{%{reply:
Chargeable-User-Identity}:-%{outer.reply:Chargeable-User-Identity}}:-Local User}#RESULT=FAIL#"
    }
}
```

**Use policy and module in your eduroam-inner-tunnel virtual server**

Add 'cui-inner' (policy already defined, you don't need to change it) and 'cui_inner_log' in post-auth section :

```
server eduroam-inner-tunnel {
...
               post-auth {
                               reply_log
                               cui_inner_log
                               cui-inner
                               Post-Auth-Type REJECT {
                                       reply_log
                                       cui_inner_log
                               }
               }
...
}
```

That's it! Now your server is prepared for eduroam IdP operation! You can add users to your "database" by amending the "users" file; if you do, you will unfortunately have to restart FreeRADIUS so that it picks up the change.

## Goodies

**Omitting User-Password in inner authentication logs**

By default, the "detail" modules log every attribute as it was received. For inner authentications with TTLS-PAP, this means that the attribute "User-Password" with the user's perceived password will be logged. This is often considered harmful. You can deactivate it by blacklisting the attribute in the auth_log module in /etc/raddb/modules/auth_log:

```
detail auth_detail {
  ...
  suppress {
    User-Password
  }}
```

**adding VLAN assignment attributes**

You can mark every user with a VLAN where he should be put into. This is done by assigning "reply items" to the user in the authentication database. In our flat file example, reply attributes are in a separate line, indented by a tab. To put our two example users into VLANs 17 and 42, respectively, the entries would look like the following:

```
icecold@group1.aq       Cleartext-Password := "snowwhite"
                        Tunnel-Type             := VLAN,
                        Tunnel-Medium-Type      := IEEE-802,
                        Tunnel-Private-Group-ID := 17

otheruser@group1.aq     Cleartext-Password := "swordfish"
                        Tunnel-Type             := VLAN,
                        Tunnel-Medium-Type      := IEEE-802,
                        Tunnel-Private-Group-ID := 42
```

**Using SQL as user database backend**

Using a flat file as in our example scales very poorly. You can use arbitrary database backends instead; the FreeRADIUS documentation can give you an overview. If you wish to use SQL, changing our example configuration is very easy: simply replace the "files" line in eduroam-inner-tunnel:authorize with "sql". You'll need to specify the connection details for your SQL backend in the corresponding module ( /etc/raddb/modules/sql ).

The schema which FreeRADIUS uses to store user information is similarly structured to the "users" file: a table radcheck holds the check items (i.e. the username and password), and the radreply table contains the reply items (for example VLAN memberships, as explained above).

**Mandating or forbidding use of anonymous outer identity**

eduroam at large supports anonymous outer identities for user logins. It is at the discretion of eduroam IdPs whether they want to

- mandate that their users use an anonymous outer identity
- forbid their users to  use an anonymous outer identity
- be agnostic in that respect

Configuring any one of the three choices is done with only a few lines of configuration. The easiest choice is being agnostic: no configuration is necessary, since there is no link between the inner and outer User-Name attribute in FreeRADIUS.

If you want to mandate the use of anonymous outer identities, the recommended way is using the identity "@realm" (i.e. the part left of the @ sign should be empty). You can enforce that only this outer User-Name is allowed to proceed to EAP authentication by adding the following to the authenticate section:

```
if ( User-Name != "@realm" ) {
      fail
}
```

If you want to forbid usage of anonymous outer identities, you can do this by comparing the two presented User-Name attributes of the outer and inner authentication. You can only do this in the eduroam-inner-tunnel virtual server obviously, since only that server has access to the inner identity. Put the following into the "authenticate" section of eduroam-inner-tunnel:

```
if ( User-Name != outer.User-Name ) {
     fail
}
```

### More information

Eduroam-in-a-box web configuration tool:http://eduroam.sourceforge.net

## Radiator

One popular RADIUS server is Open Systems Consultant's "Radiator. This section details its configuration.

Because of the EAP authentication within RADIUS, a (small) PKI is required. If there is no PKI available, you could create the required key and certificate with, for instance, TinyCA. TinyCA (http://tinyca.sm-zone.net/) is a simple graphical interface on top of OpenSSL. It is possible to use OpenSSL directly (but instructions to do so are outside the scope of this document).

There is also a bootable CD available based on Knoppix that runs TinyCA, the roCA (read-only CA) that can be found at http://www.intrusion-lab.net/roca/.

Depending on the EAP-type used, client certificates may also be needed.

Within the Radiator distribution there are also simple scripts available to create certificates for testing purposes.

The Radiator RADIUS server needs the configuration file /etc/radiator/radius.cfg.

This configuration file can be created with the editor of choice, for example

*vi /etc/radiator/radius.cfg*
or
*pico /etc/radiator/radius.cfg*

In the following examples there are two kinds of EAP that are configured at "institution":

- EAP-TLS based on client-certificates.
- EAP-TTLS and EAP-PEAP that do not require client certificates but use the traditional mechanism of username/password authentication instead.

### Clients

RADIUS is based on a client-server model. The NAS-devices (Access Points, switches etc.) forward credentials to a RADIUS server, i.e. act as a client, and therefore need to be defined on the RADIUS server. Other RADIUS servers can act as a client as well, so every kind of RADIUS-request can be forwarded to another server.

The clients are configured within Radiator using the <Client>-clause:

```
<Client 192.168.10.200>
            Secret 6.6obaFkm&RNs666
            Identifier ACCESSPOINT1
            IdenticalClients 192.168.10.201
                        RequireMessageAuthenticator
</Client>
```

In this example there is a client definition for 192.168.10.200, an Access-Point. The "secret" is a series of (at best 16) characters that are used to encrypt the credentials sent in the RADIUS-request.

It is of course recommended to create a secret that cannot be guessed easily, otherwise the RADIUS-message can be decrypted. This is not an issue with EAP-authentication using 802.1X, since the credentials are also transmitted over a SSL-encrypted tunnel between the client and the final authentication server. However, with regular credentials (like those used with Web-based redirection authentication) this is sensitive information that might be captured, therefore a reasonably complex secret and an SSL tunnel is recommended.

The Identifier in the Client-definition can be used later on in the Radiator configuration to filter a specific request.

If more then one Client is to use this same secret and identifier definition, the IdenticalClients statement can be used. If there are many clients with different IP-addresses, there is also the possibility for a "catch-all" client. This is the default client that is used after all other client definitions didn't match. Define this client as:

```
<Client DEFAULT>
```

If this kind of configuration is used, it is worth filtering with firewall-rules on RADIUS packets. There are only a few places where a RADIUS-request should come from; the management VLAN with the NAS-devices (switches and access-points), and the RADIUS infrastructure where unknown requests can be sent to.

### Realms and VLAN assignment

The processing of authentication and accounting requests is done by linear processing of the present <Realm>- or <Handler>-clauses in the Radiator configuration file. Handler-clauses are more potent than Realm clauses in terms of filtering anything besides realms, and are therefore the preferred method. A realm is the part behind a username to indicate the origin of a user. With RADIUS, the username is usually separated from the realm with a "@" so the complete username looks like a regular e-mail address.

A <Handler>-clause is terminated with a </Handler>.

Within a Handler many mechanisms can be configured that define what to do with the RADIUS request.

## PROXY example

The simplest Handler for proxying the request to another server uses the "AuthBy RADIUS" definition within this Handler.

In this example a proxy-configuration is shown. First we have a Handler that matches on any request, as long as it does not come from the client with the identifier "Proxy-Identifier". This is to prevent a proxy loop. When a request comes from a proxy-server, it should never be forwarded back to that proxy-server.

Another important part is the hostname to which the request should be forwarded. Multiple hostnames can be defined here for redundancy reasons: if the first host does not respond within three seconds, the second one is tried instead. The UDP ports to which the RADIUS-request should be forwarded can be defined in this "AuthBy RADIUS" clause as well.

```
<Handler Client-Identifier=/^(?!Proxy-Identifier$)/>
         <AuthBy RADIUS>
                   Host            192.87.36.3
                   Secret         super_secret!
                   AuthPort     1812
                   AcctPort     1813
                   StripFromReply Tunnel-Type, Tunnel-Medium-Type, Tunnel-Private-Group-ID
                   AddToReply Tunnel-Type=1:VLAN, Tunnel-Medium-Type=1:Ether_802, Tunnel-Private-Group-ID=1:909
          </AuthBy>
</Handler>
```

For a "Host", both the IP-address and FQDN can be used. The choice is more or less a personal preference of the RADIUS administrator, but be aware that the hostnames are only looked up once at the Radiator (re)start. If the lookup fails, the Host cannot be used until the next restart. This can represent a problem at a power outage, where for instance the DNS server is not yet available even though Radiator is.

While by using hostnames one benefits from the administrative ease when an IP-address is changed, it is still necessary to restart the RADIUS server.

The last part in this <AuthBy RADIUS>-definition shows the addition of RADIUS-attributes to the RADIUS- response. These attributes can be used to define a VLAN that will be assigned to users that are authenticated using this Handler. With StripFromReply, the attributes that came from the proxy-server are stripped first to prevent malicious VLAN-assignments, afterwards the attributes are added with the proper values for the local network design. In this case, VLAN 909 is used for guests.

## Secure authentication with EAP-TLS

EAP-TLS requires both server and client certificates. Rolling out such certificates is a sometimes daunting administrative process, and is out of the scope of this document. The remainder of this section assumes that client certificates have been issued to the users already.

In this example the AuthBy-definition is outside the Handler, and is referred to using the Identifier. (This is useful if the AuthBy-definition is reused in another Handler, for instance.)

```
<AuthBy  FILE>
             Identifier ID4-TLS
             Filename %D/TLS-users
             EAPType TLS
             EAPTLS_CAFile %D/cert/institution-ca-chain.pem
             EAPTLS_CertificateFile %D/cert/radius-server-cert.pem
             EAPTLS_CertificateType PEM
             EAPTLS_PrivateKeyFile %D/cert/radius-server-key.pem
             EAPTLS_PrivateKeyPassword (the secret that secures the private-key)
             EAPTLS_MaxFragmentSize 1024
             AutoMPPEKeys
             SSLeayTrace 1
             StripFromReply Tunnel-Type,Tunnel-Medium-Type,Tunnel-Private-Group-ID
             AddToReply Tunnel-Type=1:VLAN,Tunnel-Medium-Type=1:Ether_802,Tunnel- Private-Group-ID=1:909,User-
Name=%u
</AuthBy>
```

In this AuthBy-clause there is an EAPTLS file defined that lists every employee. In this way, the users that can access the infrastructure using EAP-TLS are controlled.

The definitions that follow determine what to do with the EAP-request. First the "EAPType TLS" limits the use of this AuthBy-definition for TLS-only. Here regular password authentication is not desired, just certificates. Next, the certificate files are configured and the secret that secures the private-key file can be provided. If there is no secret for the private key, this can be omitted.

The next part defines in what size blocks the EAP-messages should be fragmented. Since some parts of the EAP-TLS challenge are too big to fit in a RADIUS request, the packets should be fragmented.

The MPPE-keys (Microsoft Point to Point Encryption, the protocol for encrypting the data across links) portion is important for wireless access. With 802.1 X, encryption occurs at the edge of the network, between the Access- Point and the client. To provide this secure encryption, a unique key is created and encrypted using the MPPE- keys that are derived from the SSL-challenge. This can be done at the Access-Point and the Client end so that there is no need to transfer the WEP-key in plain text over the air. This, and the fact that the key can be rotated within a period defined by either the Access-Point or the RADIUS server, provides 802.1x users with a good level of security.

The last part of the AuthBy-definition shows how to assign a proper VLAN.

```
<Handler Realm=instituion.cc, EAP-Message=/.+/>
             AuthBy   ID4-TLS
</Handler>
```

The Handler above shows the referral to the AuthBy-definition and some filtering mechanisms to filter out the proper requests. If more things need to be filtered on, they can be added to this handler, as follows:

```
NAS-Port-Type=/^(Wireless-IEEE-802-11|Ethernet)$/
```

In this way, only requests with the proper NAS-Port-Types are allowed. For Accounting purposes, a new handler should be defined in this case, that filters on:

```
"Request-Type=Accounting-Request"
```

since the request does not match the Handler that filters on the EAP-Message.

### EAP-TTLS or EAP-PEAP

When issuing end user certificates is not an option, the EAP-mechanisms PEAP and TTLS can be used.

These two mechanisms look the same in that they both set up a TLS tunnel on which the credentials can be transported. They vary in the supported password encryption schemes.

Virtually all implementations of PEAP encrypt the user's password as an NT hash exclusively. TTLS implementations typically offer plain text transport of the password, called TTLS-PAP (the outer TLS tunnels makes sure the password cannot be eavesdropped) and sometimes other encryption schemes like MS- CHAPv2.

Administratively, the choice whether to use PEAP or TTLS can be challenging, since TTLS is not supported out of the box in the Microsoft Windows environment, therefore third party supplicant needs to be installed by users in case of a TTLS deployment.

Technically, three backend cases need to be considered for deployment:

| Backend stores passwords in... | PEAP-MSCHAPv2? | TTLS? |
|---|---|---|
| plain text or reversibly encrypted | Yes | Yes (TTLS-PAP, TTLS-MSCHAPv2) |
| NT-Hash | Yes | Yes (TTLS-PAP, TTLS-MSCHAPv2) |
| other irreversible encryption | No | Yes (TTLS-PAP) |

Where both options are possible, we suggest the following order of preference: TTLS-MSCHAPv2, PEAP- MSCHAPv2, TTLS-PAP (in descending order of preference).

Instead of a flat file, a more flexible backend for user accounts is a database like MySQL, or LDAP.

```
<Handler TunnelledByPEAP=1, Realm=tunnelled.institution.cc>
         <AuthBy FILE>
                   Filename %D/peap-users
                   EAPType MSCHAP-V2
         </AuthBy>
</Handler>

<Handler TunnelledByTTLS=1, Realm=tunneled.institution.cc>
         <AuthBy FILE>
                   Filename %D/ttls-users
         </AuthBy>
</Handler>
```

In these Handlers, the filtering options "TunneledByPEAP" and "TunnelledByTTLS" define that the tunnelled authentication (with the username and password in it) is handled here.

The "outer authentication", where the SSL tunnel is set up, looks like the TLS handler.

```
<Handler Realm=group_1>
          <AuthBy FILE>
                    Filename %D/users
                    EAPType TTLS, PEAP
                    EAPTLS_CAFile %D/root.pem
                    EAPTLS_CertificateFile %D/server.pem
                    EAPTLS_CertificateType PEM
                    EAPTLS_PrivateKeyFile %D/server.pem
                    EAPTLS_PrivateKeyPassword serverkey
                    EAPTLS_MaxFragmentSize 1024
                    EAPAnonymous anonymous@group1
                    AutoMPPEKeys
           </AuthBy>
</Handler>
```

### Microsoft NPS

The GEANT Campus Best Practice activity has created a very valuable configuration instruction document "Using Windows NPS as RADIUS in eduroam" which is available on the CBP website.

### Cisco ACS

This section begs for community input.

### ClearPass

This section begs for community input.

## RADIUS Accounting in eduroam

eduroam, as a not-for-profit roaming consortium, is in the comfortable position of not having to rely on accurate RADIUS Accounting data.

This is quite fortunate because RADIUS Accounting, even though specified in RFC2866, is very underspecified and has many interoperability issues - especially in a world-wide roaming environment with numerous vendors and firmware versions of WiFi access points, which may all report with RADIUS accounting in a slightly different way. Since it provides no tangible benefit to eduroam operations, RADIUS Accounting is generally frowned upon. For many countries, particularly for European countries, eduroam Operations expects National Roaming Operators to acknowledge and discard Accounting packets destined to an eduroam IdP outside their own national federation; i.e. to confine Accounting, if it is really desired, to their own country.

Some countries do use Accounting information inside their country for various reasons, and eduroam Operations does not interfere with that.

For eduroam SPs, the consequence is that they can safely turn off RADIUS Accounting unless their National Roaming Operator insists on accounting records being sent.

As an eduroam IdP, the consequence is that you may receive accounting records from your own users when they roam to a different hotspot for which RADIUS Accounting has been turned on. Note that the number and content of attributes in the Accounting packets varies greatly due to the underspecification in RFC2866; you can not rely on any single Accounting attribute being present. It is possible, and probably the best option, to simply discard Accounting packets which cannot be correctly understood by your RADIUS server.

## Provisioning configuration details of supplicants to end users

Once the chosen EAP method is configured and the IdP RADIUS server is connected to the authentication backend, the next step is to provision the access configuration to the actual end users.

Many operating systems support IEEE 802.1X and EAP authentication, but the user interfaces in supplicants differ significantly. For some supplicants, manually clicking through a series of GUI pages is the only option. This is sometimes tedious for end users.

If possible, an IdP administrator should prepare pre-configured packages which contain the necessary information to securely connect to eduroam:

- the SSID: "eduroam"
- the crypto setting: WPA2/AES
- the EAP type setting
- the CA that issued the eduroam IdP server's EAP server certificate
- the Common Name in the eduroam IdP server's EAP server certificate

There are tools that can be used to create such auto-installers. The use of one these windows 10 drivers update is recommended, because it will likely have a positive effect on user uptake, and reduce helpdesk load.

### eduroam CAT

eduroam CAT has been created with the sole purpose to ease eduroam installation in many different client platforms through the use of auto-installers. The IdP administrator enters the information listed in the bullets above, after which installers are created for all kinds of platforms for the end users of the IdP. Please see the documentation; or visit the production website at https://cat.eduroam.org.

### Others

In addition to eduroam CAT, there are other tools as well, e.g. su1x and XpressConnect (Cloudpath).