


# MAC Authentication Using FreeRADIUS with hostapd

Authors: Hughann Plucena and Yameen Khan

# Table of Contents

# Introduction

The purpose of this project was to have an open source Radius server on the Raspberry Pi authenticate users to access the WLAN access point. In this project, we were first required to conduct research to find out which open source Radius server is the best to implement, taking pros and cons mainly into consideration.

We were then required to explain the process in detail to explain how the implementation took place. Then we had to show a step by step procedure of the process explained earlier. The purpose of this was for any novice person to follow.

First, we were then required to configure the Raspberry Pi so that it acts as a WLAN Access Point. We did that by configuring the hostapd on one of the wireless interfaces. We then had to make sure that users were able to connect to this access point and get internet access.

Next, we had to authenticate specific users to use this access point by providing a restriction based on MAC addresses. This authentication was done by the open source Radius server implemented on the Raspberry Pi, which in our case was FreeRADIUS version 3.0.

There was only one issue found throughout the implementation. And that was related to successfully implementing MAC authentication on the FreeRADIUS server.

# Research

When it comes to creating and setting up a WLAN access point on a Raspberry Pi, there is only one way to do it. And that is through the hostapd package. The hostapd package is what allows us to create these WLAN access points. Another package that is necessary along with the hostapd package is the dnsmasq package. This package consists of easy to use DHCP and DNS servers. With these packages and correct configurations, we can create a proper WLAN access point on a Raspberry Pi.

Now that the WLAN access point on a Raspberry Pi can be up and running, any station who has the right credentials can authenticate the WLAN access point regardless of the authentication type and method used to implement the WLAN access point. To enhance the security, we can up the security by providing an extra layer of authentication, and that is MAC authentication. There are several ways of implementing a MAC authentication on the WLAN access point. One of the ways is through the hostapd package mentioned earlier. There are two certain files (hostapd.accept and hostapd.deny) that can be used along with the hostapd.conf file to implement the MAC authentication. Another way which is a better way is through another package called freeradius. The reason why FreeRADIUS is better is because it supports more authentication types and it is much easier to use. For instance, implementing WPA-PSK authentication is quite simple whether it is through hostapd or FreeRADIUS. But implementing WPA-EAP authentication is far more complicated through hostapd than through FreeRADIUS. Because with hostapd, the biggest complexity is the configurations related to certificate validation since it is all manual. With FreeRADIUS, the certificates are created by entering one command, meaning it is automated, making it fair and simple for the user. Another benefit of FreeRADIUS is that it is more secure in terms of network security since the passwords are encrypted when sent between the client and the server.

After doing a sufficient amount of research on how to implement this process, we came to the conclusion that we will use FreeRADIUS since it has more pros than cons. And if in the worst case scenario we got stuck somewhere, we can look it up on the internet and find solutions easily since this open source Radius is very popular.

# Process

RADIUS is a protocol, that uses a client to server protocol. What this entail is that RADIUS has its own client and its own server as different protocols.

The client are devices like routers, or switches or even a virtual private network that we use in order to complete the authorization process that RADIUS provides. The server allows for users to be maintained within a database, which dictates who can and who can't connect to the network.

In order for a user to connect to the client, it has to undergo a process where the client has to send a query to the server and the users will only be allowed to join the network if and only if the server authorizes the user to connect.

When a user tries to connect to a RADIUS Client, the Client sends requests to the RADIUS Server. The user can connect to the RADIUS Client only if the RADIUS Server authenticates and authorizes the user.

One of the main reasons why RADIUS is widely used is due to the fact that in increases both the security as well as the privacy of an entity's, businesses and/or organization's system as well as the users within the system. With that being said, it is a vital server in the sense that it makes it more efficient and easier for managing the security for server and system administrators.

The RADIUS works by authenticating a user and authorizing the user soon after the user inputs a designated credential (username and password). The process goes as such: First of, once the user inputs both the username and the password, the RADIUS client will then send it to the RADIUS server to be authenticated. This will then send a request to the server to grant access to the network, what's being sent to the server is what we call the Access-Request message and within the message sent by the client has what we call the shared secret message. Once the server receives the Access-Request message, it will have to ensure that the request came from an authorized client. If the client is not authorized, then the server will get rid of the Access-Request message that the client has sent. There can be two different scenarios as to what goes down after the Access-Request message has been sent out; One, it can be accepted and if it is accepted then the server will have to check if the credentials within the Access-Request message. If the credentials are found to have a match in the server's database, then the server will send an Access-Accept message back to the user who's trying to access the network. Second scenario is if the server finds that there are no

matching credentials in the server database then the server will send out an Access-Reject message. Finally, once the user gets the Access-Accept message, the user will then have been authenticated and authorized to use the RADIUS client.

In this project, Hostapd was to use the Raspberry Pi as it's very own access point. Now, RADIUS is used to authenticate users to the enterprise by filtering MAC Addresses, IP Addresses or by Domain Names, so, when you combine Hostapd, Raspberry Pi and RADIUS server, you'll have a personalized access point that authenticates and authorizes users.

# Procedure

The procedure is divided into three sections. The first section is implementing hostapd. The second section is implementing FreeRADIUS. And the third section is implementing a MAC authentication on the FreeRADIUS server.

## Section 1: hostapd

First, to create and set up a WLAN access point on a Raspberry Pi, install the hostapd package. To do that, open a terminal on the Raspberry Pi. Next, type the command **sudo apt install hostapd** to install the hostapd package. As you can see in Figure 1 below.

```
pi@raspberrypi:~ $ sudo apt install hostapd
```

After installing the hostapd package, the hostapd needs to be unmasked and enabled in order for it to start. Enter these commands one after the other as shown in Figure 2 and 3 below. The commands are **sudo systemctl unmask hostapd** and then **sudo systemctl enable hostapd**.

```
pi@raspberrypi:~ $ sudo systemctl unmask hostapd
```

```
pi@raspberrypi:~ $ sudo systemctl enable hostapd
```

Now in order to manage the clients connecting to the WLAN access point, DNS and DHCP services are required. Install a package called dnsmasq by entering the following command **sudo apt install dnsmasq** as seen in Figure 4 below. This package includes both DNS and DHCP services which are required.

```
pi@raspberrypi:~ $ sudo apt install dnsmasq
```

Next, let's install the netfilter-persistent and iptables-persistent. These two packages are to help maintain the firewall rules set even if the Raspberry Pi boots. The command to install both these packages is **sudo DEBIAN_FRONTEND=noninteractive apt install -y netfilter-persistent iptables-persistent** as seen in Figure 5 below.

```
pi@raspberrypi:~ $ sudo DEBIAN_FRONTEND=noninteractive apt install -y netfilter-
persistent iptables-persistent
```

Now that all the packages are installed, the next step is to do the IP configurations of the wireless interface (in our case it is wlan0). To do that, edit the dhcpcd configuration file by typing the command **sudo nano /etc/dhcpcd.conf**. See Figure 6 below.

```
pi@raspberrypi:~ $ sudo nano /etc/dhcpcd.conf
```

Add the following that is shown in Figure 7 below to the end of this file. Once added, save it and close it.

```
interface wlan0
    static ip_address=192.168.4.1/24
    nohook wpa_supplicant
```

Next, lets enable routing by creating a file using the command **sudo nano /etc/sysctl.d/routed-ap.conf** as shown in Figure 8 below.

```
pi@raspberrypi:~ $ sudo nano /etc/sysctl.d/routed-ap.conf
```

Within this file, add the following that is shown below in Figure 9. Once done, save it and close it.

```
net.ipv4.ip_forward=1
```

Since the network 192.168.4.0 is not configured on the main access point, the clients on this network will not be able to communicate unless if the main access point is configured, which is not required since the masquerade firewall rule using the command **sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE** substitute the IP addresses of clients on that network, allowing them to communicate. See Figure 10 below.

```
pi@raspberrypi:~ $ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Now that the firewall rule has been applied, lets save it to the netfilter-persistent service by using the command **sudo netfilter-persistent save** as seen in Figure 11 below. This

will preserve the configurations and load them every time during the boot process to maintain the firewall rules.

```
pi@raspberrypi:~ $ sudo netfilter-persistent save
```

Now that the firewall rules are set and saved, lets configure other services. Configure the DHCP and DNS services by editing the dnsmasq configuration file by entering the following command **sudo nano /etc/dnsmasq.conf**. As seen in Figure 12 below.

```
pi@raspberrypi:~ $ sudo nano /etc/dnsmasq.conf
```

Add the following that is shown in Figure 13 below to the end of this file. Once added, save it and close it.

```
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
domain=wlan
address=/gw.wlan/192.168.4.1
```

Now it may be possible that the Wi-Fi radio is blocked because of the regulators around the world. So to ensure that is not the case, use the command **sudo rfkill unblock wlan** as shown in Figure 14 below.

```
pi@raspberrypi:~ $ sudo rfkill unblock wlan
```

Now that it is ensured that the Wi-Fi radio is unblocked. Create a configuration file using the command **sudo nano /etc/hostapd/hostapd.conf** as shown in Figure 15 below.

```
pi@raspberrypi:~ $ sudo nano /etc/hostapd/hostapd.conf
```

This empty file will consist of information related to the WLAN access point. So add the following that is shown in Figure 16 below. Once added, save it and close it.

```
country_code=GB
interface=wlan0
ssid=NameOfNetwork
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=PasswordOfNetwork
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Finally, restart the Raspberry Pi using the command **sudo systemctl reboot** as shown in Figure 17 below.

```
pi@raspberrypi:~ $ sudo systemctl reboot
```

Once the Raspberry Pi has rebooted, the clients should be able to connect to the WLAN access point and get internet access. That is if this section of the procedure was carried out the same way as above.

## Section 2: FreeRADIUS

The WLAN access point implemented from the previous section uses a WPA-PSK authentication, but the WPA-EAP authentication that uses 802.1X standard is much more secure since the session between the client and the WLAN access point is encrypted using the TLS tunnel setup. For this section, the configuration that will be done on the FreeRADIUS server will be such that it will authenticate any client to access the WLAN access point. Later in section 3 the authentication will be based on the MAC address.

Before anything, first lets configure the hostapd file from the last section to make some changes that will implement the WPA-EAP authentication on the WLAN access point. To make the changes, use the command **sudo nano /etc/hostapd/hostapd.conf** as shown in Figure 18 below.

```
pi@raspberrypi:~ $ sudo nano /etc/hostapd/hostapd.conf
```

Within this file, edit it so that it matches Figure 19 below. So add and remove accordingly. Once done, save it and close it.

```
country_code=GB
interface=wlan0
ssid=halahalahala
hw_mode=g
channel=7
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-EAP
wpa_pairwise=TKIP CCMP
ieee8021x=1
eapol_key_index_workaround=1
eap_server=0
own_ip_addr=127.0.0.1
auth_server_addr=127.0.0.1
auth_server_port=1812
auth_server_shared_secret=testing123
```

Before installing the FreeRADIUS package, check the Raspberry Pi for any update(s) using the command **sudo apt update** as shown in Figure 20 below.

```
pi@raspberrypi:~ $ sudo apt update
```

Now that the Raspberry Pi is uptodate, install the FreeRADIUS package by typing the following command: **sudo apt install freeradius** as seen in Figure 21 below. Note: This may take a while.

```
pi@raspberrypi:~ $ sudo apt install freeradius
```

Once the FreeRADIUS package is installed, generate a 2048-bit DH Parameter key using the command **sudo openssl dhparam -out /etc/freeradius/3.0/certs/dh 2048**. As seen in Figure 22 below. Note: This may take much longer than before.

```
pi@raspberrypi:~ $ sudo openssl dhparam -out /etc/freeradius/3.0/certs/dh 2048
```

Now type the following command **sudo -i** which will give root privileges to execute a command. See Figure 23 below.

```
pi@raspberrypi:~ $ sudo -i
```

Next, let's generate two certificates, the ca (certificate authority) certificate and the server certificate. To generate the ca certificate, edit the ca configuration file by using the command **nano /etc/freeradius/3.0/certs/ca.cnf**. See Figure 24 below.

```
root@raspberrypi:~# nano /etc/freeradius/3.0/certs/ca.cnf
```

In this file, look for input_password and output_password under the req category, and add a passphrase for each (in our case it is "password" for both). See Figure 25 below. Do not save it and close it yet.

```
[ req ]
prompt                  = no
distinguished_name      = certificate_authority
default_bits            = 2048
input_password          = password
output_password         = password
x509_extensions         = v3_ca
```

Next, look for certificate_authority category and add the correct information as seen in Figure 26 below. Once added, save it and close it.

```
[certificate_authority]
countryName             = US
stateOrProvinceName     = NY
localityName            = Five Guys
organizationName        = Example Inc.
emailAddress            = admin@example.org
commonName              = "Example Certificate Authority"
```

Now that the ca configuration is done, let's generate the ca certificate. But first, change the directory to the certs directory using the command **cd /etc/freeradius/3.0/certs/**. See Figure 27 below.

```
root@raspberrypi:~# cd /etc/freeradius/3.0/certs/
```

Now enter the **make ca.pem** command to generate the ca certificate. See Figure 28 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# make ca.pem
```

Now that the ca certificate is generated, lets generate the server certificate. But first, configure the server file by using the command **nano /etc/freeradius/3.0/certs/server.cnf**. As shown in Figure 29 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/certs/server.cnf
```

In this file, look for input_password and output_password under the req category, and add a passphrase for each (in our case it is "password" for both). See Figure 30 below. Do not save it and close it yet.

```
[ req ]
prompt                  = no
distinguished_name      = server
default_bits            = 2048
input_password          = password
output_password         = password
```

Next, look for the server category and add the correct information as seen in Figure 31 below. Once added, save it and close it.

```
[server]
countryName             = US
stateOrProvinceName     = NY
localityName            = Five Guys
organizationName        = Example Inc.
emailAddress            = admin@example.org
commonName              = "Example Server Certificate"
```

Now that the server configuration is done, let's generate the server certificate by using the command **make server.pem**. See Figure 32 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# make server.pem
```

Now that all the required certificates are generated. Lets configure the eap file by using the command **nano /etc/freeradius/3.0/mods-enabled/eap**. See Figure 33 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/mods-enabled/eap
```

Within this file, inside the eap function, look for default_eap_type and change it to tls as shown in Figure 34 below. Do not save it and close it yet.

```
eap {
        #  Invoke the default supported EAP type when
        #  EAP-Identity response is received.
        #
        #  The incoming EAP messages DO NOT specify which EAP
        #  type they will be using, so it MUST be set here.
        #
        #  For now, only one default EAP type may be used at a time.
        #
        #  If the EAP-Type attribute is set by another module,
        #  then that EAP type takes precedence over the
        #  default type configured here.
        #
        default_eap_type = tls
```

Next, look for the tls-config tls-common function and set the private_key_password to the passphrase set in output_password in the server.cnf file (refer to Figure 23 to see the passphrase set for output_password in the server.cnf file). The setting is done below in Figure 35. Save it and close it.

```
        tls-config tls-common {
                private_key_password = password
                private_key_file = /etc/ssl/private/ssl-cert-snakeoil.key
```

Now that the eap file configuration is done, lets configure the clients file by using the command **nano /etc/freeradius/3.0/clients.conf** as seen in Figure 36 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/clients.conf
```

Within this file, look for the client function and add the IP address of the wireless interface (in our case wlan0) in the function after the client. In the secret, add the passphrase. In the shortname, add the name of the network. See Figure 37 below for the configuration. Save it and close it.

```
client 192.168.4.1/24 {
    secret    = password
    shortname = halahalahala
}
```

With the clients configuration done, lets configure the users file by entering the following command **nano /etc/freeradius/3.0/users**. See Figure 38 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/users
```

Within this file, add the name of the user and the password that will be used by the client as credentials to authenticate to the WLAN access point. See Figure 39 below to get the idea. Save it and close it.

```
bob     Cleartext-Password := "hello"
```

Now that all the necessary configurations are done, start the server using the command **/etc/init.d/freeradius start** as seen in Figure 40 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# /etc/init.d/freeradius start
```

Now that the FreeRADIUS server is up and running, lets test the connection using the command **radtest [Username] [Password] localhost [shared_secret_key]**. See Figure 41 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# radtest bob hello localhost 0 testing123
Sent Access-Request Id 104 from 0.0.0.0:57506 to 127.0.0.1:1812 length 73
        User-Name = "bob"
        User-Password = "hello"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 0
        Message-Authenticator = 0x00
        Cleartext-Password = "hello"
Received Access-Accept Id 104 from 127.0.0.1:1812 to 127.0.0.1:57506 length 20
root@raspberrypi:/etc/freeradius/3.0/certs#
```

With the test being successful, the FreeRADIUS server has been successfully implemented and as of now authenticates any client that has the right credentials. That is if this section of the procedure was carried out the same way as above.

## Section 3: MAC Authentication

From the last section, the FreeRADIUS server was implemented only to authenticate everyone with the right credentials to the WLAN access point. In this section, the MAC authentication will be applied so that the FreeRADIUS server allows only specific MAC address stations to authenticate.

To implement this, first, configure the canonicalization file by using the command **nano /etc/freeradius/3.0/policy.d/canonicalization**. See Figure 42 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/policy.d/canonicalization
```

Within this file, remove any existing configuration (or add the # symbol at the start of each line) and add the following configuration shown in Figure 43 below. Once done, save it and close it.

```
rewrite_calling_station_id {
        if (Calling-Station-Id =~ /([0-9a-f]{2})[-:]?([0-9a-f]{2})[-:.]?([0-9a-f]{2})[-:]?([0-9a-f]{2})[-:.]?([0-9a-f]{2})[-:$
                update request {
                        Calling-Station-Id := "%{tolower:%{1}-%{2}-%{3}-%{4}-%{5}-%{6}}"
                }
        }
        else {
                noop
        }
}
```

In Figure 43 above, the if statement line goes beyond the end of screen, so copy and paste this all in that line instead: **if (Calling-Station-Id =~ /([0-9a-f]{2})[-:]?([0-9a-f]{2})[-:.]?([0-9a-f]{2})[-:]?([0-9a-f]{2})[-:.]?([0-9a-f]{2})[-:]?([0-9a-f]{2})/i) {**

Now that the canonicalization file is configured, lets configure the default file using the command **nano /etc/freeradius/3.0/sites-available/default** as shown in Figure 44 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/sites-available/default
```

Within this file, look for the authorize function, and at the start of that function, add the following that is shown in Figure 45 below. Once done, save it and close it.

```
preprocess
rewrite_calling_station_id
if (Calling-Station-ID == "1a-2b-3c-4d-5e-6f") {
        reject
}
else {
        update control {
                Auth-Type := Accept
        }
}
```

In Figure 45 above, within the if statement, replace the MAC address with the MAC address(es) that are to be rejected.

Now let's configure the radiusd file to allow the radius log file to log all the information. Enter the command **nano /etc/freeradius/3.0/radiusd.conf** as shown below in Figure 46.

```
root@raspberrypi:/etc/freeradius/3.0/certs# nano /etc/freeradius/3.0/radiusd.conf
```

Within this file, look for the log function. Within that, look for auth, auth_badpass, and auth_goodpass and set them all to yes. As seen in Figure 47 below. Once done, save it and close it.

```
        auth = yes
        auth_badpass = yes
        auth_goodpass = yes
```

With all the necessary files configured. Let's stop the FreeRadius service and then start it back. To do that, use the following commands one after the other **/etc/init.d/freeradius stop** and then **/etc/init.d/freeradius start**. See Figures 48 and 49 below.

```
root@raspberrypi:/etc/freeradius/3.0/certs# /etc/init.d/freeradius stop
root@raspberrypi:/etc/freeradius/3.0/certs# /etc/init.d/freeradius start
```

Next, to test if the MAC authentication is working, open another terminal on the Raspberry Pi and enter the command **sudo hostapd /etc/hostapd/hostapd.conf** like the one in Figure 50 below.

```
pi@raspberrypi:~ $ sudo hostapd /etc/hostapd/hostapd.conf
```

Depending on the configurations done above, the station will either get authenticated or denied. To confirm that, enter the command **sudo tail /var/log/freeradius/radius.log** as shown in Figure 51 below. This file is a log file that verifies the doubt.

```
pi@raspberrypi:~ $ sudo tail /var/log/freeradius/radius.log
```

With this done, the FreeRADIUS server should be authenticating stations based on their MAC addresses. That is if this section of the procedure was carried out the same way as above. This marks the end of the entire procedure from start to finish.

# Troubleshooting

There was one major issue in particular that we faced at the end of this project for quite some time. And that was applying the MAC address restriction for stations connecting to the WLAN access point. It was assumed wrong at first that we were supposed to do this implementation within the hostapd by using the hostapd.accept or/and hostapd.deny files and calling those files in the hostapd.conf file. Until when we checked the log files, where we were not able to see any desired output such as ‘Login OK’ or ‘Invalid Login’ based on the configurations. At that point we realized that we were doing something wrong and decided to do the implementation within the FreeRADIUS.

When it came to implementing this on FreeRADIUS, it was proven quite a challenge. Unlike hostapd, where we had hostapd.accept, hostapd.deny, and hostapd.conf files to use, there were far too many files and directories to look into in FreeRADIUS. It is at this point where we decided to look up on the internet for some ideas and solutions. We were able to come across quite a few solutions that were related to our scenario. Unfortunately, most of them did not help much. But there was this one solution that gave us a hint of what we were supposed to do next in order for this to be implemented. And that was getting the MAC address of the station trying to connect to the WLAN access point, which was then compared with an if statement to the MAC address already defined in that same file. And based on that, it would either authenticate or reject that station from connecting to the WLAN access point. This depended on what permissions were given in the if block and the else block. With the if statement added, the MAC authentication on FreeRADIUS was implemented and working successfully, which the logs file proved it all. More of this is discussed (with screenshots) in the Procedure section of this document.

# Conclusion

From this project, we learned how to create and set up our own WLAN access point on the Raspberry Pi using hostapd. We also learned how to implement FreeRADIUS server and authenticate stations that are trying to connect to our WLAN access point based on the MAC address. We also learned how to overcome some of the most challenging issues that we never came across in terms of difficulty.

This project helped us develop a lot of skills, especially in the research area since the context of this project was not to the best of our knowledge and understanding. What we found of interest though was how the FreeRADIUS server was authenticating MAC addresses of stations based on the configurations done. Even though it took a while to figure it out, it was yet so satisfying to watch when it got implemented.

What we wished we could have learned more about was to know the process of certificates, how certificates work and how it plays a vital role in the security field. We were also keen to exploit various configuration files within the FreeRADIUS, but we were not confident enough to do so because we had the fear in our mind of doing something wrong that we would end up regretting. So, an overview or a module about FreeRADIUS would be very helpful. Apart from that, the idea of this project was interesting and challenging.

# References

"BYOD and Base on MAC." *Users - BYOD and Base on MAC*, freeradius.1045715.n5.nabble.com/BYOD-and-base-on-MAC-td5748995.html.

"MAC Authentication Bypass." *MAC Authentication Bypass – Benim Hayalim*, www.benimhayalim.com/mab-configuration/.

*RADIUS MAC Authentication*, www.watchguard.com/help/docs/help-center/en-US/Content/en-US/Wi-Fi-Cloud/manage_wirelessmanager/configuration/wifi_access/radius_mac_auth.html.

"Setting up a Raspberry Pi as a Routed Wireless Access Point." *Setting up a Raspberry Pi as a Routed Wireless Access Point - Raspberry Pi Documentation*, www.raspberrypi.org/documentation/configuration/wireless/access-point-routed.md.

"Setting up RADIUS on a Raspberry Pi." *EE Turunen*, 12 Dec. 2019, turunen.ee/setting-up-radius-on-a-raspberry-pi/.