

# On the Implementation of Authentication Mechanisms for Mobile Devices over Wireless Networks

Tom Blount

January 25, 2013

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Brief . . . . .	3
1.2	Requirements . . . . .	3
<b>2</b>	<b>802.1X</b>	<b>4</b>
2.1	Overview . . . . .	4
2.2	Components . . . . .	4
2.2.1	Authentication server . . . . .	4
2.2.2	Authenticator . . . . .	5
2.2.3	Supplicant . . . . .	5
2.3	EAP . . . . .	5
<b>3</b>	<b>Implementation</b>	<b>6</b>
3.1	Overview . . . . .	6
3.2	Components . . . . .	6
3.2.1	Server . . . . .	6
3.2.2	Access point . . . . .	6
3.2.3	Client . . . . .	7
3.3	Challenges . . . . .	7
<b>4</b>	<b>802.11r</b>	<b>8</b>
4.1	Overview . . . . .	8
4.2	Implementation . . . . .	8
<b>5</b>	<b>Evaluation</b>	<b>9</b>
5.1	Testing . . . . .	9
5.2	Conclusions . . . . .	9
5.3	Recommendations for future work . . . . .	9
<b>A</b>	<b>Example Code</b>	<b>11</b>
A.1	RADIUS Server . . . . .	11
A.1.1	radddb/users . . . . .	11
A.1.2	radddb/clients.conf . . . . .	11
A.1.3	radddb/eap.conf . . . . .	11
A.2	Authenticator . . . . .	12
A.2.1	hostapd.conf . . . . .	12
A.3	Supplicant . . . . .	12
A.3.1	wpa_supplicant.conf . . . . .	12
<b>B</b>	<b>Testing Output</b>	<b>13</b>
B.1	Simulation . . . . .	13
B.2	wpa_supplicant output . . . . .	14
B.3	802.11r output . . . . .	14

**Abstract** This report gives an overview of the 802.1X specification and describes the design and implementation undertaking in this investigation. The report then covers an extension to this project, the 802.11r specification, used for fast switching between access points, and determines its overall feasibility.

**Acknowledgements** Thanks of course go to the other members of my group, José Cubero, Thomas Grainger and Chris Orchard. Thanks also go to the Southampton University Wireless Society (SUWS) for lending us two wireless nodes.

# 1 Introduction

## 1.1 Brief

The initial brief we were given was to:

*“Design and demonstrate a simulation of the implementation of an efficient authentication protocol for mobile networks.”*

After discussing amongst ourselves on the merits and drawbacks of various approaches to this task (including a number of extensions to the initial specification that we considered interesting avenues of research in their own right), we consulted J. Reeve and S.J. Braithwaite to confirm our approach was appropriate. From this discussion and the preceding brief, we extrapolated and defined the following requirements for the project, presented below in MoSCoW format.

## 1.2 Requirements

**Must** Implement a simulated authentication protocol and demonstrate it working

**Must** Implement a working authentication protocol, using a single client, a single access point, and a single server and demonstrate it working

**Should** Demonstrate the use of an additional access point and the switching of the client between the two

**Could** Investigate the possibility of “fast” transitioning between access points

## 2 802.1X

### 2.1 Overview

802.1X is a method of controlling access to a local area network at the data-link layer (layer two of the Open Systems Interconnection model) of the network. Three components are defined in the specification: the supplicant, the authenticator and the authentication server (see section 2.2 for details).

Because 802.1X controls authentication through the data-link layer, local abuse within a wireless hotspot, for example, is prevented (as opposed to other authentication methods such as web-redirect). This also has the benefit of preventing unauthorised clients from being allocated an IP address – this can be invaluable to campuses with few IPv4 addresses (even more so since the exhaustion of the IPv4 address space in January 2011 [1]).

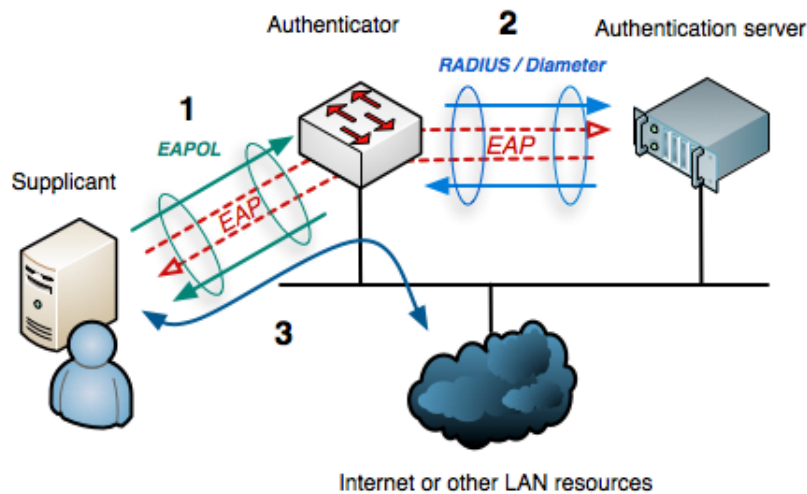


Figure 1: An image of how 802.1X works over a (wired) network [2]

### 2.2 Components

#### 2.2.1 Authentication server

The authentication server holds the credentials of clients, to determine if they should be granted access to the private section of the network (often the internet), or if their access has been revoked.

Despite there being alternative protocols for server/authenticator communication (such as Diameter<sup>1</sup> and TACACS<sup>2</sup>), RADIUS is the *de facto* standard, used by many institutions (such as eduroam<sup>3</sup>) [3].

<sup>1</sup>RFC 6733: <http://www.ietf.org/rfc/rfc6733.txt>

<sup>2</sup>RFC 1492: <http://www.ietf.org/rfc/rfc1492.txt>

<sup>3</sup><http://www.eduroam.org/>

The RADIUS protocol provides centralized AAA: Authentication, the ability to verify a user's identity via their credentials (e.g. a username and password); Authorisation, the ability to define different levels of access to users with different (predetermined) levels of clearance; and Accounting, the ability to track the access to and usage of these resources over periods of time and space.

A RADIUS server can also query a database back-end storing credentials, or even proxy requests to a remote RADIUS server - this can be used when roaming, on the same credentials, nationally or even internationally (using eduroam, for example).

### 2.2.2 Authenticator

The authenticator is usually a network switch or, in the case of wireless networks, an access point. Its role is to prevent unauthorised access to the private network "behind" it until a client is authenticated. To facilitate this, the authenticator relays messages from the client to the authentication server.

### 2.2.3 Supplicant

The supplicant can refer to both the client device wishing to be authenticated and to the software running on the client that actually provides its authentication details to the authenticator.

## 2.3 EAP

802.1X also defines the use of Extensible Authentication Protocol (a framework for authentication communication via networks, defined in RFC 3748<sup>4</sup>) over IEEE802 local area networks, or EAPOL. EAPOL is used to encapsulate EAP messages sent between the supplicant and authenticator.

EAP-Tunnelled Transport Layer Security (EAP-TTLS) is an EAP method that provides a degree of privacy and security for the client. The client first verifies the identity of the server using a PKI certificate (and, optionally, vice versa) and then establishes a secure tunnel between themselves and the authenticator using "anonymous" credentials. They can then send their actual credentials privately and securely through the tunnel.

---

<sup>4</sup><http://www.ietf.org/rfc/rfc3748.txt>

## 3 Implementation

### 3.1 Overview

The primary requirement of creating a simulated authentication could be completed by setting up a lone authentication server (see section 3.2.1 below) and running a virtual supplicant locally. This simulated messages passing from the supplicant via an authenticator and to the server.

The next stage was to implement the supplicant and authenticator in reality (see sections 3.2.2 and 3.2.3 below).

### 3.2 Components

#### 3.2.1 Server

Implementing (as well as debugging and testing) the authentication server made up the bulk of this author's individual contribution. The consisted of researching the appropriate server-side software, installing it on a virtual machine hosted by ECS and configuring it correctly.

While there are many variants of RADIUS software, we chose **freeradius** due to its open-source nature and popularity, making it ideal for use in this project. The decision was made to install a RPM package that had been pre-built in a controlled environment.

Next, the configuration files must be updated with the appropriate system details. The essential items of configuration are determining the credentials of the users who will be accessing the system (their username, password and, crucially, whether they should be allowed or rejected), and the clients they will use (the IP address for example). In this case a user **TomB** was set to be authorised when using the password **TomBsPassword**. A second user, **TomG** was set to be always refused access (to demonstrate revoked credentials - see section 5.1). The next stage was to generate the necessary certificates that were used to identify the server (using EAP-TTLS). These were edited and made from within **raddb/certs/**. Finally, the appropriate EAP method needed to be configured, and pointed towards the generated certificates.

Example code from these key files can be found in appendix A.1.

#### 3.2.2 Access point

We received on loan from the Southampton University Wireless Society (SUWS)<sup>5</sup> two “Meraki Mini” access points to use throughout the course of this project. The Meraki Mini has a 180MHz CPU and a 60mW 802.11b/g radio.

---

<sup>5</sup><http://www.sown.org.uk>



Figure 2: A Meraki Mini access point, similar to the ones used in this project [4]

However, instead of running the default Meraki firmware, the nodes were re-flashed to use `OpenWrt`<sup>6</sup>, a distribution of Linux, designed for use on embedded devices (these nodes were purchased before Meraki changed their EULA to prohibit such). This was used to build and run `hostapd` authenticator software<sup>7</sup> which runs in user space, in the background, relaying messages from supplicant to server.

### 3.2.3 Client

The client runs the supplicant software `wpa_supplicant`. This must have the same credentials as those preprogrammed on the server and must be configured with the appropriate username and password combination. Some supplicant software includes a GUI allowing users to input these details at run time, but we chose to preconfigure ours - an example config file is provided in appendix A.3.

## 3.3 Challenges

Throughout implementation we faced, and overcame, a number of challenges. Certainly one of the most arduous was attempting to configure the embedded devices with bleeding-edge software, as almost all programs needed to be cross-compiled. Another issue was that, as the authentication server was running as a virtual machine within ECS, the connection needed to be routed through a VPN (in this instance, one of our personal computers).

---

<sup>6</sup><https://openwrt.org/>

<sup>7</sup><http://w1.fi/>



## 4 802.11r

### 4.1 Overview

Developed in 2008, the 802.11r specification amends the 802.11 “WiFi” specification to allow fast Basic Service Set transitions (FTs) – in other words, it is designed to allow a mobile device to be quickly and securely transferred between access points. To achieve this, instead of renegotiating a new session and key after every transfer, part of the key is cached to allow further communication after transferring access points for some amount of time.

While this is, of course, desirable for any mobile network it is particularly important given the recent proliferation of internet-capable smart phones and Voice Over IP (VOIP) technology.

### 4.2 Implementation

Implementing this protocol involved recompiling both the authenticator (`hostapd`) and supplicant (`wpa_supplicant`) to support 802.11r, and configuring them accordingly. The utility `wpa_cli` could then be used to interact with `wpa_supplicant` and force the supplicant to roam to a new access point.

Unfortunately however, during testing we could not get the fast transition to occur. As can be seen in appendix B.3, the line `FT: Invalid group cipher (0)` which translates as the program receiving `WPA_CIPHER_NONE`. This state should not be reachable with the given configuration (it should be either `WPA_CIPHER_CCMP` or `WPA_CIPHER_TKIP` at this stage) and was possibly caused by bugs introduced by the bleeding-edge software used - regrettably there was not enough time remaining at this stage of the project to determine and fix the root cause.

## 5 Evaluation

### 5.1 Testing

Each of the implemented sections were tested incrementally, using a command line tool bundled with `wpa_supplicant` called `eapol_test`. This allows simulated EAP messages to be sent to a server, based on settings in a configuration file (specifying the EAP method, username, password, etc.) and the responses to be examined in detail.

The output from testing the server (and the simulated authenticator and supplicant) can be seen in appendix B.1 - the server is shown on the left (in debug mode, to produce the most detailed output) and the `eapol_test` on the right. Initially a valid user attempted to connect and was successfully authenticated. Next, a valid user with revoked clearance attempted to connect and was (correctly) refused access. Other tests of this nature (such as a valid user, with incorrect credentials) were performed and passed.

Once the authenticator and supplicant were implemented fully, these were also tested (and debugged as necessary) with `eapol_test`. They were then run in a live environment repeating the above tests - a sample of the output is shown in appendix B.2.

### 5.2 Conclusions

Overall, the project was a resounding success. We managed to implement not only a simulated authentication service, but a live demonstration of an efficient system with multiple hardware components. Although the proposed 802.11r extension was not fully implemented, the research into the protocol raised some interesting ideas for future work.

### 5.3 Recommendations for future work

The clear next stage for further development would be to fully implement an instance of 802.11r, and demonstrate the possible fast switching between access points. Ideally, a comparison could be drawn between the time taken using fast BSS and the transfers shown in this project, potentially with a practical demonstration of running a VOIP call during the transfer.

## References

- [1] ICAAN, *Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied*, ICAAN News Release, February 2011
- [2] Image by Arran Cudbard-Bell
- [3] H.F. Tipton and M. Krause, “Centralized Authentication Services”, in *Information Security Management Handbook*, 5xth ed. CRC Press, 2004, pp. 99.
- [4] M. Honan, *Meraki Mini Serves Up Hot Heaping Wi-Fi*, *Wired*, September 2007. Available: [http://www.wired.com/images/productreviews/2008/09/meraki\\_mini\\_f.jpg](http://www.wired.com/images/productreviews/2008/09/meraki_mini_f.jpg)

## A Example Code

### A.1 RADIUS Server

#### A.1.1 raddb/users

```
#Give access to this user
TomB    Cleartext-Password := "TomBsPassword"
        Reply-Message = "Hello there %{User-Name}! How's it going?"

#Don't give access to this user
TomG    Auth-Type := Reject
        Reply-Message = "Oh no you don't Grainger. Not again."
```

#### A.1.2 raddb/clients.conf

```
client linuxproj {
    ipaddr = 152.78.71.57
    secret = testing123
    require_message_authenticator = no

    #Used to specify a specific Network Access Server type
    #(e.g. cisco, multitech, etc.)
    nastype = other
}
```

#### A.1.3 raddb/eap.conf

```
tls {
    certdir = ${confdir}/certs
    cadir = ${confdir}/certs
    private_key_password = whatever
    private_key_file = ${certdir}/server.pem
    certificate_file = ${certdir}/server.pem
    CA_file = ${cadir}/ca.pem
    dh_file = ${certdir}/dh
    random_file = ${certdir}/random
    CA_path = ${cadir}
    cipher_list = "DEFAULT"
}

ttls {
    default_eap_type = md5
    copy_request_to_tunnel = no
}
```

```
use_tunneled_reply = no
virtual_server = "inner-tunnel"
}
```

## A.2 Authenticator

### A.2.1 hostapd.conf

```
auth_server_addr=152.78.61.5
auth_server_port=1812
auth_server_shared_secret=testing123
disable_pmksa_caching=1
okc=0
nas_identifier=kanga-cso1g09e
eapol_key_index_workaround=1
ieee8021x=1
wpa_key_mgmt=FT-EAP WPA-EAP
auth_algs=1
wpa=2
wpa_pairwise=CCMP
ssid=notthebees
bridge=br-lan
wmm_enabled=1
bssid=00:18:0a:01:3f:38
ignore_broadcast_ssid=0
mobility_domain=a1b2
r0_key_lifetime=100000
r1_key_holder=000102030406
reassociation_deadline=1000
r0kh=00:18:0a:01:3f:3f kanga-cso1g09e
      000102030405060708090a0b0c0d0e0f
r1kh=00:18:0a:01:3f:3f 00:01:02:03:04:05
      000102030405060708090a0b0c0d0e0f
r1kh=00:18:0a:01:3f:38 00:01:02:03:04:06
      000102030405060708090a0b0c0d0e0e
```

Figure 3: Code snippet by Chris Orchard

## A.3 Supplicant

### A.3.1 wpa\_supplicant.conf

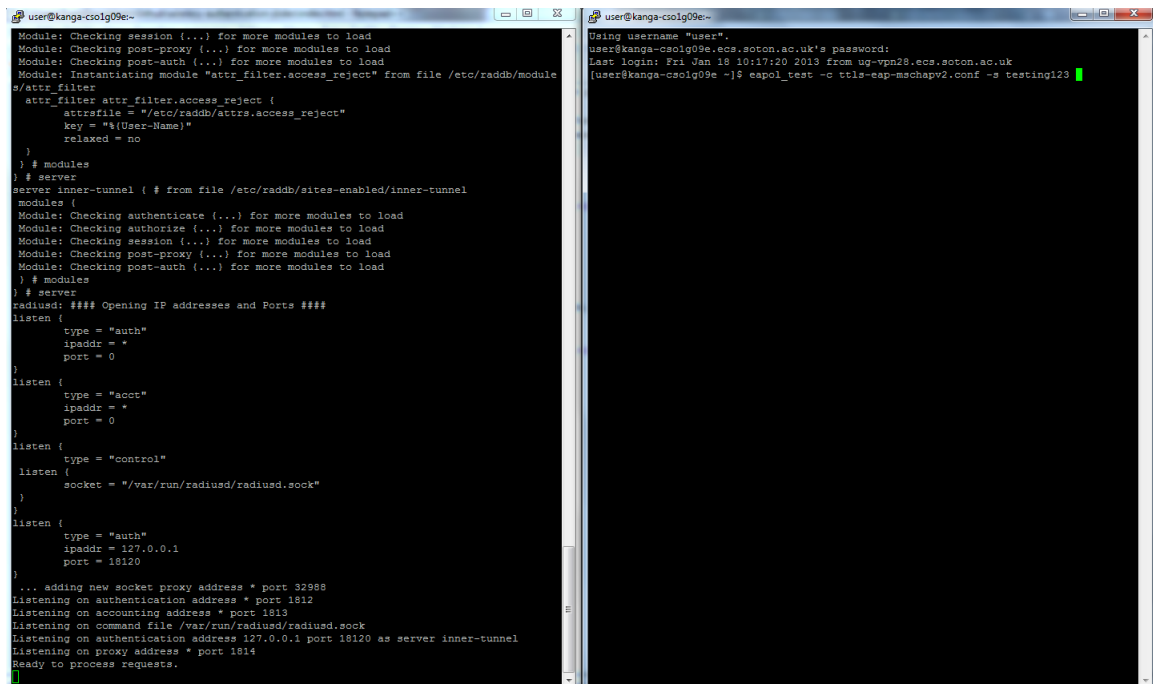
```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=1
ap_scan=1
fast_reauth=1

network={
    ssid="notthebees"
    scan_ssid=1
    key_mgmt=FT-EAP WPA-EAP
    pairwise=CCMP
    group=CCMP
    eap=TTLS
    identity="TomB"
    anonymous_identity="anon"
    password="TomBsPassword"
    phase2="auth=PAP"
}
```

Figure 4: Code snippet by Thomas Grainger

## B Testing Output

### B.1 Simulation



The image shows two terminal windows side-by-side. The left window displays the configuration of a server, likely a RADIUS server, with various modules and listening ports. The right window shows the output of a test command, including a successful login for a user named 'user'.

```
user@kanga-cs0lg09e:~$ cat /etc/raddb/sites-enabled/inner-tunnel
Module: Checking session (...) for more modules to load
Module: Checking post-proxy (...) for more modules to load
Module: Checking post-auth (...) for more modules to load
Module: Instantiating module "attr_filter.access_reject" from file /etc/raddb/module
s/attr_filter
    attr_filter attr_filter.access_reject {
        attrfile = "/etc/raddb/attrs.access_reject"
        key = "%(User-Name)"
        relaxed = no
    }
} # modules
} # server
server inner-tunnel { # from file /etc/raddb/sites-enabled/inner-tunnel
    modules {
        Module: Checking authenticate (...) for more modules to load
        Module: Checking authorize (...) for more modules to load
        Module: Checking session (...) for more modules to load
        Module: Checking post-proxy (...) for more modules to load
        Module: Checking post-auth (...) for more modules to load
    } # modules
} # server
radiusd: ### Opening IP addresses and Ports ###
listen {
    type = "auth"
    ipaddr = *
    port = 0
}
listen {
    type = "acct"
    ipaddr = *
    port = 0
}
listen {
    type = "control"
    listen {
        socket = "/var/run/radiusd/radiusd.sock"
    }
}
listen {
    type = "auth"
    ipaddr = 127.0.0.1
    port = 18120
}
... adding new socket proxy address * port 32988
Listening on authentication address * port 1812
Listening on accounting address * port 1813
Listening on command file /var/run/radiusd/radiusd.sock
Listening on authentication address 127.0.0.1 port 18120 as server inner-tunnel
Listening on proxy address * port 1814
Ready to process requests.

user@kanga-cs0lg09e:~$ eapol_test -c ttls-eap-mschapv2.conf -s testing123
Using username "user".
user@kanga-cs0lg09e:~$ eapol_test -c ttls-eap-mschapv2.conf -s testing123
Last login: Fri Jan 18 10:17:20 2013 from ug-vpn28.ecs.soton.ac.uk
[user@kanga-cs0lg09e ~]$
```

Figure 5: Setting up the server for testing

```

user@kanga-cs0lg09e~
[files] users: Matched entry TomB at line 71
[files] expand: Hello %(User-Name) -> Hello TomB
++[files] returns ok
++[expiration] returns noop
++[login] returns noop
[pap] WARNING: Auth-Type already set. Not setting to PAP
++[pap] returns noop
Found Auth-Type = PAP
# Executing group from file /etc/raddb/sites-enabled/inner-tunnel
-- entering group authenticate (...)
[eap] Request found, released from the list
[eap] EAP/md5
[eap] processing type md5
[eap] Freeing handler
++[eap] returns ok
WARNING: Empty post-auth section. Using default return values.
# Executing section post-auth from file /etc/raddb/sites-enabled/inner-tunnel
? # server inner-tunnel
[ttls] Got tunneled reply code 2
Reply-Message = "Hello TomB"
EAP-Message = 0x03010004
Message-Authenticator = 0x00000000000000000000000000000000
User-Name = "TomB"
[ttls] Got tunneled Access-Accept
[eap] Freeing handler
rlm_eap_ttls: Freeing handler for user TomB
++[eap] returns ok
# Executing section post-auth from file /etc/raddb/sites-enabled/default
-- entering group post-auth (...)
++[exec] returns noop
Sending Access-Accept of id 7 to 127.0.0.1 port 33153
MS-MPPE-Recv-Key = 0x2268952488feca54319ef3937363a5747a4408f827532cf91f5ae
c8080930
MS-MPPE-Send-Key = 0x40ceb9a2c83986f5d4b21e316965c915de1a2f0eeca1284f51678
4fcd32fa
EAP-Message = 0x03070004
Message-Authenticator = 0x00000000000000000000000000000000
User-Name = "anonymous"
Finished request 7.
Going to the next request
Waking up in 4.9 seconds.
Cleaning up request 0 ID 0 with timestamp +46
Cleaning up request 1 ID 1 with timestamp +46
Cleaning up request 2 ID 2 with timestamp +46
Cleaning up request 3 ID 3 with timestamp +46
Cleaning up request 4 ID 4 with timestamp +46
Cleaning up request 5 ID 5 with timestamp +46
Cleaning up request 6 ID 6 with timestamp +46
Cleaning up request 7 ID 7 with timestamp +46
Ready to process requests.

Next RADIUS client retransmit in 3 seconds
EAPOL: SUPP_BE entering state RECEIVE
Received 171 bytes from RADIUS server
Received RADIUS message
RADIUS message: code=2 (Access-Accept) identifier=7 length=171
Attribute 26 (Vendor-Specific) length=58
Value: 00 00 01 37 11 34 87 57 06 27 23 b5 3c 69 7e 23 c8 60 45 05 71 6b 92 d9
ef 45 46 99 97 bf 01 ef 62 0b c9 6e 48 e2 6b a5 fd 03 b1 49 fc 01 13 a6 3a 6e 7b ec 0
0 8b 13 73
Attribute 26 (Vendor-Specific) length=58
Value: 00 00 01 37 10 34 8b a5 87 04 ba a1 f7 b7 89 70 1a 2a 1a f4 0f 04 39 1b
77 4c 57 87 d1 09 f4 09 13 fb 0b d0 61 e9 fd 7f 84 43 4c 06 9c a2 74 ac fa 45 60 28 9
4 cf d8 13
Attribute 79 (EAP-Message) length=6
Value: 03 07 00 04
Attribute 80 (Message-Authenticator) length=18
Value: 57 c8 5d 80 ec a3 14 36 d8 fc e7 80 10 3b 80 99
Attribute 1 (User-Name) length=11
Value: 'anonymous'
STA 02:00:00:00:00:01: Received RADIUS packet matched with a pending request, round t
rip time 0.00 sec
RADIUS packet matching with station
MS-MPPE-Send-Key (sign) - hexdump(len=32): 40 ce b9 a2 c8 39 86 f5 d5 4b 21 e3 16 96
5c 91 5d e1 82 f0 6e ee ca 12 84 f5 16 78 4f cd 32 fa
MS-MPPE-Recv-Key (crypt) - hexdump(len=32): 22 68 95 52 48 8f ec e4 54 31 9e 1f 39 37
36 3a 57 47 a4 40 8f 82 75 32 cf 91 f5 ae c8 80 89 30
decapsulated EAP packet (code=3 id=7 len=4) from RADIUS server: EAP Success
EAPOL: Received EAP-Packet frame
EAPOL: SUPP_BE entering state REQUEST
EAPOL: getSuppRep
EAPOL: EAP entering state RECEIVED
EAP: Received EAP-Success
EAP: EAP entering state SUCCESS
CTRL-Event-EAP-SUCCESS EAP authentication completed successfully
WPA: EAPOL processing complete
EAPOL: SUPP_BE entering state AUTHENTICATED
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state SUCCESS
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=1
EAPOL: Successfully fetched key (len=32)
PKM from EAPOL - hexdump(len=32): 22 68 95 52 48 8f ec e4 54 31 9e 1f 39 37 36 3a 57
47 a4 40 8f 82 75 32 cf 91 f5 ae c8 80 89 30
EAP: deinitialize previously used EAP method (21, TTLS) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 1 mismatch: 0
SUCCESS
[user@kanga-cs0lg09e ~]$

```

Figure 6: Connecting with valid credentials

```

user@kanga-cs0lg09e~
# Executing section authorize from file /etc/raddb/sites-enabled/inner-tunnel
-- entering group authorize (...)
++[chap] returns noop
++[mschap] returns noop
[suffix] No '9' in User-Name = "TomG", looking up realm NULL
[suffix] No such realm "NULL"
++[suffix] returns noop
++[control] returns noop
[eap] EAP packet type response id 0 length 9
[eap] No EAP Start, assuming it's an on-going EAP conversation
++[eap] returns updated
[files] users: Matched entry TomG at line 75
++[files] returns ok
++[expiration] returns noop
++[login] returns noop
[pap] WARNING: Auth-Type already set. Not setting to PAP
++[pap] returns noop
Found Auth-Type = Reject
Auth-Type = Reject, rejecting user
Failed to authenticate the user.
? # server inner-tunnel
[ttls] Got tunneled reply code 3
Reply-Message = "Oh no you don't Granger!"
[ttls] Got tunneled Access-Reject
[eap] Handler failed in EAP/TTLS
rlm_eap_ttls: Freeing handler for user TomG
[eap] Failed in EAP select
++[eap] returns invalid
Failed to authenticate the user.
Using Post-Auth-Type Reject
# Executing group from file /etc/raddb/sites-enabled/default
-- entering group REJECT (...)
[attr_filter.access_reject] expand: %(User-Name) -> anonymous
attr_filter: Matched entry DEFAULT at line 11
++[attr_filter.access_reject] returns updated
Delaying reject of request 6 for 1 seconds
Going to the next request
Waking up in 0.9 seconds.
Sending delayed reject for request 6
Sending Access-Reject of id 6 to 127.0.0.1 port 36776
EAP-Message = 0x04060004
Message-Authenticator = 0x00000000000000000000000000000000
Waking up in 3.9 seconds.
Cleaning up request 0 ID 0 with timestamp +2
Cleaning up request 1 ID 1 with timestamp +2
Cleaning up request 2 ID 2 with timestamp +2
Cleaning up request 3 ID 3 with timestamp +2
Cleaning up request 4 ID 4 with timestamp +2
Cleaning up request 5 ID 5 with timestamp +2
Waking up in 1.0 seconds.

Value: 127.0.0.1
Attribute 31 (Calling-Station-Id) length=19
Value: '02-00-00-00-00-01'
Attribute 12 (Framed-MTU) length=6
Value: 1400
Attribute 61 (NAS-Port-Type) length=6
Value: 19
Attribute 77 (Connect-Info) length=24
Value: 'CONNECT 11Mbps 802.11b'
Attribute 79 (EAP-Message) length=98
Value: 02 06 00 60 15 00 17 03 01 00 20 e8 db 44 ef 85 d1 ac f8 48 e4 1f 0d 93
2c 11 d5 a0 05 9a e7 75 f3 09 22 33 1f f6 b9 bf ff b0 8d 17 03 01 00 30 ca 36 cf e9 c
6 20 45 a1 1f 78 ee e9 60 cf 26 f1 ca 21 d7 21 39 92 b9 06 46 95 5d 65 fb 7e 3e 5c f9
d2 61 dc 7f 4f 34 3e 13 ff d9 07 bd f9 b8 59
Attribute 24 (State) length=18
Value: 01 c7 2a 98 04 c1 3f a4 d2 33 f5 f3 ef 91 4e db
Attribute 80 (Message-Authenticator) length=18
Value: f7 a1 75 13 d2 9e a9 0f 13 21 60 2c c6 41 5b 7f
Next RADIUS client retransmit in 3 seconds
EAPOL: SUPP_BE entering state RECEIVE
Received 44 bytes from RADIUS server
Received RADIUS message
RADIUS message: code=3 (Access-Reject) identifier=6 length=44
Attribute 79 (EAP-Message) length=6
Value: 04 06 00 04
Attribute 80 (Message-Authenticator) length=18
Value: 61 5f 1d 5f 9a cf 8c 8c f5 ff 96 2a 58 4b 1e 56
STA 02:00:00:00:00:01: Received RADIUS packet matched with a pending request, round t
rip time 1.00 sec
RADIUS packet matching with station
decapsulated EAP packet (code=4 id=6 len=4) from RADIUS server: EAP Failure
EAPOL: Received EAP-Packet frame
EAPOL: SUPP_BE entering state REQUEST
EAPOL: getSuppRep
EAP: EAP entering state RECEIVED
EAP: Received EAP-Failure
EAP: EAP entering state FAILURE
CTRL-Event-EAP-FAILURE EAP authentication failed
EAPOL: SUPP_BE entering state REJECT
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state FAIL
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=0
EAPOL: EAP key not available
EAP: deinitialize previously used EAP method (21, TTLS) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 0 mismatch: 1
FAILURE
[user@kanga-cs0lg09e ~]$

```

Figure 7: Connecting with invalid credentials

## B.2 wpa\_supplicant output

## B.3 802.11r output

```
<3>SME: Trying to authenticate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
<3>Trying to associate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
<3>SME: Trying to authenticate with 00:18:0a:01:3f:3f
      (SSID='notthebees' freq=2462 MHz)
<3>Trying to associate with 00:18:0a:01:3f:3f
      (SSID='notthebees' freq=2462 MHz)
<3>Associated with 00:18:0a:01:3f:3f
<3>CTRL-EVENT-EAP-STARTED EAP authentication started
...
<3>CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
<3>WPA: Key negotiation completed with 00:18:0a:01:3f:3f
      [PTK=CCMP GTK=CCMP]
<3>CTRL-EVENT-CONNECTED - Connection to 00:18:0a:01:3f:3f
      completed (reauth) [id=0 id_str=]
> roam 00:18:0a:01:3f:38
OK
<3>SME: Trying to authenticate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
<3>Trying to associate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
<3>Associated with 00:18:0a:01:3f:38
<3>CTRL-EVENT-EAP-STARTED EAP authentication started
...
<3>CTRL-EVENT-EAP-SUCCESS EAP authentication
      completed successfully
<3>WPA: Key negotiation completed with 00:18:0a:01:3f:38
      [PTK=CCMP GTK=CCMP]
<3>CTRL-EVENT-CONNECTED - Connection to 00:18:0a:01:3f:38
      completed (reauth) [id=0 id_str=]
```

Figure 8: Code snippet by Thomas Grainger



```
wlan0: Trying to associate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
FT: Invalid group cipher (0)
wlan0: Authentication with 00:18:0a:01:3f:38 timed out.
wlan0: Trying to associate with 00:18:0a:01:3f:3f
      (SSID='notthebees' freq=2462 MHz)
wlan0: Authentication with 00:18:0a:01:3f:3f timed out.
wlan0: Trying to associate with 00:18:0a:01:3f:3f
      (SSID='notthebees' freq=2462 MHz)
wlan0: Authentication with 00:18:0a:01:3f:3f timed out.
wlan0: Trying to associate with 00:18:0a:01:3f:38
      (SSID='notthebees' freq=2412 MHz)
^Cwlan0: CTRL-EVENT-TERMINATING - signal 2 received
```

Figure 9: Code snippet by Thomas Grainger