Realtek Ameba-1 802.1X authentication supplicant

This document describes how to use Ameba as 802.1X authentication supplicant. The examples illustrate the usage of three different EAP methods to authenticate with backend radius server.

Table of Contents

1 Introduction

This document shows how to use Ameba as 802.1X authentication supplicant. In 802.1X authentication, the supplicant (i.e., client device) is not allowed access through the access point device to the protected side of the network until the supplicant's identity has been validated by the authentication server (radius server). The authentication procedure is based on Extensible Authentication Protocol (EAP) framework. In the Ameba SDK, it provides three types of EAP methods: EAP-TLS, PEAPv0/EAP-MSCHAPv2, and EAP-TTLS/MSCHAPv2. User can choose one of the methods to authenticate with radius server according to the server side's setting. Following sections describe the related configuration files which can be used with EAP authentication and illustrate the example code usage.

2 Configuration

This section describes the configurations provided to meet different user requirements. Please note that you also need to include lib_eap, lib_wps, polarssl, and ssl_ram_map from build to make the EAP authentication work.

2.1 EAP Methods Configuration

Ameba SDK provides three types of EAP methods: EAP-TLS, PEAPv0/EAP-MSCHAPv2, and EAP-TTLS/MSCHAPv2. User can use one of these methods according to their requirements. To support each EAP method, the corresponding definition listed below which defined in platform_opts.h should be set to value 1. On the other hand, user can set the definition to value 0 to save the memory if they do not want to use the specified EAP method.

```
/* platform_opts.h */
#define CONFIG_ENABLE_TLS           1
#define CONFIG_ENABLE_PEAP          1
#define CONFIG_ENABLE_TTLS          1
```

In the EAP authentication procedure, it would contain a SSL/TLS connection established between the supplicant and the authentication server. The following definition determines that whether the client will verify the certificate sent by server during the SSL/TLS handshake process. User need to enable the definition and provide the certification authority (CA) certificate if they want to verify the server. The setting of CA certificate can be referred to Section 3.

```
/* platform_opts.h */
#define ENABLE_EAP_SSL_VERIFY_SERVER            1
```

2.2 PolarSSL Configurations

Some configuration files are provided to configure PolarSSL library to support the SSL/TLS connection during the authentication procedure. The following sub-sections show the PolarSSL configuration for supported cipher suites and the memory size of input/output buffer.

2.2.1 SSL/TLS Cipher Suite Configuration

PolarSSL library provides several cipher suites for SSL/TLS connection. To reduce the memory requirement, a configuration file named config_rsa.h is provided to only enable all the cipher suites of RSA-AES-SHA. Therefore, only cipher suites using RSA-AES-SHA will be proposed in SSL handshake. But if all the enabled cipher suites are not supported by the authentication server, user still can enable the cipher suites they need by modifying the definitions in config_rsa.h.

2.2.2 Configuration for private key decryption

For EAP-TLS authentication method, the device needs to send its certificate to server side during the SSL/TLS handshake. To do this, user should configure the certificate and private key in the SDK. If the private key is encrypted, user should provide further password (refer to Section 3.1). But for the current customized configuration, the SDK only enable the AES related algorithm to decrypt the key. If the private key is encrypted by DES related algorithm (e.g. Triple-DES), user could enable the definition as following to support it.

```
/* polarssl/config_rsa.h */
#define POLARSSL_DES_C
```

2.2.3 Memory Configuration

Although the SDK provides a customized configuration named config_rsa.h, user still can enable further cipher suites to meet their requirements. Some cipher suites will require large heap memory for SSL input/output buffer (up to 16384 bytes). Hence user might need to modify the definition of SSL_MAX_CONTENT_LEN as following which defines the size of input/output buffer. The value may need to be increased based on the cipher suite determined by server or the size of data transferred from server.

```
/* polarssl/config_rsa.h */
#define SSL_MAX_CONTENT_LEN          4096
```

2.3 Heap Configuration

As mentioned in Section 2.2.3, some cipher suites might require large heap memory for SSL input/output buffer. Hence the FreeRTOS heap may also be increased for SSL buffer allocation. User can change the FreeRTOS heap size by modifying the following definition which defined in FreeRTOSConfig.h.

```
/* FreeRTOSConfig.h */
#define configTOTAL_HEAP_SIZE      ( ( size_t ) ( 80 * 1024 ) )
```

3 Example Testing

The Ameba SDK provides an example file named example_eap.c which shows the basic usage of supplicant authentication methods. To run the example code, please make sure the following definition is enabled which defined in platform_opts.h.

```
/* platform_opts.h */
#define CONFIG_EXAMPLE_EAP      1
```

If the above definition has been enabled, an EAP authentication example thread will be started automatically when device booting. To change the used EAP method, please modify the argument of example_eap() function invoked in example_entry.c to a specified method.

```
/* example_entry.c */
#if CONFIG_EXAMPLE_EAP
	example_eap("tls");
	//example_eap("peap");
	//example_eap("ttls");
#endif
```

There are some variables also need to be set such as the target SSID and the client credential. These variables are listed below and they are set in example_eap_config() function within example_eap.c. Note that for different EAP methods, the requirement might be different. The following sub-sections show the required settings for each EAP method and the last sub-section shows the example of verifying server’s certificate during the handshake.

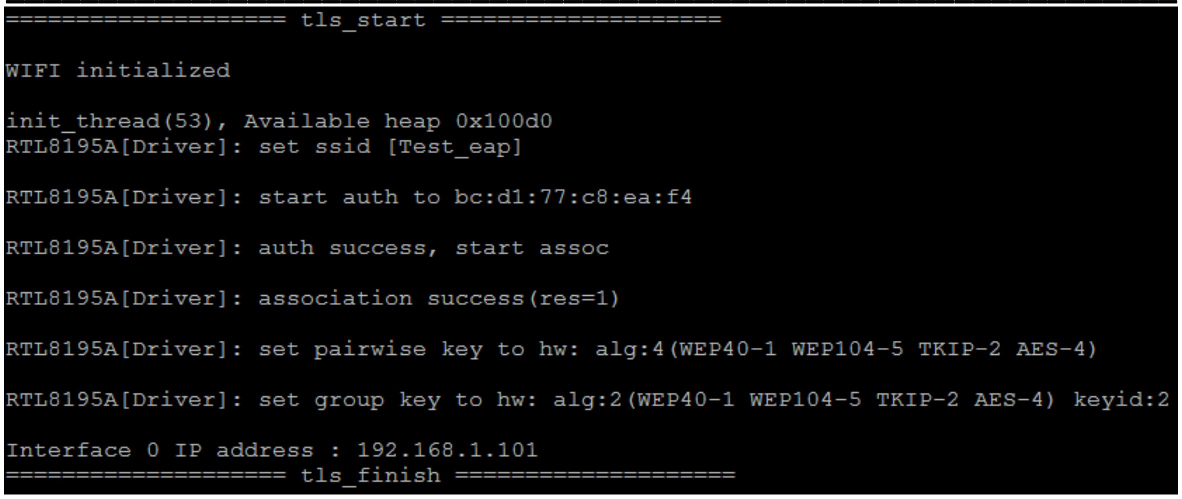
```
/* config variables in example_eap.c */
{
        eap_target_ssid,
        eap_identity,
        eap_password,
        eap_client_cert,
        eap_client_key,
        eap_client_key_pwd,
        eap_ca_cert
}
```

3.1 EAP-TLS

To use EAP-TLS in the example code, please make sure the corresponding definition has been enabled as we mentioned in Section 2.1. And the required connection settings are listed below. Make sure they are all set correctly except *eap_client_key_pwd* and *eap_ca_cert*. The variable *eap_client_key_pwd* is used only when the private key (restored in *eap_client_key*) is encrypted. And the variable *eap_ca_cert* is set only when you want to verify the server's certificate. Otherwise, just leave them and other non-required variables to NULL value.

```
/* EAP-TLS required config variables */
{
        eap_target_ssid,
        eap_identity,
        eap_client_cert,
        eap_client_key,
        (eap_client_key_pwd),
        (eap_ca_cert)
}
```

After setting these variables, build the SDK and download it to Ameba. The device will connect to the AP whose SSID matches *eap_target_ssid* automatically when device booting. And it will then process the EAP-TLS authentication flow with backend radius server. A screenshot of a successful connection example is showed below.



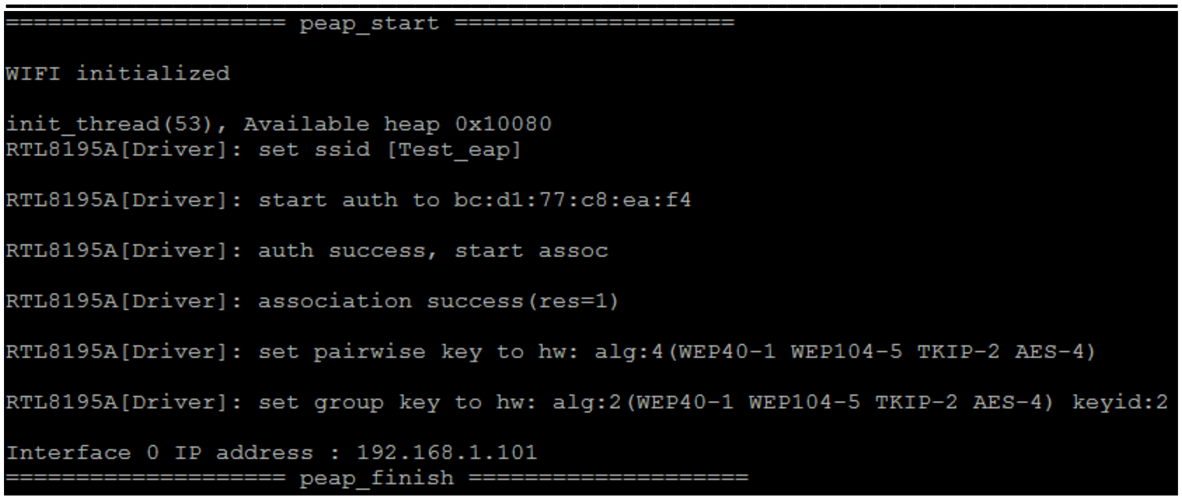
```
=================== tls_start ===================

WIFI initialized

init_thread(53), Available heap 0x100d0
RTL8195A[Driver]: set ssid [Test_eap]

RTL8195A[Driver]: start auth to bc:d1:77:c8:ea:f4

RTL8195A[Driver]: auth success, start assoc

RTL8195A[Driver]: association success(res=1)

RTL8195A[Driver]: set pairwise key to hw: alg:4(WEP40-1 WEP104-5 TKIP-2 AES-4)

RTL8195A[Driver]: set group key to hw: alg:2(WEP40-1 WEP104-5 TKIP-2 AES-4) keyid:2

Interface 0 IP address : 192.168.1.101
=================== tls_finish ===================
```

3.2 PEAP

To use PEAPv0/EAP-MSCHAPv2 in the example code, please make sure the corresponding definition has been enabled as we mentioned in Section 2.1. And the required connection settings are listed below. Make sure they are all set correctly except *eap_ca_cert*. The variable *eap_ca_cert* need to be set only when you want to verify the server's certificate. Otherwise, leave it and other non-required variables to NULL value.

```
/* PEAP required config variables */
{
        eap_target_ssid,
        eap_identity,
        eap_password,
        (eap_ca_cert)
}
```

After setting these variables, build the SDK and download it to Ameba. The device will connect to the AP whose SSID matches *eap_target_ssid* automatically when device booting. And it will then process the PEAP authentication flow with backend radius server. A screenshot of a successful connection example is showed below.



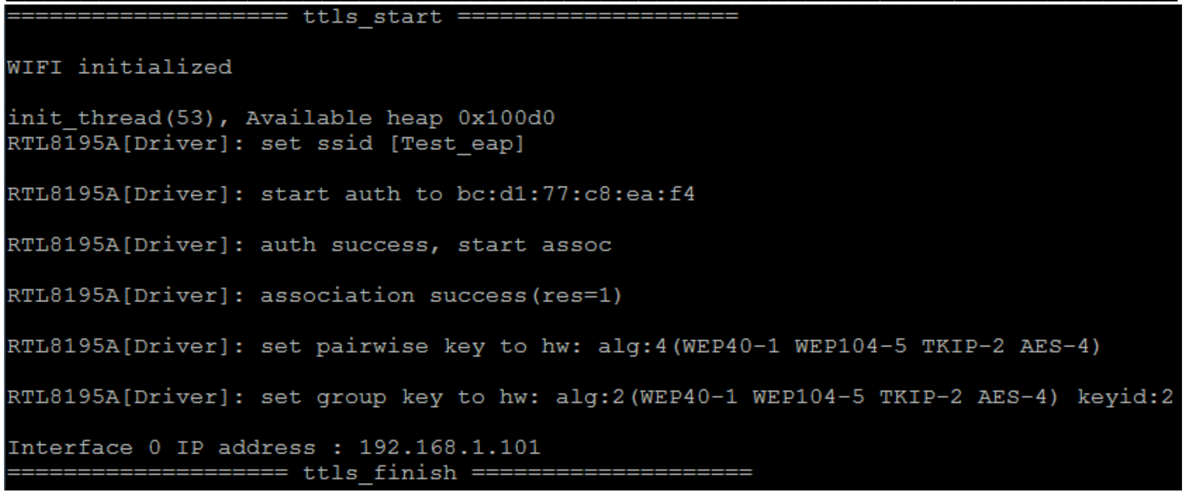
```
==================== peap_start ====================

WIFI initialized

init_thread(53), Available heap 0x10080
RTL8195A[Driver]: set ssid [Test_eap]

RTL8195A[Driver]: start auth to bc:d1:77:c8:ea:f4

RTL8195A[Driver]: auth success, start assoc

RTL8195A[Driver]: association success(res=1)

RTL8195A[Driver]: set pairwise key to hw: alg:4(WEP40-1 WEP104-5 TKIP-2 AES-4)

RTL8195A[Driver]: set group key to hw: alg:2(WEP40-1 WEP104-5 TKIP-2 AES-4) keyid:2

Interface 0 IP address : 192.168.1.101
==================== peap_finish ====================
```

3.3 EAP-TTLS

To use EAP-TTLS/MSCHAPv2 in the example code, please make sure the corresponding definition has been enabled as we mentioned in Section 2.1. And the required connection settings are listed below. Make sure they are all set correctly except *eap_ca_cert*. The variable *eap_ca_cert* need to be set only when you want to verify the server's certificate. Otherwise, leave it and other non-required variables to NULL value.

```
/* EAP-TTLS required config variables */
{
        eap_target_ssid,
        eap_identity,
        eap_password,
        (eap_ca_cert)
}
```

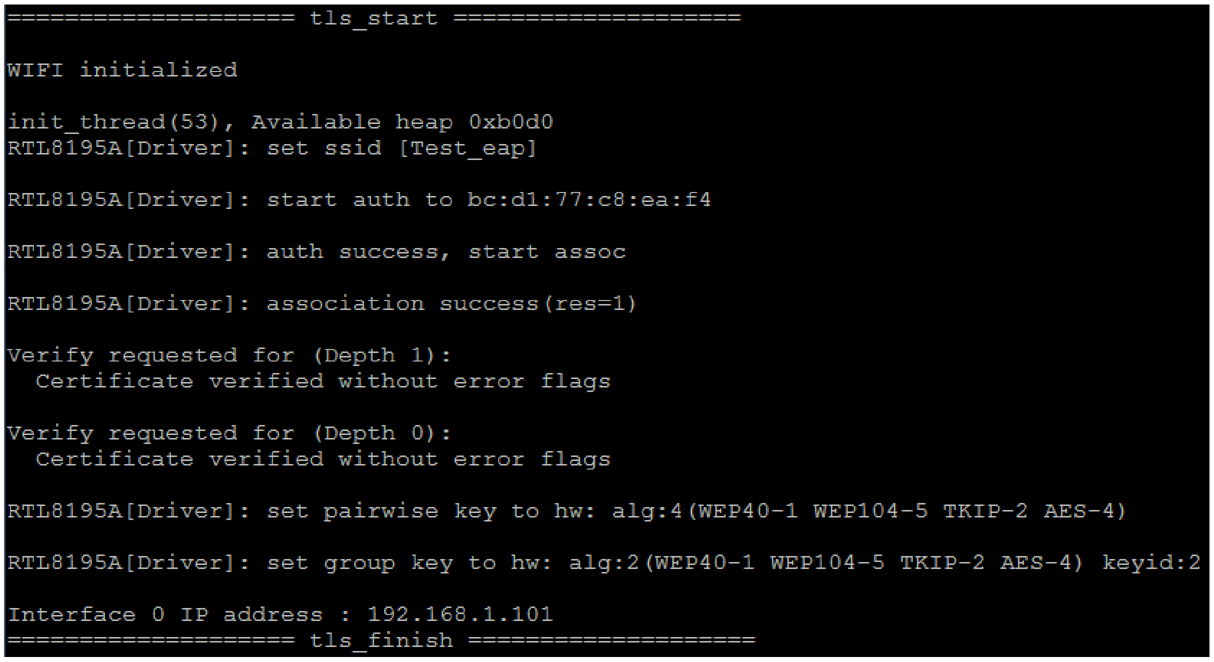
After setting these variables, build the SDK and download it to Ameba. The device will connect to the AP whose SSID matches *eap_target_ssid* automatically when device booting. And it will then process the EAP-TTLS authentication flow with backend radius server. A screenshot of a successful connection example is showed below.



```
=================== ttls_start ====================

WIFI initialized

init_thread(53), Available heap 0x100d0
RTL8195A[Driver]: set ssid [Test_eap]

RTL8195A[Driver]: start auth to bc:d1:77:c8:ea:f4

RTL8195A[Driver]: auth success, start assoc

RTL8195A[Driver]: association success(res=1)

RTL8195A[Driver]: set pairwise key to hw: alg:4(WEP40-1 WEP104-5 TKIP-2 AES-4)

RTL8195A[Driver]: set group key to hw: alg:2(WEP40-1 WEP104-5 TKIP-2 AES-4) keyid:2

Interface 0 IP address : 192.168.1.101
=================== ttls_finish ====================
```

3.4 Verify Server Certificate

As mentioned above, verifying server's certificate is an optional feature in the SDK. To use this feature, please make sure the corresponding definition has been enabled as we mentioned in Section 2.1 and set the value of CA certificate to *eap_ca_cert* in example_eap.c. After setting these configurations, the device will check the server's certificate during the SSL/TLS handshake.



```
=================== tls_start ====================

WIFI initialized

init_thread(53), Available heap 0xb0d0
RTL8195A[Driver]: set ssid [Test_eap]

RTL8195A[Driver]: start auth to bc:d1:77:c8:ea:f4

RTL8195A[Driver]: auth success, start assoc

RTL8195A[Driver]: association success(res=1)

Verify requested for (Depth 1):
  Certificate verified without error flags

Verify requested for (Depth 0):
  Certificate verified without error flags

RTL8195A[Driver]: set pairwise key to hw: alg:4(WEP40-1 WEP104-5 TKIP-2 AES-4)

RTL8195A[Driver]: set group key to hw: alg:2(WEP40-1 WEP104-5 TKIP-2 AES-4) keyid:2

Interface 0 IP address : 192.168.1.101
=================== tls_finish ====================
```